可復原式程式的邏輯
The Logic of Undoable Programs*

陳正佳
Cheng-Chia Chen

國立政治大學資訊科學系
Department of Computer Science,
National Chengchi University, Taipei, Taiwan, R.O.C.
chencc@cs.nccu.edu.tw

摘要

本文主要是在發展可用於推論可復原式程式行為的動態邏輯系統. 傳統命題式動態邏輯的語意架構並無法定義可復原式程式的語意，在本文中我們擴展了其語意架構使其足以定義可復原式程式的語意。此外，我們為可復原式程式發展了二套推論系統，證明其正確性與完備性。最後我們還證明兩系統下的可適性問題均屬 EXPTIME-complete。

關鍵詞：命題式動態邏輯，可復原式程式，推論系統，可適性問題，EXPTIME-complete.

Abstract

We develop the propositional dynamic logic (PDL) of undoable programs, which are general regular programs augmented with a construct "undo" that can be used to recover from undesirable program states. The language of the logic is essentially that of PDL augmented with an "undo" construct. In this paper we define the semantics of undoable PDL (UPDL) by refining the traditional semantical framework for PDL so that the meaning of "undo" can be treated correctly. Besides, since there are two conditions under which we are concerned with validity of assertions about undoable programs, we define two logics for UPDL. We give sound and complete axiom systems for both logics, and show that the satisfiability problems for both logics are EXPTIME-complete.

Keywords: logic of programs, propositional dynamic logic (PDL), undoable programs, satisfiability problem, EXPTIME-complete.

1. Introduction

Dynamic logic [5,7,8] is a family of logics used for reasoning about the I/O behavior of computer programs. The first order version of dynamic logic was first introduced in the fundamental work of Pratt[9] and developed further by Harel[4]. First order dynamic logic has high expressive power and is highly undecidable. For this reason Fischer and Ladner[3] introduced the propositional version of Dynamic logic(PDL), which is a propositional multi-modal logic with each modality corresponding to a regular program. Since then many variants of PDL have been introduced for reasoning about different kinds of computation[5,7,8].

The basic idea of PDL is to integrate programs into an assertional language by allowing every program to be a modal operator. So, for example, if A is a program and P is an assertion, then a new assertion [A]P can be formed, meaning that "P holds on termination of A". In addition to modal operators, the usual Boolean operations are allowed. The program constructors for PDL include ;, *, U, and ?. The constructor ";" stands for concatenation of programs; therefore, "A;B" means "first execute A and then execute B". "U" stands for nondeterministic choice; thus "AUB" means "choose A or B nondeterministically, and execute it". The constructor "*" is a nondeterministic looping operator and "A*" means "execute A a nondeterministically chosen number of times. Finally the constructor "?" is the test operation so that if P is a formula then "P?" is a program meaning that "test P, and proceed if true". PDL provides a powerful language for describing programs, their correctness and termination.

In this paper we are interested in extending PDL with one construct: "undo". "undo" has the effect of removing the effect of previous computation and returning to the previous state. The construct undo has been used popularly in interactive computing environments such as word processors, debuggers and various editing tools etc. For reasoning about and giving specification of programs (or scripts) used in these environments, traditional PDL is not expressive enough since the undo construct has effect not expressible by existing PDL frameworks.

For UPDL, there are two conditions under which we are concerned with validity of assertions about programs, one being valid in all initial states and the other valid on all states including also all intermediate states. Therefore we define two logics of UPDL, one for the former called $UPDL_I$ and the other for the later called $UPDL_G$. In this paper we develop sound and complete axiom systems for both logics. Moreover we show that the complexity of the satisfiability problems for both logics are no harder than that of standard PDL. In other words, both are EXPTIME-complete.

The remaining sections of this paper are organized as follows. In section 2 the language and syntax of the proposed logic is introduced. Section 3 covers the semantics and axiom systems of UPDL. In the following section we derive completeness and complexity results for the logic. Finally we discuss some alternative definitions and extensions

* Supported by NSC under the project: NSC85-2221-E004-001.

of UPDL that deserve further investigation.

