Tableaux for Dynamic Logic of Propositional Assignments

Tiago de Lima[1] and Andreas Herzig[2]

[1] CRIL, Univ. of Artois and CNRS, Rue Jean Souvraz, SP 18, 62307 Lens Cedex, France
[2] IRIT, Univ. of Toulouse and CNRS, 118, route de Narbonne, 31062 Toulouse Cedex 9, France

Abstract. The Dynamic Logic for Propositional Assignments (DL-PA) has recently been studied as an alternative to Propositional Dynamic Logic (PDL). In DL-PA, the abstract atomic programs of PDL are replaced by assignments of propositional variables to truth values. This makes DL-PA enjoy some interesting meta-logical properties that PDL does not, such as eliminability of the Kleene star, compactness and interpolation. We define and analytic tableaux calculus for DL-PA and show that it matches the known complexity results.

Keywords: Modal Logic; Propositional Dynamic Logic; Dynamic Logic of Propositional Assignments; analytic tableaux.

1 Introduction

Dynamic Logic of Propositional Assignments, abbreviated DL-PA, has recently been studied in [10,2] as an interesting alternative to Propositional Dynamic Logic (PDL) [7]. In a series of papers, it was shown that DL-PA is a useful tool to analyse various kinds of dynamic systems, such as normative systems [10], fusion operators [11], update and revision operators [9], or the evolution of argumentation frameworks [4]. While, in PDL, one can write formulas of the form $[a]\varphi$, meaning "after every possible execution of the abstract atomic program a, formula φ is true", in DL-PA, one can write formulas of the form $[+p]\varphi$, meaning "after assigning the truth value of p to true, formula φ is true". The atomic program $+p$ is an assignment that maps the propositional variable p to true. In fact, the atomic programs of DL-PA are sets of such assignments, that are viewed as partial functions from the set of propositional variables to $\{\top, \bot\}$. From these atomic programs, complex programs are built just as in PDL. For example, one can write in DL-PA the formula $[\neg p?; +p]p$, which means "if p is false then p is true after its truth value be assigned to true".

While the models of PDL are transition systems, the models of DL-PA are much smaller: valuations of classical propositional logic, i.e., nothing but sets of propositional variables. Due to that, DL-PA enjoys some interesting meta-logical properties that PDL lacks, such as eliminability of the Kleene star, compactness and interpolation.

The complexity of the satisfiability problem in DL-PA is the same as in PDL: it is in EXPTIME for the full language and PSPACE complete for the star-free fragment. EXPTIME membership of the full language is proved by a polynomial embedding of DL-PA into PDL, and PSPACE membership of the star-free fragment of DL-PA is proved via NPSPACE membership of its model checking problem, exploiting the fact that NPSPACE = PSPACE. However, these reductions lead to suboptimal theorem proving methods. Our aim in this paper is to define tableaux theorem proving procedures for DL-PA that are both direct and more efficient.

The paper is organized as follows. We start by recalling DL-PA (Section 2). Then, we provide a tableaux method for its star-free fragment (Section 3) and show an algorithm implementing it that works in polynomial space (Section 4). After that, we extend the tableaux method to the full language of DL-PA (Section 5) and then show an algorithm implementing it and that works in time exponential (Section 6). Section 7 discusses some issues and concludes the paper.[3]

2 Dynamic Logic of Propositional Assignments

2.1 Syntax

The vocabulary of DL-PA contains a countable set $\mathbb{P}$ of propositional variables. From this set, we build the set $\mathbb{A}$ of *propositional assignments*, which are the *atomic programs* of the language. Each propositional assignment is a non-empty finite partial function from $\mathbb{P}$ to $\{\bot, \top\}$.[4]

The *language $\mathcal{L}$ of* DL-PA is the the set of formulas φ defined by the BNF:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\pi]\varphi$$
$$\pi ::= \alpha \mid \pi ; \pi \mid \pi \cup \pi \mid \pi^* \mid \varphi?$$

where p ranges over $\mathbb{P}$ and α ranges over $\mathbb{A}$.

To ease notation and readability of programs, we write $+p$ for $(p, \top)$ and $-p$ for $(p, \bot)$. Moreover, we sometimes "forget" some parentheses and curly braces when writing propositional assignments. As a result, the formula $[\{(p, \top), (q, \bot)\}]\varphi$ is rather noted $[+p, -q]\varphi$. In some places, we use the expression $\pm p$ to talk economically about $+p$ and $-p$ at the same time.

The complex programs of DL-PA are constructed as in Dynamic Propositional Logic (PDL) [8]. As well as in PDL, formulas of the form $[\pi]\varphi$ are read "after every possible execution of π, φ is true".

We also use the common abbreviations for the connectives $\top$, $\bot$, $\vee$, $\rightarrow$ and $\leftrightarrow$. The formula $\langle\pi\rangle\varphi$ abbreviates $\neg[\pi]\neg\varphi$. The *star-free fragment of $\mathcal{L}$* is the fragment without the Kleene star operator * and is noted $\mathcal{L}^{-*}$.

The *length* of a formula or a program, given by the function len, is the number of atoms and connectives in the formula or the program. Table 1 defines it formally.

The *closure of φ* is the set $\mathrm{cl}(\varphi)$ defined in Table 2. This is almost the same as the Fisher-Ladner closure [5], which is used to show decidability and complexity results for PDL. But since the atomic programs of DL-PA are sets of assignments, there is a difference here in the definition of $\mathrm{cl}^{\Box}([\alpha]\varphi)$. It takes into account the assignments by adding the domain of the atomic program α.

[3] Proofs of the important theorems are in the appendix.

[4] We note that the original language in [2] is slightly more restrictive: α only assigns a single propositional variable. But, as shown in this paper, it does not change the known decidability and complexity results.

$$\begin{aligned}
\mathrm{len}(p) &= 1 & \mathrm{len}(\alpha) &= |\,\mathrm{dom}(\alpha)| \\
\mathrm{len}(\neg\varphi) &= 1 + \mathrm{len}(\varphi) & \mathrm{len}(\pi_1;\pi_2) &= 1 + \mathrm{len}(\pi_1) + \mathrm{len}(\pi_2) \\
\mathrm{len}(\varphi_1 \land \varphi_2) &= 1 + \mathrm{len}(\varphi_1) + \mathrm{len}(\varphi_2) & \mathrm{len}(\pi_1 \cup \pi_2) &= 1 + \mathrm{len}(\pi_1) + \mathrm{len}(\pi_2) \\
\mathrm{len}([\pi]\varphi) &= 1 + \mathrm{len}(\pi) + \mathrm{len}(\varphi) & \mathrm{len}(\pi^*) &= 1 + \mathrm{len}(\pi) \\
& & \mathrm{len}(\varphi?) &= 1 + \mathrm{len}(\varphi)
\end{aligned}$$

