Dynamic logic of propositional assignments: a well-behaved variant of PDL

Philippe Balbiani
Université de Toulouse and CNRS, IRIT
Toulouse, France
Email: Philippe.Balbiani@irit.fr

Andreas Herzig
Université de Toulouse and CNRS, IRIT
Toulouse, France
Email: Andreas.Herzig@irit.fr

Nicolas Troquard
LOA-ISTC-CNR
Trento, Italy
Email: troquard@loa.istc.cnr.it

***Abstract*—We study a version of Propositional Dynamic Logic (PDL) that we call Dynamic Logic of Propositional Assignments (DL-PA). The atomic programs of DL-PA are assignments of propositional variables to true or to false. We show that DL-PA behaves better than PDL, having e.g. compactness and eliminability of the Kleene star. We establish tight complexity results: both satisfiability and model checking are EXPTIME-complete.**

I. Introduction

Dynamic logics are logics to reason about imperative programs. Their language extends that of propositional logic by modalities $\langle\pi\rangle$, one per program π. Programs are either atomic or complex, the latter being built by means of sequential and nondeterministic composition, test and iteration ('Kleene star'). The models of PDL are transition systems: each transition is labeled with the name of an atomic program and indicates the possible execution of an atomic program from one state to another. The modal formula $\langle\pi\rangle\varphi$ is true at state s if there is an execution of π from s leading to a state satisfying φ.

The basic dynamic logic is Propositional Dynamic Logic (PDL). Its atomic programs are *abstract*: they are just letters $a, b, \ldots$ from some alphabet. Thus "PDL abstracts away from the nature of the domain of computation and studies the pure interaction between programs and propositions" [1, p. 147]. In contrast, the atomic programs of first-order dynamic logic are concrete, viz., assignments of object variables to terms [1, ch. 11]. They capture assignments as used in programming languages. We are here interested in a different kind of assignments: *assignments of propositional variables to truth values*, for short: *propositional assignments*. We write $+p$ and $-p$ for such assignments, where p is a propositional variable.

Only few authors studied such propositional assignments. The first were Meyer and Winklmann, who considered both assignments of object variables and assignments of propositional variables [2]. Tiomkin and Makowsky focused on propositional assignments [3]. They augmented the language of PDL with two kinds of propositional assignments: local and global. Local assignments modify the truth value of a propositional variable at the actual state of a Kripke model and leave its value unchanged elsewhere. Global assignments modify the value of a propositional variable everywhere in a Kripke model. Besides these two distinct kinds of assignments, their logic still contains abstract atomic programs $a, b, \ldots$ They establish that the logic embeds first-order dynamic logic [3]. It is perhaps due to the ensuing undecidability result that dynamic logics with propositional assignments did not gain much traction, despite their putative usefulness for modelling concrete properties of computational systems. A notable exception is Wilm's decidable Propositional Program Logic PPL [4]. PPL is a dynamic logic with deterministic atomic programs (alias 'Strict PDL'), plus nondeterministic composition "$\cup$", plus Tiomkin and Makowsky's global propositional assignments. (So PPL has abstract atomic programs just as Tiomkin and Makowsky's logic.) Wilm showed that the Kleene star operator can be simulated in PPL.

More recently, dynamic logics with assignments but *without* abstract programs were investigated, chiefly in logics of agents, as variants of dynamic epistemic logics (DEL). The basic version of DEL is Public Announcement Logic (PAL). PAL extends multi-agent epistemic logic with dynamic operators whose arguments are truthful public announcements of propositions. Epistemic riddles and cryptographic protocols were successfully modelled in PAL (see e.g. [5]). It was extended by *public assignments* in [6]–[8], which model how agents' knowledge changes when some propositional variable is publicly assigned to true or to false. None of these papers explored the program operators of PDL. Again, the designated culprit is an issue of computability: Miller and Moss showed that the addition of the PDL program connectives to PAL yields an undecidable logic [9].

In this paper we revisit the logic of propositional assignments. Contrarily to Tiomkin and Makowsky, our language has no abstract programs; Contrarily to the approaches in the PAL tradition, it does not have epistemic operators. We call our logic *Dynamic Logic of Propositional Assignments*, abbreviated DL-PA. We present it in Section II. We briefly compare it to PDL in Section III, showing that it can be viewed as an instantiation of PDL with assignment programs.

The star-free version of DL-PA was previously studied by van Eijck [10] and recently put to use in [11]. Van Eijck gave an axiomatisation of star-free DL-PA and stated that the addition of the Kleene star does not increase the expressivity because "an arbitrary [program] π^* only affects a finite number of atomic propositions". He however observes that "[f]inding an efficient method for translating π^* [programs] into *-free form is another matter". In Section IV we show that the Kleene star can be constructively eliminated in DL-PA, providing thus a procedure reducing DL-PA to its star-free fragment. From this, decidability of the satisfiability problem follows. Our result contrasts with both Miller and Moss's undecidability result for the extension of PAL by the PDL program connectives and with Tiomkin and Makowsky's undecidability result for the extension of PDL by local assignments.

But the decidability of DL-PA is not to be doubted given the nature of its models: its semantics does not require a transition system, given that there is no need for an accessibility relation interpreting abstract actions. Models are in fact made up of a single state, i.e., a valuation of classical propositional logic. Propositional assignments then *update* these valuations in the obvious way. Being able to do PDL-style reasoning whilst relying on such 'degenerate', succinctly specified models, has an immediate practical advantage when modelling a computational system. Indeed, it is folklore that modelling a distributed system is typically achieved by a transition system of size exponential in the number of variables. The applicability of verification via model checking is then immediately compromised: a system designer cannot realistically be expected to even represent the model, let alone verify it. Instead, the modelling of a system in view of verification with DL-PA is confined to the initial state of affairs. Naturally, quite some modelling effort is shifted to the query, which needs to represent the dynamic constraints of the relevant variables. But it happens that it often does not impact much the size of the whole input to a model checking procedure. We exemplify this by efficiently mapping the instances of a variant of a decision problem concerning the game Peek [12] into instances of the DL-PA model checking. Stockmeyer and Chandra showed that deciding the former is provably difficult. Thus, unfortunately, it also means that there is a price to pay for this comparatively effortless modelling task: the computational complexity of model checking with DL-PA is EXPTIME-complete. In contrast, the complexity of model checking a transition system with PDL is PTIME-complete. As we shall show, the complexity of DL-PA satisfiability checking remains EXPTIME-complete, which is also the complexity of PDL.[1] Under a commonly accepted conjecture in complexity theory, this indicates that van Eijck's problem does not have a full solution: there is probably no efficient method to eliminate the Kleene star from DL-PA. We introduce Peek in Section V, and provide the lower bounds for the model checking and satisfiability checking problems. Section VI addresses the upper bounds.

In Section VII we discuss the extensions of DL-PA with assignments of any formula as atomic programs (and not just true and false) and with converse as a new program construct. In Section VIII we investigate the tight relationship of DL-PA with Coalition Logic of Propositional Control and Delegation (DCL-PC) [13], correcting a mistake in [13] about the complexity of DCL-PC.

[1]This corrects an error in the published version of [11] that was already signaled by an erratum of a few lines on the IJCAI website.

II. DL-PA: dynamic logic of propositional assignments

In this section we define syntax and semantics of dynamic logic of propositional assignments DL-PA.

A. Language

Throughout the paper, $\mathbb{P} = \{p, q, \ldots\}$ denotes a (fixed) countable set of propositional variables. The language of DL-PA is defined by the following grammar:

$$\begin{array}{lll} \pi & ::= & +p \mid -p \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \varphi? \\ \varphi & ::= & p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \end{array}$$

where p ranges over $\mathbb{P}$. So *atomic programs* are of the form $+p$ or $-p$. In a few places we will use the expression $\pm p$ to talk economically about $+p$ and $-p$. The operators of sequential composition (";"), nondeterministic composition ("$\cup$"), unbounded iteration ("*", the Kleene star), and test ("?") are familiar from PDL.

The program skip abbreviates $\top$? ("nothing happens"). We use the abbreviation π^n with the obvious meaning. We also use bounded iteration "$\pi^{\leq n}$", i.e., iteration up to integer n, as a macro for $\bigcup_{k \leq n} \pi^k$. A program is said to be *sequential* if it is built up from atomic programs and tests by means of the operator ";".