2. The language and syntax of undoable PDL

The language of UPDL is based on a set of primitive programs Σ_0 and a set of propositional variables Δ_0. Two sorts of syntactic objects, namely, programs (denoted Σ) and well-formed formula (,denoted Δ), are defined in UPDL. They are defined by mutual recursion as follows:

1. Every atomic formula $p \in \Delta_0$ is a formula.
2. if P, Q are formulas, then so are $\neg P$ and $(P \wedge Q)$.
3. if A is a program and P is a formula, then $[A]P$ is a formula.
4. Every primitive program $a \in \Sigma_0$ is a program.
5. "undo" is a program.
6. if A, B are programs then so are A^*, $(A; B)$, and $(A \cup B)$.
7. if P is a formula, then $P?$ is a program.

Members of Σ and Δ are called UPDL programs and UPDL formula, respectively; members of Σ and Δ constructible without using clause (6) are called PDL programs and PDL formulas, respectively. Other connectives usually found in the literature are regarded as abbreviations. In particular, $\langle A\rangle P$ abbreviates $\neg[A]\neg P$, and *false* abbreviates $p \wedge \neg p$, where p is a particular atomic formula in Δ_0.

3. Semantic structures

First review the general definiton of *PDL-structure*:

Definition 1 *A PDL-structure M is a triple $\langle W, R, V\rangle$ where*

- *W is a set of (program states),*
- *$R : \Sigma_0 \rightarrow 2^{W\times W}$ is the I/O relation for primitive programs, which, for each primitive program a, assigns a relation $R(a)$ from W to W. Intentionally, $(s_1, s_2) \in R(a)$ means "executing atomic program a from state s_1 may possibly lead to state s_2." We say t is an a-child of s if $(s, t) \in R(a)$.*
- *$V : \Delta_0 \rightarrow 2^W$ is the valuation function, which assigns a set of states $V(p)$ for each atomic formula $p \in \Delta_0$ with the intention that for any state $s \in W$, $s \in V(p)$ iff p is true in state s.*

Let $R(\Sigma_0) = \cup_{a\in\Sigma_0} R(a)$. For any state $s.t \in W$, if $(s, t) \in R(\Sigma_0)$, we say t is a child of s (and s is a parent of t). If $(s, t) \in R(\Sigma_0)^+$, we say t is a descendant of s (and s is an ancestor of t). The PDL-structure M is said to be a *PDL-tree* if the relation $(W, R(\Sigma_0))$ is a rooted tree.

Since PDL-structure is too weak to define the semantics of undoable programs, here we adopt PDL-trees as the underlying structure for defining the semantics of undoable programs. First we define the PDL-trees generated by a PDL-structure M, which are simply unwinding trees generated from states of M.

Let s_0 be any state of W. The tree T_{s_0} of M starting from s_0 is a rooted tree with nodes labeled by states of W and edges labelled by primitive programs. T_{s_0} is defined as follows:

1. The root of the tree T is at level 0 and is labelled by s_0. For each node N, we use $l(N)$ to denote its label.
2. If N is a node at level k labelled by s, and t is an a-child of s in M, then there is one and only one a-child of N labeled by t.
3. N' is a child of N only if it satisfies condition 2.

The tree T_{s_0} can be extended to a PDL-structure $(W_{s_0}, R_{s_0}, V_{s_0})$ (since there is no worry of ambiguity, we also denoted it as T_{s_0}) where

- W_{s_0} is the set of all nodes of T_{s_0},
- $R_{s_0} : \Sigma_0 \rightarrow (W_{s_0} \times W_{s_0})$ s.t. $(n_1, n_2) \in R(a)$ iff n_2 is an a-child of n_1 in T_{s_0}.
- $V_{s_0} : \Delta_0 \rightarrow 2^{W_{s_0}}$ s.t. for every node n, and for every atomic formula p $n \in V_{s_0}(p)$ iff $l(n) \in V(p)$.