Table 1. Length

$$\begin{aligned}
\mathrm{cl}(p) &= \{p\} & \mathrm{cl}^\Box([\alpha]\varphi) &= \{[\alpha]\varphi\} \cup \mathrm{dom}(\alpha) \\
\mathrm{cl}(\neg\varphi) &= \{\neg\varphi\} \cup \mathrm{cl}(\varphi) & \mathrm{cl}^\Box([\pi_1;\pi_2]\varphi) &= \{[\pi_1;\pi_2]\varphi\} \cup \mathrm{cl}^\Box([\pi_1][\pi_2]\varphi) \\
\mathrm{cl}(\varphi_1 \land \varphi_2) &= \{\varphi_1 \land \varphi_2\} \cup \mathrm{cl}(\varphi_1) \cup \mathrm{cl}(\varphi_2) & \mathrm{cl}^\Box([\pi_1 \cup \pi_2]\varphi) &= \{[\pi_1 \cup \pi_2]\varphi\} \cup \mathrm{cl}^\Box([\pi_1]\varphi) \cup \mathrm{cl}^\Box([\pi_2]\varphi) \\
\mathrm{cl}([\pi]\varphi) &= \mathrm{cl}^\Box([\pi]\varphi) \cup \mathrm{cl}(\varphi) & \mathrm{cl}^\Box([\pi^*]\varphi) &= \{[\pi^*]\varphi\} \cup \mathrm{cl}^\Box([\pi][\pi^*]\varphi) \\
& & \mathrm{cl}^\Box([\varphi_1?]\varphi_2) &= \{[\varphi_1?]\varphi_2\} \cup \mathrm{cl}(\varphi_1)
\end{aligned}$$

Table 2. Closure

The *extended closure of* φ is the set $\mathrm{cl}^+(\varphi)$ containing $\mathrm{cl}(\varphi)$ and the negations of its formulas, i.e., $\mathrm{cl}^+(\varphi) = \mathrm{cl}(\varphi) \cup \{\neg\psi : \psi \in \mathrm{cl}(\varphi)\}$. To ease notation, we sometimes use $\mathbb{P}_\varphi$ to denote the set of propositional variables occurring in φ, i.e. $\mathbb{P}_\varphi = \mathbb{P} \cap \mathrm{cl}(\varphi)$.

The lemma below can be proved with an easy induction on the length of formulas and programs.

Lemma 1.

1. $\mathrm{card}(\mathrm{cl}^\Box([\pi]\varphi)) \leq \mathrm{len}([\pi]\varphi)$
2. $\mathrm{card}(\mathrm{cl}(\varphi)) \leq \mathrm{len}(\varphi)$.
3. $\mathrm{card}(\mathrm{cl}^+(\varphi)) \leq 2\,\mathrm{len}(\varphi)$.

Intuitively, the set of *execution traces* of π is the set $\mathrm{exe}(\pi)$ of sequences of assignments that corresponds to all possible executions of program π. The set $\mathrm{exe}(\varphi)$ corresponds to all possible executions of all programs in φ. These are defined by a mutual recursion, as displayed in Table 3. We use the symbol '()' to denote the empty sequence.

The *length of execution traces*, also given by the function len, is just the number of atomic programs in it. That is: $\mathrm{len}(()) = 0$, $\mathrm{len}(\alpha) = 1$ and $\mathrm{len}(\sigma\alpha) = \mathrm{len}(\sigma) + \mathrm{len}(\alpha)$.

The lemma below can also be proved with an easy induction on the length of programs and formulas:

Lemma 2.

1. *If π does not contain the Kleene star then* $\mathrm{len}(\sigma) \leq \mathrm{len}(\pi)$, *for all* $\sigma \in \mathrm{exe}(\pi)$.
2. *If $\varphi \in \mathcal{L}^{-*}$ then* $\mathrm{len}(\sigma) \leq \mathrm{len}(\varphi)$, *for all* $\sigma \in \mathrm{exe}(\varphi)$.

Note, however, that each $\sigma \in \mathrm{exe}(\pi^*)$ is infinite.

$$
\begin{aligned}
\mathrm{exe}(p) &= \{()\} & \mathrm{exe}(\alpha) &= \{\alpha\}\\
\mathrm{exe}(\neg\varphi) &= \mathrm{exe}(\varphi) & \mathrm{exe}(\pi_1;\pi_2) &= \{\sigma_1\sigma_2 : \sigma_1 \in \mathrm{exe}(\pi_1), \sigma_2 \in \mathrm{exe}(\pi_2)\}\\
\mathrm{exe}(\varphi \land \psi) &= \mathrm{exe}(\varphi) \cup \mathrm{exe}(\psi) & \mathrm{exe}(\pi_1 \cup \pi_2) &= \mathrm{exe}(\pi_1) \cup \mathrm{exe}(\pi_2)\\
\mathrm{exe}([\pi]\varphi) &= \mathrm{exe}(\pi) \cup \mathrm{exe}(\varphi) & \mathrm{exe}(\pi^*) &= \bigcup_{n\in\mathbb{N}_0}\{\sigma_1\dots\sigma_n : \sigma_1,\dots,\sigma_n \in \mathrm{exe}(\pi)\}\\
& & \mathrm{exe}(\varphi?) &= \mathrm{exe}(\varphi)
\end{aligned}
$$

Table 3. Execution traces

2.2 Semantics

A DL-PA *model* is a set $V \subseteq \mathbb{P}$ of propositional variables. When $p \in V$ then p is true, and when $p \notin V$ then p is false.

The interpretation of an assignment α is in terms of a *model update*. The update of a model V by an assignment α is the new model V^α such that:

$$V^\alpha = \{p : V \models \alpha(p)\}$$

where we suppose that when p is not in the domain of α then $\alpha(p)$ equals p. In particular, for the assignment $+p$ we have $V^{+p} = V \cup \{p\}$. Given a sequence of assignments $\alpha_1 \dots \alpha_n$, for the sake of readability, we sometimes write $V^{\alpha_1\cdots\alpha_n}$ instead of $(\cdots(V^{\alpha_1})\cdots)^{\alpha_n}$.

Formulas φ are interpreted as *sets of models* $||\varphi||$, while programs π are interpreted by means of a (unique) *relation between valuations* $||\pi||$. Just as in PDL, the formal definition is by a mutual recursion. It is given in Table 4.

$$
\begin{aligned}
||p|| &= \{V : p \in V\} & ||\alpha|| &= \{\langle V, V'\rangle : V' = V^\alpha\}\\
||\neg\varphi|| &= 2^{\mathbb{P}} \setminus ||\varphi|| & ||\pi_1;\pi_2|| &= ||\pi_1|| \circ ||\pi_2||\\
||\varphi \land \psi|| &= ||\varphi|| \cap ||\psi|| & ||\pi_1 \cup \pi_2|| &= ||\pi_1|| \cup ||\pi_2||\\
||[\pi]\varphi|| &= \{V : \text{if } \langle V, V'\rangle \in ||\pi|| \text{ then } V' \in ||\varphi||\} & ||\pi^*|| &= \bigcup_{n\in\mathbb{N}_0}(||\pi||)^n\\
& & ||\varphi?|| &= \{\langle V, V\rangle : V \in ||\varphi||\}
\end{aligned}
$$

Table 4. Interpretation of the DL-PA connectives

As usual, we also write $V \models \varphi$ to mean that $V \in ||\varphi||$. Moreover, given a formula φ, we say that φ is DL-PA *valid* (noted $\models \varphi$) if and only if $||\varphi|| = 2^{\mathbb{P}}$, and we say that φ is DL-PA *satisfiable* if and only if $||\varphi|| \neq \emptyset$.

For example, the formulas $[+p]\top$, $[+p]\varphi \leftrightarrow \neg[+p]\neg\varphi$, $[\pi]\top$, $[+p]p$ and $[-p]\neg p$ are all DL-PA valid.

2.3 Existing Proof Methods

We now recall the existing methods for both model checking and satisfiability checking in DL-PA. They either use a non-elementary reduction to propositional logic or a quadratic embedding into PDL. We then provide a linear reduction of satisfiability checking to model checking. This justifies our focus on a tableaux method for model checking in the rest of the paper. But first, let us recall some valid principles in DL-PA.