We abbreviate the logical connectives $\wedge$, $\rightarrow$ and $\leftrightarrow$ in the usual way. Aside the dynamic operator $\langle\pi\rangle$, we also use its dual: $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$.

The *length* of a formula φ, noted $|\varphi|$, is the number of symbols used to write down φ in the primitive language, without $\langle$, $\rangle$, and parentheses. For example, $|q \wedge r| = |\neg(\neg q \vee \neg r)| = 6$. The length of a program π, noted $|\pi|$, is defined in the same way. For example, $|-p; p?| = 5$. Also $|\langle +q\rangle(q \wedge r)| = 2{+}6 = 8$.

We define $\mathbb{P}_\varphi$ to be the set of variables from $\mathbb{P}$ occurring in formula φ, and we define $\mathbb{P}_\pi$ to be the set of variables from $\mathbb{P}$ occurring in program π. For example, $\mathbb{P}_{-p\cup+q} = \{p, q\} = \mathbb{P}_{\langle -p\rangle q}$.

B. Semantics

Models of DL-PA are nothing but models of classical propositional logic.

Definition 1. *A* valuation *is a subset of the set of propositional variables $\mathbb{P}$. The* size *of a valuation v is the cardinality of v.*

DL-PA programs are interpreted by means of a (unique) *relation between valuations*: atomic programs $+p$ and $-p$ update valuations in the obvious way, and complex programs are interpreted just as in PDL by mutual recursion. Table I gives the interpretation of the DL-PA connectives.

Definition 2. *A formula φ is* DL-PA valid *if $||\varphi|| = 2^{\mathbb{P}}$, and φ is* DL-PA satisfiable *if $||\varphi|| \neq \emptyset$.*

DL-PA validity and DL-PA satisfiability are defined as usual. For example, the formulas $\langle +p\rangle\top$ and $\langle +p\rangle\varphi \leftrightarrow \neg\langle +p\rangle\neg\varphi$ are DL-PA valid. Other examples of DL-PA validities are $\langle +p\rangle p$ and $\langle -p\rangle\neg p$. Observe that if p does not occur in φ then both $\varphi \rightarrow \langle +p\rangle\varphi$ and $\varphi \rightarrow \langle -p\rangle\varphi$ are valid. This is due to the following semantical property that we will use a few times in this paper.

$$\begin{aligned}
||+p|| &= \{(v,v') \ : \ v' = v \cup \{p\}\} \\
||-p|| &= \{(v,v') \ : \ v' = v \setminus \{p\}\} \\
||\pi;\pi'|| &= ||\pi|| \circ ||\pi'|| \\
||\pi \cup \pi'|| &= ||\pi|| \cup ||\pi'|| \\
||\pi^*|| &= \bigcup_{k \in \mathbb{N}_0} (||\pi||)^k \\
||\varphi?|| &= \{(v,v) \ : \ v \in ||\varphi||\} \\
||p|| &= \{v \ : \ p \in v\} \\
||\top|| &= 2^{\mathbb{P}} \\
||\bot|| &= \emptyset \\
||\neg\varphi|| &= 2^{\mathbb{P}} \setminus ||\varphi|| \\
||\varphi \vee \psi|| &= ||\varphi|| \cup ||\psi|| \\
||\langle\pi\rangle\varphi|| &= \{v \ : \ \text{there is } v' \text{ s.t. } (v,v') \in ||\pi|| \text{ and } v' \in ||\varphi||\}
\end{aligned}$$

TABLE I
Interpretation of the DL-PA connectives

Proposition 1. *Suppose $p \notin \mathbb{P}_\varphi$, i.e., p does not occur in φ. Then $v \cup \{p\} \in ||\varphi||$ iff $v \setminus \{p\} \in ||\varphi||$.*

The next proposition shows that the model checking problem can be reduced in logarithmic space to the satisfiability problem.

Proposition 2. *For every valuation v and formula φ, $v \in ||\varphi||$ iff the formula*

$$\varphi \wedge \left(\bigwedge_{p \in \mathbb{P}_\varphi \cap v} p \right) \wedge \left(\bigwedge_{p \in \mathbb{P}_\varphi \setminus v} \neg p \right)$$

is DL-PA *satisfiable.*

Definition 3. *Two formulas φ and φ' are* formula-equivalent *(or* logically equivalent*) if $||\varphi|| = ||\varphi'||$, i.e., if $\varphi \leftrightarrow \varphi'$ is* DL-PA *valid. Two programs π and π' are* program-equivalent*, in symbols $\pi \equiv \pi'$, if $||\pi|| = ||\pi'||$.*

C. Example: the n-bit counter

Here is a DL-PA program implementing an n bit counter. We need n propositional variables $q_0, \ldots, q_{n-1}$. The conjunction $\mathsf{time}(0) = \neg q_{n-1} \wedge \cdots \wedge \neg q_1 \wedge \neg q_0$ encodes that the counter has value zero; $\mathsf{time}(1) = \neg q_{n-1} \wedge \cdots \wedge \neg q_1 \wedge q_0$ encodes value one; $\mathsf{time}(2^n - 1) = q_{n-1} \wedge \cdots \wedge q_1 \wedge q_0$ encodes 2^n-1. Let $\mathsf{time}(N)$ encode that the counter has value N.

Here is a program incr_n incrementing the n bit counter until 2^n-1 is attained and then fails:

$$\begin{aligned}
\mathsf{incr}_n = {} & \neg(q_{n-1} \wedge \cdots \wedge q_0)?; \\
& \left(\bigcup_{0 \le k \le n-1} (\neg q_k \wedge q_{k-1} \cdots \wedge q_0?; +q_k; -q_{k-1}; \cdots; -q_0) \right)
\end{aligned}$$

Observe that the length of incr_n is quadratic in n. Observe moreover that the formula

$$\mathsf{time}(0) \to [\mathsf{incr}_n{}^*] \bigwedge_{N < 2^{n-1}} (\mathsf{time}(N) \to (\langle\mathsf{incr}_n\rangle\top \wedge [\mathsf{incr}_n]\mathsf{time}(N+1)))$$

and the formula

$$[\mathsf{incr}_n{}^*](\mathsf{time}(2^n-1) \to [\mathsf{incr}_n]\bot))$$

are both DL-PA valid. The reader may check that the counter indeed runs from zero to 2^n-1 and then fails: the formula

$$\neg q_{n-1} \cdots \wedge \neg q_0 \to \langle\mathsf{incr}_n{}^*\rangle(q_{n-1} \cdots \wedge q_0 \wedge [\mathsf{incr}_n]\bot)$$

is DL-PA valid.

III. Relation with PDL

Let us briefly recall PDL. Its grammar has exactly the same form as that of our DL-PA, except that PDL has a countable set Π_0 of abstract atomic programs instead of the DL-PA assignments $+p$ and $-p$. The models of PDL are triples $M = (W, R, V)$ where W is a nonempty set of states, $R : \Pi_0 \longrightarrow 2^{W \times W}$ associates with every atomic program $\pi_0 \in \Pi_0$ an accessibility relation $R(\pi_0)$ on W, and $V : W \longrightarrow 2^{\mathbb{P}}$ associates with every state a subset of $\mathbb{P}$. We write $M, w \Vdash \varphi$ when the PDL formula φ is true at state w of model M.

For the sake of comparison, suppose the set of atomic programs Π_0 contains $\{+p \ : \ p \in \mathbb{P}\} \cup \{-p \ : \ p \in \mathbb{P}\}$. Then the set of DL-PA validities obviously contains the set of PDL validities. To see this, build the Kripke model $M^{\text{DL-PA}} = (W, R, V)$ such that $W = 2^{\mathbb{P}}$, V is the identity function, and such that if π_0 is of the form $+p$ or $-p$ then $(v_1, v_2) \in R(\pi_0)$ iff $(v_1, v_2) \in ||\pi_0||$; we can prove that for every valuation v and DL-PA formula φ we have $v \in ||\varphi||$ iff $M^{\text{DL-PA}}, v \Vdash \varphi$. So when $\varphi \leftrightarrow \varphi'$ is PDL valid then φ and φ' are formula-equivalent in DL-PA. The converse does not hold: for instance, $\langle +p \rangle\top \leftrightarrow \top$ is valid in DL-PA, but not in PDL.