T_{s_0} is said to be the PDL-tree generated from s_0. Obviously T_{s_0} itself is a PDL-structure. Also note that the relation $R(\Sigma_0)$ in a PDL-tree is irreflexive and backward linear, i.e., for every state s. there is at most one state t with $(s, t) \in R(\Sigma_0)^{-1}$. For each PDL-structure $M = (W, R, V)$, let T_M denote the set of all PDL-trees generated from states of M.

Given a PDL-structure $M = (W, R, V)$, the denotations of PDL programs and formulas are defined by extending the domains of R and V to all PDL programs and formulas as follows:

1. $R(A; B) = R(A) \cdot R(B)$, where "$\cdot$" is the relation composition operation.
2. $R(A \cup B) = R(A) \cup R(B)$.
3. $R(A^*) = R(A)^*$, where "$*$" at the right hand side is the reflexive and transitive closure operation on relations.
4. $R(P?) = \{(s, s) \mid P \in V(s)\}$.
5. $V(P \wedge Q) = V(P) \cap V(Q)$
6. $V(\neg P) = W - V(P)$,
7. $V([A]P) = \{s \mid \forall t \in W$,if $(s, t) \in R(A)$ then $t \in V(P)\}$.

By the definition of $\langle A\rangle P$, clause (7) implies $V(\langle A\rangle P) = \{s \mid \exists t \in W, (s,t) \in R(A) \wedge t \in V(P)\}$.

The extended R and V are called the PDL-extensions of R and V, respectively. $R(A)$ and $V(P)$ are called the PDL-interpretation of A and P, respectively, in M. A PDL formula P is said to be satisfiable (or true) at a state s of a PDL-structure M, in symbols, $M, s \models P$, if $s \in R(P)$. If $R(P) \neq \phi$ (resp., $R(P) = W$), we say P is satisfiable (resp., valid) in M. Let $\mathcal{C}$ be a class of PDL-structures, we say P is satisfiable in $\mathcal{C}$ if P is satisfiable in some structure M of $\mathcal{C}$ and say P is valid in $\mathcal{C}$ if P is valid in all structures of $\mathcal{C}$. In particular if P is valid in all PDL-structures (resp., satisfiable in some PDL-structures), we say P is PDL-valid (resp., PDL-satisfiable).

To define the meaning of an undoable program A, possibly resulted from executing a program B at some initial state, we have to keep track of all past states, beginning from the initial one where B is started to the current one where the remaining part of B is A. PDL-structure alone certainly does not give us such information. However, if we interpret the root of a PDL-tree as the initial state of a program's computation and treat the path from the root to a node as the computation history of the program started from the initial state to the current state corresponding to the node, then the meaning of "undo" is simply "back to the parent". So we can define the meaning of all undoable programs and formulas on PDL-trees as follows:

Definition 2 *Let $M = (W, R, V)$ be a PDL-structure. We extend R and V to $\mathcal{R}$ and $\mathcal{V}$, respectively, as follows:*

Clause 1-7 are the same as those given in the defintion of PDL-extensions.

8. $\mathcal{R}(undo) = \{(t,s) \mid (s,t) \in R(\Sigma_0)\}$.

$\mathcal{R}$ and $\mathcal{V}$ together is called the UPDL-extension of R and V. Note that although the semantics of UPDL programs and formulas are defined only on PDL-trees via $\mathcal{R}$ and $\mathcal{V}$, for technical reason we allow $\mathcal{R}$ and $\mathcal{V}$ to be defined on all PDL-structures, as specified in Definition 2, where "undo" is treated as if it were the converse of the union of all primitive programs. In summary, though both the PDL-extension and the UPDL-extension are defined on all PDL-structures, the reader should keep in mind that only on PDL-trees does the UPDL extension define correctly the meaning of programs containing "undo".