Proposition 1 ([10]). *The following principles are valid in* DL-PA*:*

1. $[\alpha]p \leftrightarrow \alpha(p)$
2. $[\psi?]\varphi \leftrightarrow (\psi \to \varphi)$
3. $[\pi]\neg\varphi \leftrightarrow \neg[\pi]\varphi$
4. $[\pi](\varphi \wedge \psi) \leftrightarrow ([\pi]\varphi \wedge [\pi]\psi)$
5. $[\pi_1;\pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi$
6. $[\pi_1 \cup \pi_2]\varphi \leftrightarrow ([\pi_1]\varphi \wedge [\pi_2]\varphi)$
7. $[\pi^*]\varphi \leftrightarrow (\varphi \wedge [\pi][\pi^*]\varphi)$
8. *From* $\psi \to (\varphi \wedge [\pi^*]\psi)$ *infer* $\psi \to [\pi^*]\varphi$

It follows from Proposition 1.1–1.6 plus the rule of substitution of valid equivalences that the star-free fragment of DL-PA is reducible to propositional logic. This however fails to provide an efficient theorem proving method because the reduced formula might be exponentially longer than the original formula. In [2], it is also shown that the Kleene star can be eliminated in DL-PA, i.e., there is an algorithm that translates every formula in $\mathcal{L}$ to an equivalent formula in $\mathcal{L}^{-*}$. Such translation, however, also leads to much longer formulas. In fact, this is a non-elementary reduction because it starts from the innermost Kleene star operator.

Satisfiability checking in DL-PA is shown to be in EXPTIME in [2]. The proof is given via a translation to satisfiability checking in PDL. For every DL-PA formula φ, the translation tr returns a PDL formula which is obtained by just replacing each assignment $\pm p$ by an abstract PDL program $a_{\pm p}$. To guarantee that the abstract programs behave the same way as the original assignment, the following set of formulas Γ_φ is also used:

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

Proposition 2 ([2]). *Let* U_φ *be the* PDL *program* $(\bigcup_{p\in\mathbb{P}_\varphi}(a_{+p} \cup a_{-p}))^*$. *For every* DL-PA *formula* φ, φ *is* DL-PA *satisfiable if and only if*

$$\mathrm{tr}(\varphi) \wedge [U_\varphi]\left(\bigwedge \Gamma_\varphi\right)$$

is PDL *satisfiable.*

Note that, even though this reduction is polynomial, a quadratically longer formula is produced. Precisely, the size of Γ_φ is bounded by $5\,\mathrm{len}(\varphi)^2$. Moreover, if we consider the star-free fragment of DL-PA, this transformation is sub-optimal, because of the Kleene star operator in U_φ.[5]

If follows from the next result that satisfiability checking in DL-PA can be linearly reduced to model checking in DL-PA.

Proposition 3. *Let a formula $\varphi \in \mathcal{L}$ be given. Let $\mathbb{P}_\varphi = \{p_1, \ldots, p_n\}$. and let M_φ be the* DL-PA *program $(+p_1 \cup -p_1); \ldots; (+p_n \cup -p_n)$. Formula φ is satisfiable if and only if $V \models \langle M_\varphi \rangle \varphi$ for any model V.*

Proof. It suffices to see that the interpretation of the program M_φ relates all possible valuations in the vocabulary of φ, while leaving the other variables unchanged. □

The operation $[M_\varphi]$ works as a master modality and thus $\langle M_\varphi \rangle$ works as its dual. Because it does not contain the Kleene star operator, the length of $\langle M_\varphi \rangle \varphi$ is bounded by $3\,\mathrm{len}(\varphi)$. Also note that, in particular, φ is satisfiable if and only if $V = \emptyset$ satisfies $\langle M_\varphi \rangle \varphi$. This means that the input $(V, \langle M_\varphi \rangle \varphi)$ for the model checking problem is also linear on the length of φ. Therefore, in order to perform satisfiability checking in DL-PA, one could take advantage of an efficient algorithm for model checking in DL-PA. This motivates the tableaux methods presented in the next section.

Before concluding this section, let us recall that, in DL-PA, model checking has the same computational complexity as satisfiability checking. This follows from Proposition 3 above and Proposition 4 below.

Proposition 4 ([2]). *For every valuation V and formula φ, $V \in ||\varphi||$ if and only if the formula*

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

is DL-PA *satisfiable.*

3 A Tableaux Method for Star-Free DL-PA

In this section, we define a model checking procedure for the star-free fragment of DL-PA using analytic tableaux. We start with some useful definitions.

A *labeled formula* is a pair $\lambda = \langle \sigma, \varphi \rangle$, where $\sigma = \alpha_1 \ldots \alpha_n$ is a (possibly empty) sequence of propositional assignments and $\varphi \in \mathcal{L}$. A *branch* is a set of labelled formulas.

Definition 1 (Tableau). *Let $V \subseteq \mathbb{P}$ and $\varphi_0 \in \mathcal{L}^{-*}$. The initial branch for (V, φ_0) is the the set $b_0 = \{\langle (), p \rangle : p \in \mathbb{P}_{\varphi_0} \cap V\} \cup \{\langle (), \neg p \rangle : p \in \mathbb{P}_{\varphi_0} \setminus V\} \cup \{\langle (), \varphi_0 \rangle\}$. A tableau for (V, φ_0) is a set of branches T that satisfies one of the following two conditions:*

1. *$T = \{b_0\}$, which is called the* initial tableau *for (V, φ_0).*

[5] For the star-free fragment, a transformation without the Kleene star operator is also possible. In this case, the program $\bigcup_{p \in \mathbb{P}_\varphi} (a_{+p} \cup a_{-p})$ must be iterated up to $\mathrm{len}(\varphi)$, but this leads to an even longer formula.

2. *$T = (T' \setminus \{b\}) \cup B$, where T' is a tableau for (V, φ_0) containing the branch b and B is a set of k branches $\{b \cup b_1, \ldots, b \cup b_k\}$ generated by one of the following* tableau rules *below:*[6]

   (R¬) $\langle \sigma, \neg\neg\varphi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, \varphi \rangle\}$.
   (R∧) $\langle \sigma, \varphi \wedge \psi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, \varphi \rangle, \langle \sigma, \psi \rangle\}$.
   (R∨) $\langle \sigma, \neg(\varphi \wedge \psi) \rangle \in b$ *implies* $k = 2$, $b_1 = \{\langle \sigma, \neg\varphi \rangle\}$ *and* $b_2 = \{\langle \sigma, \neg\psi \rangle\}$.
   (R$[\alpha]$) $\langle \sigma, [\alpha]\varphi \rangle \in b$ *implies* $k = 1$ *and*
   $b_1 = \{\langle \sigma\alpha, \varphi \rangle\} \cup \{\langle \sigma\alpha, p \rangle : \alpha(p) = \top\} \cup \{\langle \sigma\alpha, \neg p \rangle : \alpha(p) = \bot\}$.
   (R$\langle\alpha\rangle$) $\langle \sigma, \neg[\alpha]\varphi \rangle \in b$ *implies* $k = 1$ *and*
   $b_1 = \{\langle \sigma\alpha, \neg\varphi \rangle\} \cup \{\langle \sigma\alpha, p \rangle : \alpha(p) = \top\} \cup \{\langle \sigma\alpha, \neg p \rangle : \alpha(p) = \bot\}$.
   (R[?]) $\langle \sigma, [\psi?]\varphi \rangle \in b$ *implies* $k = 2$, $b_1 = \{\langle \sigma, \neg\psi \rangle\}$ *and* $b_2 = \{\langle \sigma, \varphi \rangle\}$.
   (R⟨?⟩) $\langle \sigma, \neg[\psi?]\varphi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, \psi \rangle, \langle \sigma, \neg\varphi \rangle\}$.
   (R[;]) $\langle \sigma, [\pi_1; \pi_2]\varphi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, [\pi_1][\pi_2]\varphi \rangle\}$.
   (R⟨;⟩) $\langle \sigma, \neg[\pi_1; \pi_2]\varphi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, \neg[\pi_1][\pi_2]\varphi \rangle\}$.
   (R[∪]) $\langle \sigma, [\pi_1 \cup \pi_2]\varphi \rangle$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, [\pi_1]\varphi \rangle, \langle \sigma, [\pi_2]\varphi \rangle\}$.
   (R⟨∪⟩) $\langle \sigma, \neg[\pi_1 \cup \pi_2]\varphi \rangle \in b$ *implies* $k = 2$, $b_1 = \{\langle \sigma, \neg[\pi_1]\varphi \rangle\}$ *and* $b_2 = \{\langle \sigma, \neg[\pi_2]\varphi \rangle\}$.
   (RP1) $\{\langle \sigma, p \rangle, \langle \sigma\alpha, \psi \rangle\} \subseteq b$ *for some* ψ *and* $p \notin \mathrm{dom}(\alpha)$ *implies* $k = 1$ *and*
   $b_1 = \{\langle \sigma\alpha, p \rangle\}$.
   (RP2) $\{\langle \sigma, \neg p \rangle, \langle \sigma\alpha, \psi \rangle\} \subseteq b$ *for some* ψ *and* $p \notin \mathrm{dom}(\alpha)$ *implies* $k = 1$ *and*
   $b_1 = \{\langle \sigma\alpha, \neg p \rangle\}$.