As to program equivalence, let us say that programs π and π' are *program-equivalent in* PDL if for every model $M = (W, R, V)$ we have $R(\pi) = R(\pi')$. Clearly, when π and π' are program-equivalent in PDL then π and π' are also program-equivalent in DL-PA. Again, the converse does not hold; to witness, $+p; +q$ and $+q; +p$ are program equivalent in DL-PA, but not in PDL.

We are going to show in Sections V and VI that both the model checking and the satisfiability problem of DL-PA have the same complexity as the satisfiability problem of PDL. The proof involves a polynomial embedding of DL-PA into PDL (Section VI, Proposition 16).

IV. Constructive eliminability of the Kleene star

In this section we show how dynamic operators can be eliminated from formulas. The key step is the elimination of the star operator. This allows us to establish decidability of satisfiability (Section IV-C) and compactness (Section IV-D).

A. Some remarkable properties

We now state some properties of DL-PA that already hold for PDL. We say that a program is *star-free* if it does not contain the Kleene star "*". We recall that in star-free PDL, the program operators can be entirely eliminated from formulas.

Proposition 3. *The following formula equivalences are* DL-PA *valid.*

$$\langle \pi;\pi' \rangle \varphi \leftrightarrow \langle \pi \rangle \langle \pi' \rangle \varphi$$
$$\langle \pi \cup \pi' \rangle \varphi \leftrightarrow \langle \pi \rangle \varphi \vee \langle \pi' \rangle \varphi$$
$$\langle \psi? \rangle \varphi \leftrightarrow \psi \wedge \varphi$$
$$\langle \pi^{\leq 0} \rangle \varphi \leftrightarrow \varphi$$
$$\langle \pi^{\leq n+1} \rangle \varphi \leftrightarrow \varphi \vee \langle \pi \rangle \langle \pi^{\leq n} \rangle \varphi$$

Program equivalence is a congruence relation w.r.t. ';' and '$\cup$': if $\pi_1 \equiv \pi'_1$ and $\pi_2 \equiv \pi'_2$ then both $\pi_1;\pi_2 \equiv \pi'_1;\pi'_2$ and $\pi_1 \cup \pi_2 \equiv \pi'_1 \cup \pi'_2$.

Proposition 4. *The following program equivalences are* DL-PA *valid.*

$$\pi\,;(\pi';\pi'') \equiv (\pi;\pi')\,;\pi'' \quad (1)$$
$$\pi\,;(\pi' \cup \pi'') \equiv (\pi;\pi') \cup (\pi;\pi'') \quad (2)$$
$$(\pi \cup \pi')\,;\pi'' \equiv (\pi;\pi'') \cup (\pi';\pi'') \quad (3)$$
$$\varphi?\,;\psi? \equiv \varphi \wedge \psi? \quad (4)$$
$$\texttt{skip};\pi \equiv \pi \quad (5)$$
$$\pi\,;\varphi? \equiv \langle \pi \rangle \varphi?\,;\pi \quad \textit{if } \pi \textit{ is sequential} \quad (6)$$
$$(\varphi?;\pi) \cup \pi \equiv \pi \quad (7)$$
$$\varphi? \cup \neg\varphi? \equiv \texttt{skip} \quad (8)$$
$$\pi^{\leq 0} \equiv \texttt{skip} \quad (9)$$
$$\pi^{\leq n+1} \equiv \texttt{skip} \cup \pi;\pi^{\leq n} \quad (10)$$

Proposition 5. *In* DL-PA*, if φ and φ' are formula-equivalent then $\varphi?$ and $\varphi'?$ are program-equivalent.*

Proposition 6. *In* DL-PA*, if π and π' are program-equivalent then $\langle \pi \rangle \varphi$ and $\langle \pi' \rangle \varphi$ are formula-equivalent.*

Proposition 7. *Let ϵ be an expression (either a formula or a program). Let π be a program occurring in ϵ. Let ϵ' be obtained from ϵ by replacing some occurrence of π by π'. If π and π' are program-equivalent then ϵ and ϵ' are (program or formula) equivalent.*

Proof: The proof is a routine induction on the length of the expression ϵ. For the case $\epsilon = \varphi?$ we use Proposition 5, and for the case $\epsilon = \langle \pi \rangle \varphi$ we use Proposition 6. ■

We call a program a *conditioned sequence of assignments* if it is of the form

$$\varphi?;\pi_1;\cdots;\pi_n, \text{ for } n \geq 0$$

where every π_k is an atomic program. When $n = 0$ we identify the sequence with $\varphi?$.

Proposition 8. *In* DL-PA*, every star-free program π is program-equivalent to the nondeterministic composition of conditioned sequences of assignments.*

Proof: We start by distributing ";" over '$\cup$', applying the program equivalences (2) and (3) of Proposition 4. The result is a nondeterministic composition of sequential programs. In each of these sequences we shift tests to the left and then group together test sequences into a single test, applying the program equivalences (6) and (4) of Proposition 4. The result is a nondeterministic composition of conditioned assignment sequences of the form

$$(\varphi_1?;\pi_1) \cup \cdots \cup (\varphi_n?;\pi_n)$$

where each π_k is a (possibly empty) sequence of atomic programs. ■

The next proposition is pivotal: it states a sufficient condition for the eliminability of the Kleene star.

Proposition 9. *Let π be a program of the form $\varphi_1?;\pi_1 \cup \cdots \cup \varphi_n?;\pi_n$ such that for every $k, m \leq n$ we have $\pi_k;\pi_m \equiv \pi_m$. Then $\pi^* \equiv \pi^{\leq n}$.*

Proof: It suffices to prove that $\pi^{\leq n+1} \equiv \pi^{\leq n}$. Consider any sequence of length $n+1$; such a sequence necessarily contains a repetition of some $\varphi_k?;\pi_k$. Take for example the following sequence σ of length $n+1$ wherein $\varphi_1?;\pi_1$ occurs twice:

$$\sigma = \varphi_1?;\pi_1;\varphi_2?;\pi_2;\varphi_3?;\pi_3;\cdots;\varphi_n?;\pi_n;\varphi_1?;\pi_1.$$

We show that σ is subsumed by $\pi^{\leq n}$, in the sense that $\sigma \cup \pi^{\leq n} \equiv \pi^{\leq n}$. Our proof makes extensive use of Proposition 7.

We first use the program equivalence $\pi_1;\varphi_2? \equiv \langle \pi_1 \rangle \varphi_2?\,;\pi_1$ (which is an instance of program equivalence (6) of Proposition 4) to push the second test to the left: we rewrite σ to

$$\varphi_1?;\langle \pi_1 \rangle \varphi_2?\,;\pi_1;\pi_2;\varphi_3?;\pi_3;\cdots;\varphi_n?;\pi_n;\varphi_1?;\pi_1.$$

Pushing the third test leftwards we obtain:

$$\varphi_1?;\langle \pi_1 \rangle \varphi_2?;\langle \pi_1;\pi_2 \rangle \varphi_3?\,;\pi_1;\pi_2;\pi_3;\cdots;\varphi_n?;\pi_n;\varphi_1?;\pi_1.$$

Pushing all the tests leftwards in this way, we obtain a sequence of tests that is followed by $\pi_1;\cdots;\pi_n;\pi_1$, namely:

$$\varphi_1?;\langle \pi_1 \rangle \varphi_2?;\cdots;\langle \pi_1;\cdots;\pi_{n-1} \rangle \varphi_n?;\langle \pi_1;\cdots;\pi_n \rangle \varphi_1?\,;\pi_1;\cdots;\pi_n;\pi_1.$$

By hypothesis, $\pi_1;\cdots;\pi_n;\pi_1 \equiv \pi_1$. Therefore our program σ is equivalent to

$$\varphi_1?;\langle \pi_1 \rangle \varphi_2?;\cdots;\langle \pi_1;\cdots;\pi_n \rangle \varphi_1?\,;\pi_1.$$

Proposition 4(4) allows us to group all the tests from the second test on into a single test:

$$\varphi_1?\,;(\langle \pi_1 \rangle \varphi_2 \wedge \cdots \wedge \langle \pi_1;\cdots;\pi_n \rangle \varphi_1)?\,;\pi_1.$$

Let us abbreviate $\langle \pi_1 \rangle \varphi_2 \wedge \cdots \wedge \langle \pi_1;\cdots;\pi_n \rangle \varphi_1$ by χ. Proposition 5 and Proposition 4(4) allow us to permute the two tests $\varphi_1?$ and $\chi_1?$:

$$\varphi_1?;\chi?;\pi_1$$

is program equivalent to

$$\chi?;\varphi_1?;\pi_1.$$

Finally, the latter is subsumed by the program $\varphi_1?\,;\pi_1$, in the sense that

$$(\chi?;\varphi_1?;\pi_1) \cup (\varphi_1?;\pi_1)$$

is program-equivalent to $\varphi_1?\,;\pi_1$ by Proposition 4(7) . ■

Proposition 10. *The following program equivalences are* DL-PA *valid:*

$$p?;+p \equiv p?$$
$$\neg p?;-p \equiv \neg p?$$
$$\pm p;+p \equiv +p$$
$$\pm p;-p \equiv -p$$
$$\pm p;+q \equiv +q;\pm p \qquad \textit{if } p \neq q$$
$$\pm p;-q \equiv -q;\pm p \qquad \textit{if } p \neq q$$

The next proposition contains a sufficient condition for the applicability of Proposition 9.