The definition of the satisfaction of UPDL formulas on states of PDL-structures via $\mathcal{R}$ and $\mathcal{V}$ are analogous to that of PDL-formulas on PDL-structures via R and V. A UPDL formula P is said to be U-satisfied (or true) at a state s of a PDL-structure M, in symbols, $M, s \models_U P$, if $s \in \mathcal{R}(P)$. If $\mathcal{R}(P) \neq \phi$ (resp., $\mathcal{R}(P) = W$), we say P is U-satisfiable (resp., U-valid) in M. Let $\mathcal{C}$ be a class of PDL-structures, we say P is U-satisfiable in $\mathcal{C}$ if P is U-satisfiable in some structure M of $\mathcal{C}$ and said P is U-valid in $\mathcal{C}$ if P is U-valid in all structures of $\mathcal{C}$. In particular if P is U-valid in all PDL-trees (resp., U-satisfiable in some PDL-tree), we say P is globally valid (resp., globally satisfiable).

In addition to global validity, where we are concerned with validity of UPDL formulas evaluated at any state with arbitrary computation history, of interest is another validity, called initial validity, where we evaluate validity of formulas only one initial states (i.e., those where no state has passed). More formally, we say a UPDL formula P is initially valid if P is U-satisfied at the root state of every PDL-trees and say P is initially satisfiable if it is U-satisfied at the root state of some PDL-tree.

The first problem about both notions of validities is whether they define the same set of valid UPDL formulas. A simple observation shows that they don't. Obviously, every formula which is globally valid is initially valid, but the converse in general is not true. To show this, consider the formula $P = [undo]false$, Obviously P is U-satisfied at all initial states but not in any non-initial state. Hence $[undo]false$ is initially valid but not globally valid.

There are some relationships among PDL-trees, PDL-structures, and PDL-trees they generated. Due to space limit, they are omitted here. The interested readers are referred to Chen[1]. The main result is that, when restricted to PDL only, adopting PDL-structures or PDL-trees as underlying semantic structures does not change the sets of valid and satisfiable formulas.

Theorem 1 *Let P be any PDL-formula. Then*

1. *P is PDL-valid iff (P is valid in all PDL-trees iff) P is globally valid.*
2. *P is PDL-satisfiable iff (P is satisfiable in some PDL-trees iff) P is initially satisfiable.*

So by interpreting PDL and UPDL both on unifying PDL-trees, UPDL can be regarded as an extension of PDL both on syntax and on semantics.

4. Axiom systems for UPDL

The axiom systems for UPDL include the axioms used in PDL together with additional ones for dealing with "undo". Consider the following axiom schemes, where A,B are any UPDL programs and P and Q are any UPDL formulas,

A1: all tautologies of propositional calculus

A2: $[A](P \wedge Q) \leftrightarrow [A]P \wedge [A]Q$

A3: $[A;B]P \leftrightarrow [A][B]P$

A4: $[A \cup B]P \leftrightarrow [A]P \wedge [B]P$

A5: $[A^*]P \rightarrow P \wedge [A][A^*]P$

A6: $[A^*](P \rightarrow [A]P) \rightarrow (P \rightarrow [A^*]P)$

A7: $\langle a\rangle[undo]P \rightarrow P$, where a is any primitive program.

A8: $\langle undo\rangle[\Sigma_0]P \rightarrow P$, where $\Sigma_0 = \bigcup_{a\in\Sigma_0} a$.

A9: $\langle undo\rangle P \rightarrow [undo]P$

A10: $\langle undo^*\rangle[undo]false$

A11: $[undo]false$

and inference rules:

R1(MP): from P and $P \rightarrow Q$, infer Q.

R2(G): from P infer $[A]P$.

R3(G'): from P infer $[A]P$ with the proviso that P must be derivable without using A10.[1]

Axiom schemes (A1–A6) and rules (R1–R2) are the standard axiom system for PDL. Axiom 7 says that "undo" can return to the the present state from those that may be reached by executing any primitive program at the present state. Axiom 8 states that, by executing "undo", the program will goto the previous state, from which the current one must be reachable by executing some primitive program. Axiom 9 states that "undo" is deterministic, axioms 10 states that no histories of states are infinitely long. Finally axiom 11 says that the current state cannot be "undo" any more and hence is true of all initial states only.