The initial tableau corresponds to the input of the problem in the tableau. Rules R¬, R∧ and R∨ are the standard tableaux rules for Boolean connectives. RP1 and RP2 (propagation rules) propagate literals whose the truth value is not changed by assignments: if the model updated by σ satisfies p and α does not change the truth value of p then the model updated by $\sigma\alpha$ also satisfies p. The other rules just reflect the semantic definition of the corresponding programs. For instance, for the rule R$[\alpha]$, if the model updated by the sequence of assignments σ satisfies $[\alpha]\varphi$ then the model updated by the sequence $\sigma\alpha$ satisfies φ. Note that they also correspond to the validities 1–6 in Proposition 1.

A branch b is *blatantly inconsistent* if and only if b contains both $\langle \sigma, \varphi \rangle$ and $\langle \sigma, \neg\varphi \rangle$, for some σ and φ. A branch b is *closed* if and only if it is blatantly inconsistent. A tableau is closed if and only if all its branches are closed. A tableau is *open* if and only if it is not closed.

The idea is that, if there is a closed tableau for the input (V, φ_0) then $V \not\models \varphi_0$. On the other hand, if there is no closed tableau for (V, φ_0) then $V \models \varphi_0$.

Example 1. Table 5 shows how the method can be used to prove that the model $V = \{p, q\}$ does not satisfy the formula $\varphi_0 = \neg[+p \cup -p]q$. In the table, lines 1–3 consist of the initial tableau for the input (V, φ_0). Rule applications are indicated between parentheses on the left of each line. Line 4 is generated by the application of R[∪] to line 3. This generates two different branches. The rule applications continue until both branches are closed.

[6] Some of these rules are also presented in the more traditional numerator-denominator form in Table 10 of the Appendix.

				1.	()	p	
				2.	()	q	
				3.	()	$\neg[+p \cup -p]q$	
4.	()	$\neg[+p]q$	(R$\langle\cup\rangle$: 3)	4.	()	$\neg[-p]q$	(R$\langle\cup\rangle$: 3)
5.	$+p$	p	(R$\langle\alpha\rangle$: 4)	5.	$-p$	$\neg p$	(R$\langle\alpha\rangle$: 4)
6.	$+p$	$\neg q$	(R$\langle\alpha\rangle$: 4)	6.	$-p$	$\neg q$	(R$\langle\alpha\rangle$: 4)
7.	$+p$	q	(RP1: 2, 5)	7.	$+p$	q	(RP1: 2, 5)
8.		(closed)	(6, 7)	8.		(closed)	(6, 7)

Table 5. Tableau for $V = \{p, q\}$ and $\varphi_0 = \neg[+p \cup -p]q$

Example 2. Table 6 shows how the method can be used to prove that the model $V = \emptyset$ satisfy the formula $\varphi_0 = [\neg p?; +p]p$. Note that RP2 is not applicable to the labelled formulas in lines 1 and 5 because $p \in \mathrm{dom}(+p)$. Thus, the branch on the right remains open, which means that $V \models \varphi_0$.

				1.	()	$\neg p$	
				2.	()	$[\neg p?; +p]p$	
				3.	()	$[\neg p?][+p]p$	(R[;]: 2)
4.	()	$\neg\neg p$	(R[?]: 3)	4.	()	$[+p]p$	(R[?]: 3)
5.	()	p	(R$\neg$: 4)	5.	$+p$	p	(R[α]: 4)
		(closed)	(1, 5)			(open)	

Table 6. Tableau for $V = \emptyset$ and $\varphi_0 = [\neg p?; +p]p$

In the sequel, we show the soundness of the method. The idea is to show that, if $V \models \varphi_0$, then successive rule applications can never close the tableau. But first, a useful definition and a lemma are presented.

Definition 2 (Consistent Branch). *A branch b is consistent if and only if $V^\sigma \models \varphi$ for every $\langle\sigma, \varphi\rangle \in b$.*

Lemma 3 (Consistency Preservation). *For each tableau rule ρ, if branch b is consistent, then the set of branches B generated by the application of ρ to b contains a consistent branch.*

Theorem 1 (Soundness). *If $V \models \varphi_0$ then there is no closed tableau for (V, φ_0).*

We now address the completeness of the method. The idea is to show that, if the tableau remains open after all possible applications of the tableau rules, then $V \models \varphi_0$. But first, some useful definitions are presented.

Definition 3 (Witness). *A witness to rule ρ in branch b is a labelled formula $\langle\sigma, \varphi\rangle \in b$ allowing the application of ρ.*

For example, $\langle (), \neg\neg p \rangle$ is a witness to R$\neg$, and $\langle \beta, \neg[+p, -q]p \rangle$ is a witness to R$\langle \alpha \rangle$. Moreover, the formula $\langle \sigma, p \rangle$ is a witness to RP1 in b if there is a formula $\langle \sigma\alpha, \psi \rangle \in b$ and $p \notin \text{dom}(\alpha)$.

Definition 4 (Saturated Tableau). *The label σ in the branch b is saturated under the tableau rule ρ if and only if for each witness $\langle \sigma, \varphi \rangle$ to ρ in b, b contains some b_i generated by the application of ρ to b. The branch b is saturated under the tableau rule ρ if and only if all its labels are saturated. A branch is saturated if and only if it is saturated under all tableau rules. A tableau is saturated if and only if all its branches are saturated.*

Theorem 2 (Completeness). *If there is no closed tableau for (V, φ_0) then $V \models \varphi_0$.*

4 An Optimal Procedure for Star-Free DL-PA

In this section we define an algorithm to check whether $V \models \varphi_0$, for $\varphi_0 \in \mathcal{L}^{-*}$. Such an algorithm is displayed in Table 7. It implements the tableaux method using the recursive function mcTableau. It takes as argument a tableau branch b and returns whether b is consistent. When called with the initial tableau for (V, φ_0) it returns whether $V \models \varphi_0$. The execution of mcTableau explores in a depth-first manner a tree whose nodes are tableau branches and each child is generated by the application of a tableau rule to its parent.