Proposition 11. *Suppose the assignment sequences π and π' have exactly the same variables, i.e., $\mathbb{P}_\pi = \mathbb{P}_{\pi'}$. Then $\pi;\pi'$ is program-equivalent to π'.*

Proof: This follows by iterated application of the program equivalences for assignments of Proposition 10. ■

The next proposition tells us how we can add 'dummy' assignments in order to ensure that two assignment sequences are about the same variables.

Proposition 12. *Every program π is program-equivalent to*

$$(p?;+p;\pi) \cup (\neg p?;-p;\pi).$$

Proof: By Proposition 4(5), π is equivalent to $\texttt{skip};\pi$. By Proposition 4(8), the latter is equivalent to

$$(p? \cup \neg p?);\pi.$$

By distribution of ";" over "$\cup$" (Proposition 4(3)) this is equivalent to

$$(p?;\pi) \cup (\neg p?;\pi).$$

Finally, the above is program-equivalent to

$$(p?;+p;\pi) \cup (\neg p?;-p;\pi)$$

due to the first two program equivalences of Proposition 10.

Note that each of the above steps is correct thanks to Proposition 7. ■

Proposition 13 ([10]). *The following formula equivalences are* DL-PA *valid.*

$$\begin{aligned}
\langle \pm p\rangle\top &\leftrightarrow \top \\
\langle \pm p\rangle\bot &\leftrightarrow \bot \\
\langle \pm p\rangle\neg\varphi &\leftrightarrow \neg\langle \pm p\rangle\varphi \\
\langle \pm p\rangle(\varphi_1 \vee \varphi_2) &\leftrightarrow \langle \pm p\rangle\varphi_1 \vee \langle \pm p\rangle\varphi_2 \\
\langle +p\rangle q &\leftrightarrow \begin{cases} \top & \textit{if } q = p \\ q & \textit{otherwise} \end{cases} \\
\langle -p\rangle q &\leftrightarrow \begin{cases} \bot & \textit{if } q = p \\ q & \textit{otherwise} \end{cases}
\end{aligned}$$

B. Elimination of the Kleene star

We now give a procedure eliminating the Kleene star from formulas. This contrasts with PDL where elimination is impossible. Our procedure has three steps.

1) Take some innermost star-operator, i.e., some π^* such that π is star-free. Transform π into the nondeterministic composition of conditioned sequences of assignments

$$\pi' = (\varphi_1?;\pi_1) \cup \cdots \cup (\varphi_n?;\pi_n)$$

where every π_k is a sequence of assignments.

2) Make all the assignment sequences π_k assign exactly the same variables: replace π_k by $(p?;+p;\pi_k)\cup(\neg p?;-p;\pi_k)$ whenever $p \notin \mathbb{P}_{\pi_k}$ and $p \in \mathbb{P}_{\pi_l}$ for some $l \leq n$, and put the result again in the form of a nondeterministic composition of conditioned assignment sequences. We obtain therefore a program of the form

$$\pi'' = (\varphi''_1?;\pi''_1) \cup \cdots \cup (\varphi''_m?;\pi''_m)$$

with $\mathbb{P}_{\pi''_1} = \cdots = \mathbb{P}_{\pi''_m}$.

3) Replace $(\pi'')^*$ by $(\pi'')^{\leq m}$.

The first step preserves program equivalence by Proposition 8. The second step does so because of Proposition 12 (to be applied at most n times per assignment sequence). The third step is guaranteed by Proposition 9, which applies because of Proposition 11: when the assignment sequences π and π' have exactly the same variables then $\pi;\pi' \equiv \pi'$. All three steps preserve formula equivalence: the formulas resulting from replacement of the program equivalences are formula-equivalent thanks to Proposition 7.

Summing up, in the first three steps we have succeeded in eliminating an innermost star-operator. By iterating these three steps we eliminate star operators entirely from formulas.

Theorem 1. *For every* DL-PA *formula φ there exists a formula φ' without the Kleene star such that φ and φ' are formula equivalent in* DL-PA.

C. Deciding satisfiability

Definition 4. *The problem of satisfiability checking SAT has the following input and output.*

- *Input: a formula φ;*
- *Output:*
  - *true, when $||\varphi|| \neq \emptyset$;*
  - *false, otherwise.*

The decision procedure is as follows:

1) Eliminate the Kleene star from φ.
2) Eliminate the sequential and the nondeterministic composition operators as well as the test operators.
3) Eliminate all the dynamic operators with assignments.

The first step uses Theorem 1. The second step uses the standard PDL equivalences of Proposition 3 that we have recalled in the beginning of Section IV-A. We therefore end up with formulas having only atomic programs, i.e., assignments. We can then distribute these dynamic operators over the boolean operators (using in particular the equivalence for negation of Proposition 13) and finally eliminate them by the DL-PA equivalences for assignments of Proposition 13. The resulting formula has no more dynamic operators, it is equivalent to the original formula, and its validity or satisfiability may be checked by means of any theorem prover for classical propositional logic.

Theorem 2. *For every* DL-PA *formula φ there is a propositional formula φ' such that $\varphi \leftrightarrow \varphi'$ is* DL-PA *valid.*

Corollary 1. *The problem of satisfiability of a* DL-PA *formula is decidable.*

Observe that the only principles beyond those of star-free PDL that we have used in our decision procedure are propositions 5 and 6 (that allow us to prove Proposition 7 and that can be viewed as inference rules), the formula equivalences of Proposition 13 (though the one for disjunctions is already provable in star-free PDL), and the program equivalences of Proposition 4. The latter axiom schemes are a bit unusual: they do not have the form of formulas but that of program equivalences. Together, these principles provide a complete but somewhat non-standard axiomatisation of DL-PA.

D. Compactness and interpolation

In PDL, there are two ways to define the consequence relation $\models$ between a set of formulas Γ and a formula φ. In contrast, in DL-PA there is only one way to do so: φ is a consequence of Γ iff $\bigcap_{\psi \in \Gamma} ||\psi|| \subseteq ||\varphi||$. In other words, for every valuation v, if $v \in ||\psi||$ for every $\psi \in \Gamma$ then $v \in ||\varphi||$. Compactness of the DL-PA consequence relation immediately follows from Theorem 2.

Theorem 2 also helps us to prove the interpolation property for DL-PA. We actually need a slightly stronger version guaranteeing that the propositional formula that is equivalent to φ has the same propositional variables as φ.

Theorem 3. *For every* DL-PA *formula φ there is a propositional formula φ' such that $\mathbb{P}_{\varphi'} = \mathbb{P}_{\varphi}$, and $\varphi \leftrightarrow \varphi'$ is* DL-PA *valid.*

Proof: All the rewriting rules of the procedure underlying Theorem 2 preserve the variables, except those for subformulas of the form $\langle \pm p \rangle \top$, $\langle \pm p \rangle \bot$, $\langle +p \rangle q$, and $\langle -p \rangle q$. The equivalences for the latter eliminate propositional variables. Fortunately they have variants that preserve variables and that are also valid:

$$\langle \pm p \rangle \top \leftrightarrow p \vee \neg p$$

$$\langle \pm p \rangle \bot \leftrightarrow p \wedge \neg p$$

$$\langle +p \rangle q \leftrightarrow \begin{cases} p \vee \neg p & \text{if } q = p \\ q \wedge (p \vee \neg p) & \text{otherwise} \end{cases}$$

$$\langle -p \rangle q \leftrightarrow \begin{cases} p \wedge \neg p & \text{if } q = p \\ q \wedge (p \vee \neg p) & \text{otherwise} \end{cases}$$

■

Now an interpolant of $\psi \models \varphi$ can be obtained by rewriting φ and ψ to propositional φ' and ψ' in a way such that their respective languages are preserved. By the interpolation property for propositional logic, there exists an interpolant χ of $\psi' \models \varphi'$. As the new equivalences are language-preserving, χ is also an interpolant of $\psi \models \varphi$.