Let $UPDL_I$ be the axiom system consisting of (A1–A11) and R1 and R3, and let $UPDL_G$ be the axiom system consisting of (A1–A10) and R1 and R2. Let provability of a formula P in $UPDL_I$ and $UPDL_G$ be denoted $\vdash_I P$ and $\vdash_G P$, respectively. A formula P is said to be consistent in an axiom system H if its negation is not a theorem of H (i.e. $\nvdash_H \neg P$.); a set of formulas F is said to be consistent if $\bigwedge F$ is consistent. It is noted that R3 is equivalent to the statement that if $\vdash_G P$ (hence P can be derived without A10) then $\vdash_I P$. Therefore, since A10 is not provable in $UPDL_G$, $UPDL_G$ is a system properly weaker than $UPDL_I$. Both provability relations are related by the following lemma[1].

Lemma 1 *Let P be any UPDL formulas. Then*

$$\vdash_I P \text{ iff } \vdash_G [undo]false \rightarrow P$$

It is easy to prove[1] that $UPDL_I$ and $UPDL_G$ are sound with respect to initial validity and global validity, respectively, defined in the previous section.

Theorem 2 *The axiom systems $UPDL_G$ and $UPDL_I$ are sound for global validity and initial validity of UPDL, respectively.*

We defer the completeness of both systems to the next section.

[1] I.e., there is a proof of P where A10 is not used.

5. Decidability and completeness of UPDL

In this section we are concerned with the decidability and completeness of the satisfiability problems for UPDL. We will show that both the initial satisfiability and the global satisfiability problems for UPDL can be solved in exponential time. Hence, together with the well-known lower bound result for PDL, the satisfiability problem for UPDL is shown to be EXPTIME-complete. Our proof is essential a combination of those techniques used in D. Harel and R. Sherman[6, 5, 10] and M. Ben-Ari et.al.[2]. First, as usual, we require the notions of the subformula closure of a formulas.

Definition 3 (Fischer-Ladner closure of formulas) *Let P_0 be a UPDL formula, the Fischer-Ladner closure of P_0, denoted $FL(P_0)$, is defined to be the least set S of formulas such that:*

1. *$P_0 \in S$, and $\langle undo^*\rangle[undo]false \in S$.*
2. *If $\neg P \in S$ then $P \in S$.*
3. *If $P \wedge Q \in S$, then $P, Q \in S$*
4. *If $[Q?]P \in S$ then $Q, P \in S$.*
5. *If $[A]P \in S$ then $P \in S$.*
6. *If $[A;B]P \in S$ then $[A][B]P \in S$*
7. *If $[A^*]P \in S$ then $P \in S$ and $[A][A^*]P \in S$.*
8. *If $[A \cup B]P \in S$ then $[A]P \in S$ and $[B]P \in S$.*
9. *$[undo]P \in S$ iff $\langle undo\rangle P \in S$.*

Let $\neg FL(P_0)$ be defined as $\{\neg P | P \in FL(P_0)\}$ and define $Z(P_0) = FL(P_0) \cup \neg FL(P_0)$.

For each formula P, let $|P|$ denote the number of symbols used for representing P; for a set S, let $|S|$ to denote the cardinality of the set. Analgous to Fischer and Ladner[3], it is easy to show that $|FL(P_0)|$ is bounded by the size of P_0.

Proposition 1 *If $|P_0| = n$ then $|FL(P_0)| = O(n)$.*

For easy of presentation, we identify every formula of the form $\neg\neg P$ with P in the rest of the section. Similar to that defined in [5, 10], certain sets of formulas from $Z(P_0)$ are called atoms, which are free of immediate inconsistencies.