The rules are applied in a specific order and, after the application of a rule, the witness is marked 'non-applicable', thus avoiding an infinite loop. Lines 8–21 perform what is called 'local saturation'. That is, only rules that do not create labelled formulas with different labels than that of the witness are applied. Its first part (lines 8–11) applies rules that do not create more than one branch in the tableau. Its second part (lines 12–21) apples rules that create more than one branch in the tableau. At the end of the local saturation, only witnesses to rules R$[\alpha]$, R$\langle \alpha \rangle$ remain. Note that no new label is created in the local saturation part, which means that there can be no witnesses to rules RP1 and RP2. Then, in lines 22–38 the algorithm performs what is called 'successor creation'. First (line 22), it tests whether there is a successor to be created, i.e., if there is a witness λ to R$[\alpha]$ or R$\langle \alpha \rangle$. It creates the successor (line 23) and then marks the witness as 'non-applicable' (line 24). After that (lines 25–34), it propagates the suitable formulas to the successor, as follows: assume that the witness is $\lambda = \langle \sigma, [\alpha]\psi \rangle$. Then, for every labelled formula $\langle \sigma, [\alpha]\psi' \rangle$ and $\langle \sigma, \langle \alpha \rangle \psi' \rangle$ there must be a labelled formula $\langle \sigma\alpha, \psi' \rangle$ in the successor. This is done in lines 25–29. And also, every labelled formula $\langle \sigma, p \rangle$ (resp. $\langle \sigma, \neg p \rangle$) must be propagated, i.e., there must be a labelled formula $\langle \sigma\alpha, p \rangle$ (resp. $\langle \sigma\alpha, \neg p \rangle$) in the successor b_1. This is done in lines 30–34. The last part (lines 35–37) makes a recursive call to mcTableau with the b_1. The current branch is considered satisfiable if all recursive calls return **true**.

This algorithm has two important features. First, its successor creation part guarantees that each time mcTableau is called with branch b as argument, all the labelled formulas in b have the same label. Second, the first feature implies that the list of successors created during successive recursive calls of mcTableau corresponds to one execution trace from input formula φ_0. These are the key arguments used in the proof of complexity result below.

```
input: $(V, \varphi_0)$
output: $\begin{cases} \textbf{true}, & \text{if } b \text{ is satisfiable} \\ \textbf{false}, & \text{otherwise} \end{cases}$
begin
    mcTableau($b_0$)
end

function mcTableau($b$)
begin
    if $b$ contains an applicable witness $\lambda$ to a rule $\rho \in \{R\neg, R\wedge, R\langle ? \rangle, R[;], R\langle ; \rangle, R[\cup]\}$ then
        $b_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda$ as witness
        mark $\lambda$ as 'non-applicable'
        return mcTableau($b \cup b_1$)
    else if $b$ contains an applicable witness $\lambda$ to a rule $\rho \in \{R\vee, R[?], R\langle \cup \rangle, RC\}$ then
        $B \leftarrow$ the set of branches $\{b_1, \dots, b_n\}$ generated by the application of $\rho$ to $b$ using $\lambda$ as witness
        mark $\lambda$ as 'non-applicable'
        for each $b_i \in B$ do
            if mcTableau($b \cup b_i$) = true then
                return true
            end if
        end for
        return false
    end if
    while there is an atomic program $\alpha$ such that $b$ contains an applicable witness
            $\lambda = \langle \sigma, \varphi \rangle$ to rule $\rho \in \{R[\alpha], R\langle \alpha \rangle\}$, where $\varphi = [\alpha]\psi$ or $\varphi = \langle \alpha \rangle \psi$ do
        $b_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda$ as witness
        mark $\lambda$ as 'non-applicable'
        while $b$ contains an applicable witness $\lambda' = \langle \sigma, \varphi' \rangle$ to rule $\rho \in \{R[\alpha], R\langle \alpha \rangle\}$,
                where $\varphi' = [\alpha]\psi'$ or $\varphi' = \langle \alpha \rangle \psi'$ do
            $b'_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda'$ as witness
            mark $\lambda'$ as 'non-applicable'
            $b_1 \leftarrow b_1 \cup b'_1$
        end while
        while $b \cup b_1$ contains an applicable witness $\lambda''$ to rule $\rho \in \{RP_1, RP_2\}$ do
            $b'_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda''$ as witness
            mark $\lambda''$ as 'non-applicable'
            $b_1 \leftarrow b_1 \cup b'_1$
        end while
        if mcTableau($b_1$) = false then
            return false
        end if
    end while
    return true
end
```

Table 7. Algorithm implementing the tableaux method for star-free DL-PA

Theorem 3 (Termination). *The algorithm in Table 7 halts for every input* (V, φ_0).

Theorem 4 (Complexity). *The amount of memory used by the algorithm in Table 7 is a polynomial function of the length of the input* (V, φ_0).

Therefore, the algorithm in Table 7 works in space polynomial in the length of the input. This is an optimal algorithm, given that the satisfiability problem in star-free DL-PA is PSPACE-complete [10].

5 A Tableaux Method for Full DL-PA

In this section, we define an extension of the tableaux method that also takes into account the Kleene star operator.

Definition 5 (Tableau). *Let* (V, φ_0) *be the input under concern (thus, the initial tableau is the same as in Definition 1). The tableau rules for full* DL-PA *are those of Definition 1 plus the following ones:*

(R$[*]$**)** $\langle \sigma, [\pi^*]\varphi \rangle \in b$ *implies* $k = 1$ *and* $b_1 = \{\langle \sigma, \varphi \rangle, \langle \sigma, [\pi][\pi^*]\varphi \rangle\}$.
(R$\langle * \rangle$**)** $\langle \sigma, \neg[\pi^*]\varphi \rangle \in b$ *implies* $k = 2$, $b_1 = \{\langle \sigma, \neg\varphi \rangle\}$ *and* $b_2 = \{\langle \sigma, \varphi \rangle, \langle \sigma, \neg[\pi][\pi^*]\varphi \rangle\}$.

The two rules above reflect the fix point property of Proposition 1.7. For instance, if the model $V^\sigma \models [\pi^*]\varphi$ then $V^\sigma \models \varphi$ and also $V^\sigma \models [\pi][\pi^*]\varphi$.

Definition 6 (Fulfillment). *An eventuality* $\langle \sigma, \neg[\pi^*]\varphi \rangle)$ *is fulfilled in a tableau branch* b *if and only if there is a (possibly empty) execution trace* $\sigma' \in \text{exe}(\pi)$ *such that* $\langle \sigma\sigma', \neg\varphi \rangle \in b$.

Definition 7 (Closed Branch). *A branch b is closed if and only if (1) b is blatantly inconsistent or (2) b is saturated and contains an unfulfilled eventuality.*

Example 3. Table 8 shows how the method can be used to prove that model $V = \{p, q\}$ does not satisfy the formula $\varphi_0 = \neg[(+p \cup -p)^*]q$. The leftmost branch is closed because it is blatantly inconsistent. In the branch of the middle, the same pattern will be repeated indefinitely. Thus, it is an infinite brunch, but it is saturated. Since the eventuality in line 3 is not fulfilled, it is also closed. The right-most branch is analogous to the one in the middle.

If the input formula contains a sub-formula of the form $\neg[\pi^*]\varphi$, the method invariably creates tableaux with infinite branches that repeat the same pattern over and over again, as in Example 3. The repetition can be detected and it is possible to provide a terminating algorithm. This is presented in Section 6. Here, we show the correctness of the method presented so far.

Lemma 4 (Consistency Preservation). *For each tableau rule* ρ*, if branch b is consistent, then the set of branches B generated by the application of* ρ *to b contains a consistent branch.*

Theorem 5 (Soundness). *If* $V \models \varphi_0$ *then there is no closed tableau for* (V, φ_0).