V. Complexity of decision problems: lower bounds

We now give complexity results for our logic DL-PA. We have already defined the problem SAT in Section IV-C; it remains to define model checking.

Definition 5. *The problem of model checking MC has the following input and output.*

- *Input: a model v and a formula φ such that $v \subseteq \mathbb{P}_{\varphi}$;*
- *Output:*
  - *true, when $v \in ||\varphi||$;*
  - *false, otherwise.*

Note that the constraint $v \subseteq \mathbb{P}_{\varphi}$ implies that v is finite. We are now going to provide lower bounds for both problems.

Theorem 4. *The problems of* DL-PA *model checking and* DL-PA *satisfiability are both EXPTIME-hard.*

Proof: The result is established for model checking in Section V-A, and for satisfiability checking in Section V-B. ■

A. Lower bound for model checking: encoding PEEK

We prove that model checking with DL-PA is EXPTIME-hard by means of a logarithmic-space reduction of the problem PEEK-G_5 [12] into the problem MC.

PEEK-G_5 is in terms of two players E ('Eloise', the existential player) and A ('Abelard', the universal player).

Definition 6. An instance of Peek *is a tuple $PE = (X_E, X_A, \Phi, v_0, \tau)$ where*

- *X_E and X_A are finite sets of propositional variables such that $X_E \cap X_A = \emptyset$, where the idea is that Player E controls the variables in X_E and Player A controls the variables in X_A;*
- *Φ is a propositional formula over $X_E \cup X_A$;*
- *$v_0 \subseteq X_E \cup X_A$ indicates which variables are currently true;*
- *τ is either A or E, indicating which player makes the next move.*

Informally, each instance $PE = (X_E, X_A, \Phi, v_0, \tau)$ of Peek is played as follows. Agents' turns strictly alternate. At their respective turn, Player E (resp. A) *moves* by changing the truth value of at most one variable of X_E (resp. X_A) in the current valuation, either adding or withdrawing it from the valuation. The game ends when Φ first becomes true. We are considering PEEK-G_5, where Player A cannot move from a situation where Φ is true, and Player E wins the game if Φ ever becomes true. We say that *Player E has a winning strategy in PE* if she can make a sequence of moves at her turns and ensure to win whatever the moves made by Player A at his turn. Let us make this more precise.

The *game tree* associated with PE is a tree labeled by valuations such that the root is labeled v_0, starting with the agent τ at the root, all nodes are obtained by E and A alternatively performing all possible moves. A node is an E node if it is Eloise's turn to move, and it is an A node if it is Abelard's turn.

A (memoryless) *strategy* of Eloise is a function from valuations to actions of E. A branch $(n_0, n_1, n_2, \ldots)$ of the game tree is compatible with Eloise's strategy s_E if and only if for every E-node n_k, the valuation v_{k+1} labeling n_{k+1} is obtained by applying E's action $s_E(n_k)$ to the valuation v_k.

Eloise has a winning strategy if and only if there is a strategy s_E such that every branch $(n_0, n_1, n_2, \ldots)$ of the game tree that is compatible with s_E has a node n_k such that $v_k \in ||\Phi||$.

Definition 7. *The decision problem PEEK-G_5 is the following:*

- *Input: an instance $PE = (X_E, X_A, \Phi, v_0, \tau)$ of Peek;*
- *Output:*

– *true, when Player E has a winning strategy in PE;*
– *false, otherwise.*

Theorem 5 ([12]). *PEEK-G_5 is EXPTIME-complete.*

We prove that deciding the model checking problem MC for DL-PA is EXPTIME-hard by reducing PEEK-G_5 to it. Beyond the variables of $X_E \cup X_A$, we use the propositional variables elo and nowin. The intended meaning of elo is that it is Player E's turn to play, whereas the intended meaning of nowin is that Player E has no winning strategy. Let $PE = (X_E, X_A, \Phi, v_0, \tau)$ be an instance of the Peek problem. We associate with it an instance (v, φ) of the MC problem as follows. First, the valuation v is defined by: if $\tau = A$ then $v = v_0 \cup \{\mathsf{nowin}\}$ else $v = v_0 \cup \{\mathsf{nowin}, \mathsf{elo}\}$. Second, in order to define the formula φ, we need to introduce the following abbreviations:

$$\mathsf{moveE} \overset{\mathsf{def}}{=} \mathsf{elo}?; \bigcup_{x \in X_E} (-x \cup +x); -\mathsf{elo}$$

$$\mathsf{moveA} \overset{\mathsf{def}}{=} \neg\mathsf{elo}?; \cup_{y \in X_A} (-y \cup +y); +\mathsf{elo}$$

$$\mathsf{move} \overset{\mathsf{def}}{=} (\mathsf{moveE} \cup \mathsf{moveA})\,; ((\Phi?; -\mathsf{nowin}) \cup \neg\Phi?)$$

Now, we define

$$\varphi \overset{\mathsf{def}}{=} [\mathsf{move}^*](\mathsf{nowin} \to (\neg\Phi \wedge (\mathsf{elo} \to [\mathsf{move}]\mathsf{nowin}) \wedge (\neg\mathsf{elo} \to \langle\mathsf{move}\rangle\mathsf{nowin})))$$

Obviously, given PE, the computation of (v, φ) can be done in logarithmic space. Moreover,

Lemma 1. *PEEK-$G_5(PE)$ returns false iff $MC(v, \varphi)$ returns true.*

Informally, the idea is that PEEK-$G_5(PE)$ returns false iff when we assume that Eloise does not have a winning strategy (by putting nowin in v along with the the initial situation of PE described by v_0), then we can verify that the following is an invariant in the game-tree starting at v:

> "if Eloise has no winning strategy, then (1) at Eloise's turn, she has no winning strategy after any move, (2) at Abelard's turn, there is a successor node where Eloise has no winning strategy."

Proof: Suppose PEEK-$G_5(PE)$ returns false: Eloise does not have a winning strategy in PE. Therefore the nodes of the game tree associated with PE can be further labeled by nowin in the following way:

- The root is labeled nowin;
- For E-nodes labeled nowin, each of E's moves leads to a node that it labeled nowin;
- For A-nodes labeled nowin, some move of A's moves leads to a node that it labeled nowin;
- A node whose valuation satisfies Φ cannot be labeled nowin.

The root is therefore now labeled by $v_0 \cup \{\mathsf{nowin}\}$ if $\tau = A$ and by $v_0 \cup \{\mathsf{nowin}, \mathsf{elo}\}$ if $\tau = E$. Then

$$[\mathsf{move}^*](\mathsf{nowin} \to (\neg\Phi \wedge (\mathsf{elo} \to [\mathsf{move}]\mathsf{nowin}) \wedge (\neg\mathsf{elo} \to \langle\mathsf{move}\rangle\mathsf{nowin}))))$$

is true at v because move correctly encodes the moves. Therefore

$$MC(v, [\mathsf{move}^*](\mathsf{nowin} \to (\neg\Phi \wedge (\mathsf{elo} \to [\mathsf{move}]\mathsf{nowin}) \wedge (\neg\mathsf{elo} \to \langle\mathsf{move}\rangle\mathsf{nowin}))))\,)$$

returns true.

The other way round, suppose

$$MC(v, [\mathsf{move}^*](\mathsf{nowin} \to (\neg\Phi \wedge (\mathsf{elo} \to [\mathsf{move}]\mathsf{nowin}) \wedge (\neg\mathsf{elo} \to \langle\mathsf{move}\rangle\mathsf{nowin}))))\,)$$

returns true. Consider the PDL model $M^{\mathsf{DL\text{-}PA}}$ that we have built in Section III and focus on the relation $R(\mathsf{move})$. The alternating application of the actions of E and A that it gives us directly provides a winning strategy for Abelard at v. Therefore Eloise cannot have a winning strategy. ■

Proposition 14. *The* DL-PA *model checking problem is EXPTIME-hard.*

Proof: This follows directly from Lemma 1. ■

B. Lower bound for satisfiability checking: encoding MC

We finally establish that the problem of satisfiability checking is EXPTIME-hard. Since we have proved in Section V-A that the model checking problem MC is EXPTIME-hard (Proposition 14), it suffices to reduce MC to the problem of satisfiability checking.