Definition 4 *An atom for P_0 is a subset S of $Z(P_0)$ satisfying the following: for every program A, B and formula P, Q.*

1. *if $P \in Z(P_0)$, then $P \in S$ iff $\neg P \notin S$*
2. *if $P \wedge Q \in Z(P_0)$ then $P \wedge Q \in S$ iff $P \in S$ and $Q \in S$.*
3. *if $[P?]Q \in Z(P_0)$ then $[P?]Q \in S$ iff $P \notin S$ or $Q \in S$.*

4. *if* $[A;B]P \in Z(P_0)$ *then* $[A;B]P \in S$ *iff* $[A][B]P \in S$

5. *if* $[A \cup B]P \in Z(P_0)$ *then* $[A \cup B]P \in S$ *iff* $[A]P \in S$ *and* $[B]P \in S$

6. *if* $[A^*]P \in Z(P_0)$ *then* $[A^*]P \in S$ *iff* $P \in S$ *and* $[A][A^*]P \in S$

7. *if* $[undo]P \in Z(P_0)$ *then* $\langle undo\rangle P \in S$ *only if* $[undo]P \in S$.

8. *if* S *contains* $[undo]false$, *then* S *does not contains any formula of the form* $\langle undo\rangle P$

   If S *contains* $[undo]false$, *we say it is an initial atom, otherwise, we say it is a non-initial atom.*

Denote the set of atoms for P_0 by $AT(P_0)$; it is easy to prove (cf. Fischer and Ladner[3]) that $|AT(P_0)| \leq 2^{O(|P_0|)}$. If P_0 is a formula, let $\Sigma(P_0)$ be the set of all UPDL programs appearing in P_0 and let $\Delta_0(P_0)$ (resp., $\Delta(P_0)$) be the set of atomic formulas (resp., UPDL formulas) appearing in $Z(P_0)$.

In the remaining part of this section let P_0 be any specific UPDL formula. We are now interested in determining the global or initial satisfiability of the formula. Our goal is to construct a PDL-tree U-satisfying P_0 if it is U-satisfiable. As proceeded in Harel[10, 5] a sequence of PDL-structures $M_0, M_1, M_2, \cdots$ is defined in steps as follows: $M_0 = (W_0, R_0, V_0)$ is defined by

- $W_0 = AT(P_0)$.
- $R_0 : \Sigma_0(P_0) \to 2^{W_0^2}$ where, for each $a \in \Sigma_0$, $s,t \in W_0$, $(s,t) \in R_0(a)$ iff (1) t is a noninitial atom and (2) for every $[a]P \in s$, $P \in t$ and (3) for every $\langle undo\rangle P \in t$, $P \in s$.
- $V_0 : \Delta_0(P_0) \to 2^{W_0}$ where, for each $p \in \Delta_0(P_0)$ and for each $s \in W_0$, $s \in V_0(p)$ iff $p \in s$.

We extend $R_0 : \Sigma_0 \to 2^{W_0^2}$ to $R_0' : \Sigma(P_0) \to 2^{W_0^2}$ and $V_0 : \Delta_0(P_0) \to 2^{W_0}$ to $V_0' : \Delta(P_0) \to 2^{W_0}$ like the usual UPDL-extension defined in Definition 2 with the following exception:

$$R_0'(P?) = \{(s,s) \mid s \in W_0 \text{ and } P \in S\}$$

Recall that for the program "undo" we have

$$R_0'(undo) = \{(t,s) \mid (s,t) \in \cup_{a \in \Sigma_0} R(a)\}$$

It should be noted that M_0 in general is not a PDL-tree. Hence $R_0'(undo)$ may not be deterministic as we might expect. For $i \geq 0$, let $M_{i+1} = (W_{i+1}, R_{i+1}, V_{i+1})$ be given by

- $W_{i+1} = \{s \mid s \in W_i$ and for every $\langle A\rangle P \in s$ there exists $t \in W_i$ with $(s,t) \in R_i'(A)\}$.
- $R_{i+1} = R_i \cap W_{i+1}^2$
- $V_{i+1} = V_i \cap W_{i+1}$.