Theorem 6 (Completeness). *If there is no closed tableau for* (V, φ_0) *then* $V \models \varphi_0$.

	1. () p	
	2. () q	
	3. () $\neg[(+p \cup -p)^*]q$	
4. () $\neg q$ (R$\langle * \rangle$: 3)	4. () $\neg[+p \cup -p][(+p \cup -p)^*]q$ (R$\langle * \rangle$: 3)	
5. (blat. inc.) (2, 4)	5. () $\neg[+p][(+p \cup -p)^*]q$ (R$\langle \cup \rangle$: 3)	$\vdots$
	6. $+p$ p (R$\langle \alpha \rangle$: 5)	(closed)
	7. $+p$ $\neg[(+p \cup -p)^*]q$ (R$\langle \alpha \rangle$: 5)	
	8. $+p$ q (RP1: 2, 6)	
	$\vdots$	
	(closed)	

Table 8. Tableau for $V = \{p, q\}$ and $\varphi_0 = \neg[(+p \cup -p)^*]q$

6 An EXPTIME Procedure for Full DL-PA

In this section, we define a procedure to model check formulas in $\mathcal{L}$. As before, we define an algorithm. Here, it must detect the aforementioned repetitions of the applications of R$\langle * \rangle$ in the tableau. This is done by performing equality tests. A label σ_1 is said to be equal to a label σ_2 if and only if the set of formulas labelled by σ_1 and σ_2 are the same. More formally we have:

Definition 8 (Equality). *Let σ_1 and σ_2 be two labels in the tableau T. Label σ_1 is equal to label σ_2 (noted $\sigma_1 = \sigma_2$) if and only if there are two branches $b_1, b_2 \in T$ such that $\{\varphi : \langle \sigma_1, \varphi \rangle \in b_1\} = \{\varphi : \langle \sigma_2, \varphi \rangle \in b_2\}$.*

An equality test between labels can prevent the tableau to enter in an infinite loop. Then one can try to provide an algorithm that is similar to the one in Section 4, by first adding rules R$[*]$ and R$\langle * \rangle$ in their suitable places and the equality test just before the exploration of a new successor. Such an algorithm works, but is not optimal. For instance, the application of the method to the formula $[(+p_1 \cup -p_1 \cup \cdots \cup +p_n \cup -p_n)^*]p$ creates 2^n different successors from a single tableau branch. Then such a method may explore a tree whose the number of nodes is bounded by $2^{2^{\mathrm{len}(\varphi_0)}}$. However, satisfiability in DL-PA is proven to be in EXPTIME.

A different technique than that in Section 4 must be employed in order to obtain a more efficient method for full DL-PA. Such a technique is implemented in the algorithm of Table 9. It is somewhat similar to the algorithm in Section 4, but there are some important differences. The most important ones are the addition of the equality test in lines 17–19 and the fact that this algorithm now maintains the entire tableau T in memory. It does not uses a recursive function anymore, for it now uses the tableau T as the search tree. Once the initial tableau for (V, φ_0) is created in line 4, it enters a loop that finishes when T is closed or saturated (recall that a branch is also considered to be closed if it is saturated and contains an unfulfilled eventuality). As before, there is a 'local saturation' part (lines 9–16) and a 'successor creation' part (lines 20–33). In lines 36–40, the algorithm tests whether T is still open to return the right answer.

```
1: **input:** $(V, \varphi_0)$
2: **output:** $\begin{cases} \textbf{true}, & \text{if } \varphi_0 \text{ is satisfiable} \\ \textbf{false}, & \text{otherwise} \end{cases}$
3: **begin**
4:   $T \leftarrow \{b_0\}$
5:   **while** $T$ is open and unsaturated **do**
6:     pick an open and unsaturated branch $b \in T$
7:     **while** $b$ is open and unsaturated **do**
8:       pick an open unsaturated label $\sigma$ of $b$
9:       **if** $\lambda = \langle \sigma, \varphi \rangle \in b$ is an applicable witness to a rule $\rho \in \{\mathrm{R}\neg, \mathrm{R}\wedge, \mathrm{R}\langle ? \rangle, \mathrm{R}[;], \mathrm{R}\langle ; \rangle, \mathrm{R}[\cup], \mathrm{R}[*]\}$ **then**
10:        $b_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda$ as witness
11:        mark $\lambda$ as ‘non-applicable’
12:        $T \leftarrow (T \setminus \{b\}) \cup \{b \cup b_1\}$
13:      **else if** $\lambda = \langle \sigma, \varphi \rangle \in b$ is an applicable witness to a rule $\rho \in \{\mathrm{R}\vee, \mathrm{R}[?], \mathrm{R}\langle \cup \rangle, \mathrm{R}\langle * \rangle, \mathrm{RC}\}$ **then**
14:        $\{b_1, b_2\} \leftarrow$ the branches generated by the application of $\rho$ to $b$ using $\lambda$ as witness
15:        mark $\lambda$ as ‘non-applicable’
16:        $T \leftarrow (T \setminus \{b\}) \cup \{b \cup b_1, b \cup b_2\}$
17:      **else if** there is a label $\sigma'$ in $T$ such that $\sigma = \sigma'$ **then**
18:        mark all formulas in $b$ labelled by $\sigma$ as ‘non-applicable’
19:        **if** $\sigma'$ is closed **then** close branch $b$ **end if**
20:      **else if** there is an atomic program $\alpha$ such that $b$ contains an applicable witness
             $\lambda = \langle \sigma, \varphi \rangle$ to rule $\rho \in \{\mathrm{R}[\alpha], \mathrm{R}\langle \alpha \rangle\}$, where $\varphi = [\alpha]\psi$ or $\varphi = \langle \alpha \rangle \psi$ **do**
21:        $b_1 \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda$ as witness
22:        mark $\lambda$ as ‘non-applicable’
23:        **while** $b$ contains an applicable witness $\lambda' = \langle \sigma, \varphi' \rangle$ to rule $\rho \in \{\mathrm{R}[\alpha], \mathrm{R}\langle \alpha \rangle\}$,
               where $\varphi' = [\alpha]\psi'$ or $\varphi' = \langle \alpha \rangle \psi'$ **do**
24:        $b_1' \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda'$ as witness
25:        mark $\lambda'$ as ‘non-applicable’
26:        $b_1 \leftarrow b_1 \cup b_1'$
27:        **end while**
28:        **while** $b \cup b_1$ contains an applicable witness $\lambda''$ to rule $\rho \in \{\mathrm{RP}_1, \mathrm{RP}_2\}$ **do**
29:          $b_1' \leftarrow$ the branch generated by the application of $\rho$ to $b$ using $\lambda''$ as witness
30:          mark $\lambda''$ as ‘non-applicable’
31:          $b_1 \leftarrow b_1 \cup b_1'$
32:        **end while**
33:        $T \leftarrow (T \setminus \{b\}) \cup \{b \cup b_1\}$
34:      **end if**
35:    **end while**
36:  **end while**
37:  **if** $T$ is open **then**
38:    **return true**
39:  **else**
40:    **return false**
41:  **end if**
42: **end**
```

Table 9. Algorithm implementing the tableaux method for $\mathcal{L}$

Theorem 7 (Termination). *The algorithm in Table 9 halts for every input* (V, φ_0).

Theorem 8 (Complexity). *The amount of time used by the algorithm in Table 9 is an exponential function of the length of the input* (V, φ_0).

Thus, the algorithm in Table 9 works in time exponential on len(φ_0). This is as expected, given that the model checking problem in full DL-PA is in EXPTIME [10].