Proposition 15. *The problem of* DL-PA *satisfiability checking is EXPTIME-hard.*

Proof: This follows directly from propositions 2 and 14. ■

We already know that star-free DL-PA is PSPACE-complete [11]. We can then state the following:

Corollary 2. *A logarithmic-space reduction from the* DL-PA *satisfiability problem to the satisfiability problem of its star-free fragment exists iff PSPACE = EXPTIME.*

So van Eijck's problem of finding an efficient method for translating DL-PA programs into star-free programs has a positive answer if and only if PSPACE = EXPTIME.

VI. Complexity of decision problems: upper bounds

We are now going to provide upper bounds for the problems MC and SAT. This establishes that the lower-bounds of Section V are tight. While our rewriting procedure of Section IV-C could be turned into decision procedures for both problems, it would however lead to upper bounds far beyond those we are going to establish now.

Theorem 6. *The problems of* DL-PA *model checking and* DL-PA *satisfiability are both in EXPTIME.*

The proof uses a satisfiability preserving polynomial transformation $\mathbf{tr}_{\mathsf{PDL}}(.)$ from the formulas and programs of DL-PA to the formulas and programs of PDL. Our translation replaces each assignment $\pm p$ by an abstract program $a_{\pm p}$. Precisely it is defined by recursion as follows:

$$\begin{aligned}\mathbf{tr}_{\mathsf{PDL}}(p) &\overset{\text{def}}{=} p \qquad \text{if } p \in \mathbb{P}\\ \mathbf{tr}_{\mathsf{PDL}}(+p) &\overset{\text{def}}{=} a_{+p}\\ \mathbf{tr}_{\mathsf{PDL}}(-p) &\overset{\text{def}}{=} a_{-p}\end{aligned}$$

and homomorphic for the other formula and program connectives. Observe that the length of every translated formula $\mathbf{tr}_{\mathsf{PDL}}(\varphi)$ is linear in the length of φ.

The abstract programs $a_{\pm p}$ that occur in a formula φ should behave in the same way as the original assignment $\pm p$. We achieve this by means of the following set of formulas.

$$\begin{aligned}\Gamma_\varphi = &\{[a_{+p}]p \ : \ p \in \mathbb{P}_\varphi\} \cup\\ &\{[a_{-p}]\neg p \ : \ p \in \mathbb{P}_\varphi\} \cup\\ &\{\langle a_{\pm p}\rangle\top \ : \ p \in \mathbb{P}_\varphi\} \cup\\ &\{q \to [a_{\pm p}]q \ : \ p, q \in \mathbb{P}_\varphi, p \neq q\} \cup\\ &\{\neg q \to [a_{\pm p}]\neg q \ : \ p, q \in \mathbb{P}_\varphi, p \neq q\}\end{aligned}$$

Observe that for every φ, the set Γ_φ is finite; its cardinality is quadratic in the length of φ. Each element of Γ_φ having constant length, the length of $\bigwedge \Gamma_\varphi$ is quadratic in the length of φ.

Let U_φ be the program $\left(\bigcup_{p \in \mathbb{P}_\varphi}(a_{+p} \cup a_{-p})\right)^*$. The modal operator $[U_\varphi]$ plays the role of a master modality in our proof: given a valuation v, the program U_φ relates v to every valuation that is examined during the evaluation of φ at v.

Proposition 16. *For every* DL-PA *formula φ, φ is* DL-PA *satisfiable if and only if*

$$\mathbf{tr}_{\mathsf{PDL}}(\varphi) \wedge [U_\varphi]\left(\bigwedge \Gamma_\varphi\right)$$

is PDL *satisfiable.*

Proof: From the left to the right, let v_0 be a DL-PA valuation such that $v_0 \in ||\varphi||$. Define $M^\varphi = (W, R^\varphi, V)$ where $W = 2^{\mathbb{P}}$ is the set of all valuations, V is the identity function (i.e., $V(v) = v$ for every $v \in W$), and where the accessibility relation is as follows for the relevant atomic programs $a_{\pm p}$ such that $p \in \mathbb{P}_\varphi$:

$$\begin{aligned}R^\varphi(a_{+p}) &= \{(v, v') \ : \ v' = v \cup \{p\}\}\\ R^\varphi(a_{-p}) &= \{(v, v') \ : \ v' = v \setminus \{p\}\}\end{aligned}$$

Furthermore, we set $R^\varphi(\pi_0) = \emptyset$ for all atomic programs π_0 for which there is no $p \in \mathbb{P}_\varphi$ with $\pi_0 = a_{+p}$ or $\pi_0 = a_{-p}$. Observe that M^φ is indeed a model of PDL as defined in Section III. We are going to prove that

$$M^\varphi, v_0 \Vdash \mathbf{tr}_{\mathsf{PDL}}(\varphi) \wedge [U_\varphi]\left(\bigwedge \Gamma_\varphi\right)$$

First, by construction of M^φ we have $M^\varphi, v \Vdash \bigwedge \Gamma_\varphi$ for every $v \in W$. (In particular, each of the accessibility relations $R^\varphi(a_{\pm p})$ is serial.) Therefore $M^\varphi, v_0 \Vdash [U_\varphi]\left(\bigwedge \Gamma_\varphi\right)$.

It remains to establish that $M^\varphi, v_0 \Vdash \mathbf{tr}_{\mathsf{PDL}}(\varphi)$. We prove by simultaneous induction on the form of subformula ψ and program π occurring in φ that

1) $v \in ||\psi||$ iff $M^\varphi, v \Vdash \mathbf{tr}_{\mathsf{PDL}}(\psi)$ and
2) $(v, v') \in ||\pi||$ iff $(v, v') \in R^\varphi(\mathbf{tr}_{\mathsf{PDL}}(\pi))$.

for all valuations v and v'. When the subformula ψ of φ is atomic then

$$\begin{aligned}v \in ||p|| \quad &\text{iff} \quad p \in v\\ &\text{iff} \quad M^\varphi, v \Vdash p.\end{aligned}$$

When the program π occurring in φ is of the form $+p$ then

$$\begin{aligned}(v, v') \in ||+p|| \quad &\text{iff} \quad v' = v \cup \{p\}\\ &\text{iff} \quad (v, v') \in R^\varphi(a_{+p});\end{aligned}$$

and similarly for $-p$. For the induction step, the only interesting case is that of the modal operator. We have:

$$\begin{aligned}v \in ||\langle\pi\rangle\psi|| \quad &\text{iff} \quad \text{there is } v' \text{ such that } (v, v') \in ||\pi||\\ &\qquad \text{and } v' \in ||\psi||\\ &\text{iff} \quad \text{there is } v' \text{ such that } (v, v') \in R^\varphi(\mathbf{tr}_{\mathsf{PDL}}(\pi))\\ &\qquad \text{and } M^\varphi, v' \Vdash \mathbf{tr}_{\mathsf{PDL}}(\psi)\\ &\qquad\qquad\qquad\qquad \text{(by I.H., twice)}\\ &\text{iff} \quad M^\varphi, v \Vdash \langle\mathbf{tr}_{\mathsf{PDL}}(\pi)\rangle\mathbf{tr}_{\mathsf{PDL}}(\psi)\\ &\text{iff} \quad M^\varphi, v \Vdash \mathbf{tr}_{\mathsf{PDL}}(\langle\pi\rangle\psi)\end{aligned}$$

From the right to the left, let $M = (W, R, V)$ be a PDL model and let w_0 be a state in M such that

$$M, w_0 \Vdash \mathbf{tr}_{\mathsf{PDL}}(\varphi) \wedge [U_\varphi]\left(\bigwedge \Gamma_\varphi\right).$$

First, observe that $M, w \Vdash [U_\varphi]\left(\bigwedge \Gamma_\varphi\right)$ for every w such that $(w_0, w) \in R(U_\varphi)$. We therefore have:

- if $(w, w') \in R(a_{+p})$ then $V(w') \cap \mathbb{P}_\varphi = (V(w) \cup \{p\}) \cap \mathbb{P}_\varphi$
- if $(w, w') \in R(a_{-p})$ then $V(w') \cap \mathbb{P}_\varphi = (V(w) \setminus \{p\}) \cap \mathbb{P}_\varphi$.

for every $p \in \mathbb{P}_\varphi$ and every w such that $(w_0, w) \in R(U_\varphi)$. This allows us to prove by induction on the form of subformula ψ and program π occurring in φ that

1) $M, w \Vdash \mathbf{tr}_{\mathsf{PDL}}(\psi)$ iff $V(w) \in ||\psi||$ and
2) $(w, w') \in R(\mathbf{tr}_{\mathsf{PDL}}(\pi))$ iff $(V(w), V(w')) \in ||\pi||$

for every w such that $(w_0, w) \in R(U_\varphi)$ and for every w'. For the three base cases: when the subformula ψ of φ is atomic then

$$\begin{aligned}M, w \Vdash \mathbf{tr}_{\mathsf{PDL}}(p) \quad &\text{iff} \quad p \in V(w)\\ &\text{iff} \quad V(w) \in ||p||;\end{aligned}$$

when the program π occurring in φ is of the form $+p$ then

$$\begin{aligned}(w, w') \in R(a_{+p}) \quad &\text{iff} \quad V(w') = V(w) \cup \{p\}\\ &\text{iff} \quad (V(w), V(w')) \in ||+p||;\end{aligned}$$

and similarly for $-p$. For the induction step the only interesting case is that of the modal operator. We have:

$$\begin{aligned}M, w \Vdash \mathbf{tr}_{\mathsf{PDL}}(\langle\pi\rangle\psi) \quad &\text{iff} \quad M, w \Vdash \langle\mathbf{tr}_{\mathsf{PDL}}(\pi)\rangle\mathbf{tr}_{\mathsf{PDL}}(\psi)\\ &\text{iff} \quad \text{there is } w' \text{ s.t. } (w, w') \in R(\mathbf{tr}_{\mathsf{PDL}}(\pi))\\ &\qquad \text{and } M, w' \Vdash \mathbf{tr}_{\mathsf{PDL}}(\psi)\\ &\text{iff} \quad \text{there is } w' \text{ s.t. } (V(w), V(w')) \in ||\pi||\\ &\qquad \text{and } V(w') \in ||\psi|| \qquad \text{(by I.H.)}\\ &\text{iff} \quad V(w') \in ||\langle\pi\rangle\psi||\end{aligned}$$

This ends the proof of Proposition 16. ■

To conclude, Theorem 6 follows from Proposition 16, the fact that $\mathbf{tr}_{\mathsf{PDL}}(.)$ is a satisfiability preserving polynomial transformation, and Proposition 2.

VII. Extensions

We now discuss some extensions of our basic logic DL-PA.

A. More general assignments

Our assignments are more restricted than those of [6]–[8]. There, assignments take the form $p{\leftarrow}\varphi$: to p the truth value of φ is assigned, where φ is any formula (and not just $\top$ or $\bot$ as in our DL-PA).

Programs $p{\leftarrow}\varphi$ can be simulated in our language by $(\varphi?; +p) \cup (\neg\varphi?; -p)$. This may result in an exponential increase of formula size. However, an inspection of our proofs shows that all our complexity results also hold for the richer language with assignments $p{\leftarrow}\varphi$. It remains to find out whether the language with general assignments is more succinct.

B. Other program connectives

Our logic can be extended as well by other program connectives that are familiar from PDL. We briefly discuss the properties of DL-PA extended with the *converse operator* "$^-$". Its interpretation is $||\pi^-|| = (||\pi||)^{-1}$, where $(||\pi||)^{-1}$ is the inverse of the relation $||\pi||$.

Consider an innermost occurrence π^-. Thanks to the program equivalences of Section IV, we may suppose that π is star-free. The converse operator distributes over sequential and nondeterministic composition in the usual way:

$$\begin{aligned}(\pi_1;\pi_2)^- &\equiv \pi_2^-;\pi_1^- \\ (\pi_1 \cup \pi_2)^- &\equiv \pi_1^- \cup \pi_2^-\end{aligned}$$

and when it meets a test or an atomic program it can be eliminated by means of the following equivalences:

$$\begin{aligned}(\psi?)^- &\equiv \psi? \\ (+p)^- &\equiv p? \cup (p?;-p) \\ (-p)^- &\equiv \neg p? \cup (\neg p?;+p)\end{aligned}$$

Converse DL-PA can therefore still be reduced to propositional logic.

As to complexity, the extension of DL-PA by the converse operator can be mapped into PDL with converse by extending the polynomial transformation of Section III in a straightforward way. Converse PDL being still in EXPTIME [14], it follows that the satisfiability problem of converse DL-PA is EXPTIME-complete, too.

VIII. Relationship with the logics of propositional control

It was shown in [11] that star-free DL-PA embeds the logics based on the idea of *propositional control* that were recently studied in the multi-agent literature ([15], [16]). We give their embedding in Section VIII-A (adapting the presentation to our notation) and extend it in Section VIII-B in order to show how full DL-PA relates to Coalition Logic of Propositional Control and Delegation [13].

A. Coalition logic of propositional control

Models of Coalition Logic of Propositional Control (CL-PC) have a twofold interpretation of propositional variables: besides the valuation function v mapping propositional variables to truth values, there is a control function ξ mapping each propositional variable to some agent in a finite set of agents $\mathbb{A}$. Saying that the agent $\xi(p)$ controls p, we mean that i can set p to true and can set p to false. As ξ is a function, control is exclusive: for every p there is at most one i controlling p. As the function ξ is total, control is also assumed exhaustive: for every p there is at least one i controlling p.

With CL-PC one can model the capabilities of agents and groups of agents to achieve a state of affairs: in the model (v, ξ), agent i is capable to achieve φ if there exists a valuation v' that differs from v only in the interpretation of the variables that are under i's control, and is such that φ is true in (v', ξ). Formally:

$$\begin{aligned}||\Diamond_i\varphi|| = \{\,(v,\xi)\ &: \text{there is } v' \text{ such that } (v',\xi) \in ||\varphi|| \text{ and} \\ &\quad \text{for every } p, \text{if } \xi(p) \neq i \text{ then } v(p) = v'(p)\,\}\end{aligned}$$

The interpretation of the coalition operator $\Diamond_J$ straightforwardly generalises this truth condition.

In star-free DL-PA we can encode agent i's control over p by means of a special propositional variable $\mathsf{c}_{i,p}$. Then exclusivity and exhaustivity of control over a set of variables $P \subseteq \mathbb{P}$ are expressed by:

$$\begin{aligned}\mathsf{Exc}(P) &\overset{\text{def}}{=} \bigwedge_{p \in P}\ \bigwedge_{i,j \in \mathbb{A}, i \neq j} \neg(\mathsf{c}_{i,p} \wedge \mathsf{c}_{j,p}) \\ \mathsf{Exh}(P) &\overset{\text{def}}{=} \bigwedge_{p \in P}\ \bigvee_{i \in \mathbb{A}} \mathsf{c}_{i,p}\end{aligned}$$

Finally, we set a translation $\mathbf{tr}$ such that:

$$\begin{aligned}\mathbf{tr}(p) &\overset{\text{def}}{=} p \quad \text{if } p \in \mathbb{P} \\ \mathbf{tr}(\Diamond_i\varphi) &\overset{\text{def}}{=} \langle \mathtt{skip} \cup (\mathsf{c}_{i,p_1}?;+p_1) \cup (\mathsf{c}_{i,p_1}?;-p_1)\rangle \cdots \\ &\qquad \langle \mathtt{skip} \cup (\mathsf{c}_{i,p_n}?;+p_n) \cup (\mathsf{c}_{i,p_n}?;-p_n)\rangle\ \mathbf{tr}(\varphi) \\ &\qquad\qquad \text{if } \mathbb{P}_\varphi = \{p_1, \ldots, p_n\}\end{aligned}$$

and homomorphic for the boolean operators.

The following result corresponds to Theorem 4 in [11].