Let $R_{i+1}' : \Sigma(P_0) \to 2^{W_{i+1}^2}$ and $V_0' : \Delta(P_0) \to 2^{W_{i+1}}$ be the UPDL extension of R_{i+1} and V_{i+1} defined like R_0' and V_0'.

Since $AT(P_0)$ is finite, there is some $J < |AT(P_0)|$ where the construction closes up, i.e., $M_J = M_k$ for all $k \geq J$. Finally let $M = (W, R, V) = M_J$.

Lemma 2 *For every* $s \in W$, *for each* $\langle A\rangle P \in \Delta(P_0)$ $\langle A\rangle P \in s$ *iff* $\exists t \in W$ *s.t.* $(s,t) \in \mathcal{R}(A)$ *and* $P \in t$.

The above proof can be found in Chen[1]. It is essentially the same as that of Harel [10, 5, 6]. Unfortunately, since the constructed structure M in general is not a PDL-tree. the standard "undo" semantics may not be defined appropriately. Therefore, even there may exist some state U-satisfying P_0, we still cannot assure that P_0 is globally satisfiable. However, The following lemmas tell us that P_0 is globally satisfiable provided it is U-satisfiable at some state of the constructed M. The key point is that although M is not a PDL-tree, nodes of the PDL-trees generated from M and their corresponding labeling states in M agree on U-satisfaction of all formulas in $Z(P_0)$.

Lemma 3 *Let* u *be an initial atom* $u \in W$ *and* $T_u = (W_u, R_u, V_u)$. *Then for any* $s \in W_u$ *and for any* $P \in Z(P_0)$, $M, l(s) \models_U P$ *iff* $T_u, s \models_U P$

We now begin the completeness of $UPDL_I$ and $UPDL_G$ for the sets of initial and global valid UPDL formulas, respectively. Due to space restriction, only key lemmas and theorems are stated and their proofs can be found in Chen[1].

First let $\hat{s}$ denote $\wedge_{P \in s} P$ if s is a set of UPDL formulas. The following lemma show that only atoms appearing in W can be consistent in $UPDL_G$.

Lemma 4 *For every* $s \in AT(P_0)$, *if* $s \notin W$, *then* $\vdash_G \neg\hat{s}$.

Based on Lemma 4, we can show the following two results.

Lemma 5 *Let* $s \in W$, *if* s *is not reachable from any initial atom of* W, *then* $\vdash_G \neg\hat{s}$.

Lemma 6 1. P_0 *is globally satisfiable iff* $P_0 \in s$ *for some* $s \in W$ *reachable from an initial atom.*

2. P_0 *is initially satisfiable iff* $P_0 \in s$ *for some initial atom* $s \in W$.

As a result, we have the following theorems.

Theorem 3 *The axiom system* $UPDL_G$ *is complete; i.e., for every UPDL formula* P, *if* P *is globally valid then it is provable in* $UPDL_G$.

Theorem 4 *The axiom system* $UPDL_I$ *is complete; i.e., for every UPDL formula* P, *if* P *is initially valid then it is provable in* $UPDL_I$.

As to complexity, we have the following theorem.

Theorem 5 *1. The global satisfiability problem for UPDL is EXPTIME-complete.*

2. The initial satisfiability problem for UPDL is EXPTIME-complete.

Proof : (1) By Lemma 6 P is globally satisfiable iff $p \in s$ for some $s \in W$ reachable from an initial atom. As pointed out in [3], the construction of W can be carried out deterministically in exponential time. It is also easy to check in exponential time whether $P \in s$ for some atom s reachable from some initial atom. Hence the problem can be solved in exponential time. The hardness part is implied by the fact that even the PDL part is already EXPTIME-hard [3].