7 Discussion and Conclusion

In this paper, we have defined a linear reduction of satisfiability checking into model checking in DL-PA. We also define analytic tableaux methods for model checking formulas in the star-free fragment and in full DL-PA. The complexity of these methods match the complexity class of their respective problems. In the sequel, we compare such methods to similar approaches and discuss possible improvements and extensions.

Comparisons. The methods presented in this paper have been inspired by others already proposed in the literature. For instance, De Giacomo and Massacci [3] (see also [12]) inspired the technique for the Kleene star. As already mentioned, the naive strategy would generate tableau branches with size exponential in the length of the input formula. The idea of keeping the tree in memory and perform equality tests comes from that work.

Assignments of Propositional Variables to Formulas. DL-PA can be extended with assignments α to formulas in $\mathcal{L}$, instead of the simpler $\{\top, \bot\}$. The corresponding tableau rule R[α] would be as follows:

$$\begin{array}{l|l|c|l}
\multicolumn{4}{c}{\sigma : [\alpha]\varphi} \\
\hline
\sigma : \psi_1 & \sigma : \neg\psi_1 & & \sigma : \neg\psi_1 \\
\sigma : \psi_2 & \sigma : \psi_2 & & \sigma : \neg\psi_2 \\
\vdots & \vdots & & \vdots \\
\sigma : \psi_n & \sigma\alpha : \psi_n & & \sigma : \neg\psi_n \\
\sigma\alpha : p_1 & \sigma\alpha : \neg p_1 & \dots & \sigma\alpha : \neg p_1 \\
\sigma\alpha : p_2 & \sigma\alpha : p_2 & & \sigma\alpha : \neg p_2 \\
\vdots & \vdots & & \vdots \\
\sigma\alpha : p_n & \sigma\alpha : p_n & & \sigma\alpha : \neg p_n \\
\sigma\alpha : \varphi & \sigma\alpha : \varphi & & \sigma\alpha : \varphi
\end{array}$$

where we assume that the domain of α is $\{p_1, \dots, p_n\}$ and let $\alpha(p_i) = \psi_i$.

In spite of the apparent complexity of this tableau rule, we believe that the complexity of the method is not affected in the star-free fragment. For the full language, we have to include a cut rule that ranges over all sub-formulas of the input formula φ_0. The reason is to permit the equality test to work also with all formulas ψ_i that are included in the tableau when the new rule R[α] is applied. Again, we believe that the complexity remains the same.

Other PDL *Connectives.* The integration of converse, complement, intersection and other PDL program connectives is also on our agenda. For instance, we believe that we can apply techniques similar to the ones in [13,6,1] for the converse. In this case though, it is not clear whether complexity (or even decidability) results remain the same. This is subject of future work.

References

1. Pietro Abate, Rajeev Goré, and Florian Widmann. An on-the-fly tableau-based decision procedure for PDL-satisfiability. *Electr. Notes Theor. Comput. Sci.*, 231:191–209, 2009.
2. Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments: a well-behaved variant of PDL. In Orna Kupferman, editor, *Logic in Computer Science (LICS), New Orleans, June 25-28, 2013*, http://www.ieee.org/, juin 2013. IEEE.
3. Giuseppe De Giacomo and Fabio Massacci. Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Information and Computation*, 162(1–2):117–137, 2000.
4. Sylvie Doutre, Andreas Herzig, and Laurent Perrussel. A dynamic logic framework for abstract argumentation. In Chitta Baral and Giuseppe De Giacomo, editors, *Proc. KR 2014*. Morgan Kaufmann, 2014.
5. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.
6. Rajeev Goré and Florian Widmann. Optimal and cut-free tableaux for propositional dynamic logic with converse. In *Automated Reasoning*, volume 6173 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2010.
7. David Harel. Dynamic logic. In Dov M. Gabbay and Franz Günthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. D. Reidel, Dordrecht, 1984.
8. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.
9. Andreas Herzig. Belief change operations: a short history of nearly everything, told in dynamic logic of propositional assignments. In Chitta Baral and Giuseppe De Giacomo, editors, *Proc. KR 2014*. Morgan Kaufmann, 2014.
10. Andreas Herzig, Emiliano Lorini, Frédéric Moisan, and Nicolas Troquard. A dynamic logic of normative systems. In Toby Walsh, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, Barcelona, 2011. Morgan Kaufmann Publishers. erratum at `http://www.irit.fr/~Andreas.Herzig/P/Ijcai11.html`.
11. Andreas Herzig, Pilar Pozos Parra, and François Schwarzentruber. Belief merging in Dynamic Logic of Propositional Assignments. In Christoph Beierle and Carlo Meghini, editors, *International Symposium on Foundations of Information and Knowledge Systems (FoIKS) (FoIKS), Bordeaux*. Springer, 2014.
12. Ullrich Hustadt and Renate A. Schmidt. A comparison of solvers for propositional dynamic logic. In Renate A. Schmidt, Stephan Schulz, and Boris Konev, editors, *PAAR-2010*, volume 9 of *EPiC Series*, pages 63–73. EasyChair, 2012.
13. Linh Anh Nguyen and Andrzej Szałas. An optimal tableau decision procedure for converse-pdl. In *Proceedings of KSE-09*, pages 207–214. IEEE Computer Society, 2009.

A Rules in Numerator-Denominator Form

For the comfort of the reader we present here the tableau rules in the more traditional numerator-denominator form.

$$(\mathrm{R}[\alpha]) \quad \frac{\sigma : [\alpha]\varphi}{\begin{array}{l} \sigma\alpha : p_1 \\ \vdots \\ \sigma\alpha : p_n \\ \sigma\alpha : \neg p_{n+1} \\ \vdots \\ \sigma\alpha : \neg p_{n+m} \\ \sigma\alpha : \varphi \end{array}} \qquad\qquad (\mathrm{R}\langle\alpha\rangle) \quad \frac{\sigma : \neg[\alpha]\varphi}{\begin{array}{l} \sigma\alpha : p_1 \\ \vdots \\ \sigma\alpha : p_n \\ \sigma\alpha : \neg p_{n+1} \\ \vdots \\ \sigma\alpha : \neg p_{n+m} \\ \sigma\alpha : \neg\varphi \end{array}}$$

$$(\mathrm{R}[?]) \quad \frac{\sigma : [\psi?]\varphi}{\sigma : \neg\psi \mid \sigma : \varphi} \qquad\qquad (\mathrm{R}\langle ? \rangle) \quad \frac{\sigma : \neg[\psi?]\varphi}{\begin{array}{l} \sigma : \psi \\ \sigma : \neg\varphi \end{array}}$$

$$(\mathrm{R}[;]) \quad \frac{\sigma : [\pi_1;\pi_2]\varphi}{\sigma : [\pi_1][\pi_2]\varphi} \qquad\qquad (\mathrm{R}\langle ; \rangle) \quad \frac{\sigma : \neg[\pi_1;\pi_2]\varphi}{\sigma : \neg[\pi_1][\pi_2]\varphi}$$

$$(\mathrm{R}[\cup]) \quad \frac{\sigma : [\pi_1 \cup \pi_2]\varphi}{\begin{array}{l} \sigma : [\pi_1]\varphi \\ \sigma : [\pi_2]\varphi \end{array}} \qquad\qquad (\mathrm{R}\langle \cup \rangle) \quad \frac{\sigma : \neg[\pi_1 \cup \pi_2]\varphi}{\sigma : \neg[\pi_1]\varphi \mid \sigma : \neg[\pi_2]\varphi}$$

$$(\mathrm{R}[*]) \quad \frac{\sigma : [\pi^*]\varphi}{\begin{array}{l} \sigma : \varphi \\ \sigma : [\pi][\pi^*]\varphi \end{array}} \qquad\qquad (\mathrm{R}\langle * \rangle) \quad \frac{\sigma : \neg[\pi^*]\varphi}{\sigma : \neg\varphi \mid \sigma : \neg[\pi][\pi^*]\varphi}$$

Table 10. Tableau rules for the operator [ ]. In R$[\alpha]$ and R$\langle\alpha\rangle$, we assume that $\mathrm{dom}(\alpha) = \{p_1, \ldots, p_n, p_{n+1} \ldots, p_{n+m}\}$ and also $\alpha(p_1) = \cdots = \alpha(p_n) = \top$, and $\alpha(p_{n+1}) = \cdots = \alpha(p_{n+m}) = \bot$.