Proposition 17. *A* CL-PC *formula* φ *is* CL-PC *satisfiable iff* $\mathsf{Exc}(\mathbb{P}_\varphi) \wedge \mathsf{Exh}(\mathbb{P}_\varphi) \wedge \mathbf{tr}(\varphi)$ *is (star-free)* DL-PA *satisfiable.*

It is also clear that the lengths of $\mathsf{Exc}(\mathbb{P}_\varphi)$, $\mathsf{Exh}(\mathbb{P}_\varphi)$, and $\mathbf{tr}(\varphi)$ are all polynomial in the length of φ. Our translation therefore provides a polynomial embedding of CL-PC into DL-PA.

B. Coalition logic of propositional control and delegation

Coalition logic of propositional control and delegation (DCL-PC) extends CL-PC by dynamic operators of transfer of control over a propositional variable. They were called delegation programs in [13], [17], but the term *control transfer program* appears to be more appropriate. Atomic control transfer programs are of the form $i \rightsquigarrow_p j$ and are read "i

transfers his control over p to j". The intuition is that $i \rightsquigarrow_p j$ is applicable when i controls p and that it changes the control function ξ such that j gets control over p (and i looses it, control being exclusive). Complex delegation programs are defined by means of the standard PDL operators.

The interpretation of a delegation program is a binary relation on the set of models of propositional control. For atomic programs we have:

$$\begin{aligned} ||i \rightsquigarrow_p j|| \ = \ \{ ((v,\xi),(v,\xi')) \ : & \ \xi(p) = i, \ \xi'(p) = j, \text{ and} \\ & \xi(q) = \xi'(q) \text{ for } q \neq p \} \end{aligned}$$

The interpretation of complex programs and of formulas is as usual. The interpretation of $\langle\pi\rangle\varphi$ is:

$$\begin{aligned} ||\langle\pi\rangle\varphi|| \ = \ \{ (v,\xi) \ : & \text{ there is } (v',\xi') \text{ such that} \\ & ((v,\xi),(v',\xi')) \in ||\pi|| \text{ and } (v',\xi') \in ||\varphi|| \} \end{aligned}$$

We extend **tr** to provide a translation from the richer language of DCL-PC into the one of DL-PA. The key clauses are:

$$\begin{aligned} \mathbf{tr}(\langle\pi\rangle\varphi) & \overset{\mathtt{def}}{=} \langle\mathbf{tr}(\pi)\rangle\mathbf{tr}(\varphi) \\ \mathbf{tr}(\varphi?) & \overset{\mathtt{def}}{=} \mathbf{tr}(\varphi)? \\ \mathbf{tr}(i \rightsquigarrow_q j) & \overset{\mathtt{def}}{=} \mathsf{c}_{i,p}? \, ; \, -\mathsf{c}_{i,p} \, ; \, +\mathsf{c}_{j,p} \end{aligned}$$

Theorem 7. *A* DCL-PC *formula* φ *is* DCL-PC *satisfiable iff* $\mathsf{Exc}(\mathbb{P}_\varphi) \wedge \mathsf{Exh}(\mathbb{P}_\varphi) \wedge \mathbf{tr}(\varphi)$ *is* DL-PA *satisfiable.*

The other way round, DL-PA can also be translated into DCL-PC if the set of agents contains at least two agents. Let us call these agents t and f. We then encode truth of p as t controlling p and falsity of p as f controlling p. Then the atomic assignment $+p$ corresponds to the control transfer program $t \rightsquigarrow_p t \cup f \rightsquigarrow_p t$ and $-p$ corresponds to $f \rightsquigarrow_p f \cup t \rightsquigarrow_p f$. The other program connectives and boolean operators of DL-PA are mapped homomorphically. It is then routine to prove that a DL-PA formula is DL-PA satisfiable if and only if its translation into the language of DCL-PC is DCL-PC satisfiable.

It follows that both the model checking problem and the satisfiability problem of DCL-PC are EXPTIME-hard. Provided that PSPACE ≠ EXPTIME, this contradicts the claim in [13] that both are PSPACE-complete.[2]

IX. Conclusion

Our logic DL-PA is an interesting alternative to PDL. On the one hand, we can basically reason about the same kind of phenomena as in PDL: programs changing the truths of the world. This is demonstrated by the example of the counter of Section II-C and also by our encoding of the PEEK problem in Section V-A; this is also demonstrated by the embeddings of Coalition Logic of Propositional Control that we have mentioned in Section VIII. On the other hand, the mathematical properties of DL-PA are better than those of PDL: DL-PA is compact, has the interpolation property, and the Kleene star can be eliminated.

[2]The model checking algorithm in [13] is claimed to work in nondeterministic space (NPSPACE, which is the same complexity class as PSPACE). The algorithm consists in two mutually recursive procedures calling each other that decompose formulas according to the truth conditions: a procedure deciding whether $(v,\xi) \in ||\varphi||$ [13, Fig. 8] and a procedure deciding whether $(\xi,\xi') \in ||\pi||$ [13, Fig. 7]. However, that algorithm in fact requires alternating space (APSPACE).

Acknowledgment

We are grateful to a reviewer whose detailed comments helped improve the presentation. First and second authors acknowledge the support of the ANR 11 BS02 011 03 grant (project DynRes), while the third author acknowledges the support of the program Marie Curie People Action Trentino (project LASTS).

References

[1] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. MIT Press, 2000.

[2] A. R. Meyer and K. Winklmann, "On the expressive power of dynamic logic (preliminary report)," in *STOC*, M. J. Fischer, R. A. DeMillo, N. A. Lynch, W. A. Burkhard, and A. V. Aho, Eds. ACM, 1979, pp. 167–175.

[3] M. L. Tiomkin and J. A. Makowsky, "Propositional dynamic logic with local assignments," *Theor. Comput. Sci.*, vol. 36, pp. 71–87, 1985.

[4] A. Wilm, "Determinism and non-determinism in PDL," *Theor. Comput. Sci.*, vol. 87, no. 1, pp. 189–202, 1991.

[5] H. v. Ditmarsch, "The Russian cards problem," *Studia Logica*, vol. 75, no. 1, pp. 31–62, 2004.

[6] H. P. v. Ditmarsch, W. v. d. Hoek, and B. Kooi, "Dynamic epistemic logic with assignment," in *Proc. AAMAS'05*, 2005, pp. 141–148.

[7] B. Kooi, "Expressivity and completeness for public update logic via reduction axioms," *Journal of Applied Non-Classical Logics*, vol. 17, no. 2, pp. 231–253, 2007.

[8] H. P. v. Ditmarsch, A. Herzig, and T. d. Lima, "From Situation Calculus to Dynamic Logic," *Journal of Logic and Computation*, vol. 21, no. 2, pp. 179–204, 2011.

[9] J. S. Miller and L. S. Moss, "The undecidability of iterated modal relativization," *Studia Logica*, vol. 79, no. 3, pp. 373–407, 2005.

[10] J. v. Eijck, "Making things happen," *Studia Logica*, vol. 66, no. 1, pp. 41–58, 2000.

[11] A. Herzig, E. Lorini, F. Moisan, and N. Troquard, "A dynamic logic of normative systems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, T. Walsh, Ed. Barcelona: IJCAI/AAAI, 2011, pp. 228–233, erratum at http://www.irit.fr/~Andreas.Herzig/P/Ijcai11.html.

[12] L. J. Stockmeyer and A. K. Chandra, "Provably difficult combinatorial games," *SIAM J. Comput.*, vol. 8, no. 2, pp. 151–174, 1979.

[13] W. v. d. Hoek, D. Walther, and M. Wooldridge, "On the logic of cooperation and the transfer of control," *J. of AI Research (JAIR)*, vol. 37, pp. 437–477, 2010.

[14] G. De Giacomo and F. Massacci, "Combining deduction and model checking into tableaux and algorithms for converse-PDL," *Information and Computation*, vol. 162, no. 1–2, pp. 117–137, 2000.

[15] W. v. d. Hoek and M. Wooldridge, "On the logic of cooperation and propositional control," *Artif. Intell.*, vol. 164, no. 1-2, pp. 81–119, 2005.

[16] J. Gerbrandy, "Logics of propositional control," in *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, Eds. ACM, 2006, pp. 193–200.

[17] W. v. d. Hoek and M. Wooldridge, "On the dynamics of delegation, cooperation and control: a logical account," in *Proc. AAMAS'05*, 2005.