(2) Since by definition P initially satisfiable iff $P \wedge [undo]false$ is globally satisfiable, the initial satisfiability problem is reducible to the global satisfiability problem in polynomial time. Hence by (1) the initial satisfiability problem can also be determined in exponential time. $\bowtie$

6. Discussion

We have defined the semantics of "undo" as an error (i.e., undefined) when we try to undo at an initial state, i.e., at the root. This aspect of the meaning of "undo" is reflected on the axiom $A11 : [undo]false$. Another alternative for the semantics of "undo" at initial states is simply to let the "undo" have no effect at all initial states, namely, executing "undo" at an initial state will go to the same state. If we adopt such definition, the axiom schemes $A10 : \langle undo^* \rangle [undo]false$ and $A11 : [undo]false$ are no longer true and have to be replaced by two new ones $A10' : \langle undo^* \rangle([undo]P \rightarrow P)$ and $A11' : [undo]P \rightarrow P$. The resulting logic systems can be proved to be sound and complete for global and initial validities, respectively, under the new semantics. The complexity of both systems are the same as the old ones, i.e., both are EXPTIME-complete for the satisfiability problem. The method is essentially the same as what we have done for the old semantics, with only a few modifications corresponding to the differences between A10–A11 and A10'–A11'.

We can only undo the execution of *one* primitive program by using one "undo" each time. However, in many applications we need not only undo primitive programs but also need to undo any block of programs each time. To reason about such kind of programs, we require one additional construct which we call "atomize". "atomize" has the effect of encapsulating a compound program, which may produce many computation steps when executed, into a primitive one so that the "undo" construct cannot return to the intermediate states of the computation history produced by the compound program. Consider the program fragment:"$(a;b;b \cup c;d)$; if not OK then undo". Our intended function of the "undo" in the program is to recover to the beginning of the whole program once OK is not true, but, according to the current semantics of "undo", it can only return to the state corresponding to the execution of "a;b" or "c". By adding atomize to our logic, the intended function of "undo" can be achieved by reprogram the fragment as : "$at(a;b;b \cup c;d)$; if not OK then undo", where "at" encapsulates the compound program $a;b;b \cup c;d$ as if it were a primitive one so that "undo" will undo the whole program instead of just the last instruction of the program.

References

[1] Cheng-Chia Chen, Development of formal logics for modeling and reasoning about some aspects of the behavior of rational agents and programs, NSC technical report under project no:NSC85-2221-E004-001, 1997.

[2] M. Ben-Ari, J. Y. Halpern, and A. Pnueli, Deterministic propositional dynamic logic: finite models, complexity, and completeness, *Journal of Computer and Systems Science* 25:3, 1982, 402-417.

[3] M.J. Fischer and R.E. Ladner, Propositional dynamic logic of regular program, *Journal of Computer and System Science*, 18(2) 1979, 194–211

[4] D. Harel, First-order dynamic logic, LNCS Vol. 68, Springer-Verlag, Berlin, 1979.

[5] D. Harel, Dynamic logic, in Gabbay and Guenthner, eds.,*Handbook of Philosophical Logic II: Extensions of Classical Logic* , Reidel, 1984, 497–604.

[6] D. Harel, and R. Sherman, Dynamic logic of flowcharts, *Information and Control*, 64, 1985,119-135.

[7] D. Kozen and J. Tiuryn, Logic of Programs, in: J. van Leeuwen ed., *Handbook of Theorectical Computer Science*, Vol. B (Elsevier, Amsterdam. 1990) 789–840.

[8] R. Parikh, Propositional dynamic logic of programs: a survey, in: E. Engeler, ed., *Proc. Workshop on Logic of Programs*, Lecture Note in Computer Science, Vol. 125 (1981) 102–144.

[9] V.R. Pratt, Semantical considerations on Floyd-Hoare logic, *17th IEEE Symposium on Foundations of Computer Science*, , 1976, 109–121.

[10] R. Sherman and D. Harel, A combined proof of one exponential decidability and completeness for PDL, in *1st Int. Workshop on Found. Theoret. Comput. Sci.*, GTI, Paderborn, Germany, 1983, 221-233.