B Proofs

Lemma 3 (Consistency Preservation). For each tableau rule ρ, if branch b is consistent, then the set of branches B generated by the application of ρ to b contains a consistent branch.

Proof. The proofs for the rules R$\neg$, R$\wedge$ and R$\vee$ are easy and left to the reader. For rule R$[\alpha]$, note that, because b is consistent, we have $V^\sigma \models [\alpha]\varphi$. Then $V^{\sigma\alpha} \models \varphi$ by the truth condition for $[\alpha]$. Moreover, by the definition of updates we have:

– $V^{\sigma\alpha} \models p$ for all $p \in \mathrm{dom}(\alpha)$ such that $\alpha(p) = \top$, and

– $V^{\sigma\alpha} \models \neg p$ for all $p \in \mathrm{dom}(\alpha)$ such that $\alpha(p) = \bot$.

For the remaining tableau rules, namely R$\langle\alpha\rangle$, R[;], R$\langle;\rangle$, R[$\cup$], and R$\langle\cup\rangle$, RP1 and RP2, the reasoning is similar and left to the reader. □

Theorem 1 (Soundness). If $V \models \varphi_0$ then there is no closed tableau for (V, φ_0).

Proof. Assume that $V \models \varphi_0$. Then the initial tableau for (V, φ_0) is consistent. It follows from Lemma 3 that all tableaux for φ_0 have at least one consistent branch b. Now, towards a contradiction, assume that b is closed. Then b contains both $\langle\sigma, \psi\rangle$ and $\langle\sigma, \neg\psi\rangle$, for some σ and ψ. However, since b is consistent, $V^\sigma \models \psi$ and $V^\sigma \models \neg\psi$, which is a contradiction. Therefore, b is not closed neither is the tableau containing it. □

Theorem 2 (Completeness). If there is no closed tableau for (V, φ_0) then $V \models \varphi_0$.

Proof. Suppose there is no closed tableau for (V, φ_0). Let b be an open and saturated branch of a tableau for (V, φ_0). We prove that, for every pair $\langle\sigma, \psi\rangle \in b$, we have $V^\sigma \models \psi$. The proof is done by induction on $\mathrm{len}(\sigma) + \mathrm{len}(\psi)$ and, in particular, establishes that $V \models \varphi_0$, since $\langle(), \varphi_0\rangle \in b$.

Induction base: We consider two cases:

– Let $\sigma = ()$ and $\psi = p \in \mathbb{P}$. Then $V \models p$, otherwise b would be closed since $b_0 \subseteq b$.
– Let $\sigma = ()$ and $\psi = \neg p$. Then $V \models \neg p$, otherwise b would be closed since $b_0 \subseteq b$.

Induction Hypothesis: For every $\langle\sigma, \psi\rangle \in b$, if $\mathrm{len}(\sigma) + \mathrm{len}(\psi) \leq n$, then $V^\sigma \models \psi$.

Induction step: Let $\mathrm{len}(\sigma) + \mathrm{len}(\psi) = n + 1$. We only give some of all possible cases:

– Let $\sigma = \sigma_1\alpha$ and $\psi = p \in \mathbb{P}$. We consider two sub-cases:
  - Let $p \notin \mathrm{dom}(\alpha)$. We have $\langle\sigma_1, \neg p\rangle \notin b$, otherwise b would be closed, because it is saturated under RP2. Then we have $\langle\sigma_1, p\rangle \in b$, because the branch is saturated under RP1 and $b_0 \subseteq b$. By induction hypothesis, we have $V^{\sigma_1} \models p$. Since $p \notin \mathrm{dom}(\alpha)$, we also have $V^{\sigma_1\alpha} \models p$.
  - Let $p \in \mathrm{dom}(\alpha)$. We then must have $\alpha(p) = \top$: otherwise b would not only contain $\langle\sigma_1\alpha, p\rangle$, but also $\langle\sigma_1\alpha, \neg p\rangle$ (by the application of rule R[α]) and b would therefore be closed. Hence, by the definition of updates $p \in V^{\sigma_1\alpha}$. The latter means that $V^{\sigma_1\alpha} \models p$.
– Let $\sigma = \sigma_1\alpha$ and $\psi = \neg p$. Again, we consider two sub-cases:
  - Let $p \notin \mathrm{dom}(\alpha)$. We have $\langle\sigma_1, p\rangle \notin b$: otherwise, b would be closed, since it is saturated under RP1. Then we have $\langle\sigma_1, \neg p\rangle \in b$, because the branch is saturated under RP2 and $b_0 \in b$. Then $p \notin V^{\sigma_1}$ (by the induction hypothesis) and thus $p \notin V^{\sigma_1\alpha}$. Then $V^{\sigma_1\alpha} \models \neg p$.
  - Let $p \in \mathrm{dom}(\alpha)$. Note that we have $\alpha(p) = \bot$: otherwise b would be closed, because it would contain $\langle\sigma_1\alpha, p\rangle$ and $\langle\sigma_1\alpha, \neg p\rangle$, since it is saturated under R[α]. Then $p \notin V^{\sigma_1\alpha}$ (by its definition) Then $V^{\sigma_1\alpha} \models \neg p$.
– Let $\psi = \neg\neg\psi_1$. If $\langle\sigma, \neg\neg\psi_1\rangle \in b$ then $\langle\sigma, \psi_1\rangle \in b$ (because b is saturated under R$\neg$). By Induction Hypothesis we have $V^\sigma \models \psi_1$. Therefore $V^\sigma \models \neg\neg\psi_1$ by the truth condition for negation.

– Let $\psi = \psi_1 \wedge \psi_2$. If $\langle \sigma, \psi_1 \wedge \psi_2 \rangle \in b$ then $\langle \sigma, \psi_1 \rangle, \langle \sigma, \psi_2 \rangle \in b$ (because b is saturated under R$\wedge$). By Induction Hypothesis we have $V^\sigma \models \psi_1$ and $V^\sigma \models \psi_2$. Therefore $V^\sigma \models \psi_1 \wedge \psi_2$ by the truth condition for conjunction.
– Let $\psi = \neg(\psi_1 \wedge \psi_2)$. If $\langle \sigma, \neg(\psi_1 \wedge \psi_2) \rangle \in b$ then $\langle \sigma, \neg\psi_1 \rangle \in b$ or $\langle \sigma, \neg\psi_2 \rangle \in b$ (because b is saturated under R$\vee$). By Induction Hypothesis we have $V^\sigma \models \neg\psi_1$ or $V^\sigma \models \neg\psi_2$. Therefore $V^\sigma \models \neg(\psi_1 \wedge \psi_2)$ by the truth conditions for negation and conjunction.
– Let $\psi = [\alpha]\psi_1$. If $\langle \sigma, [\alpha]\psi_1 \rangle \in b$ then $\langle \sigma\alpha, \psi_1 \rangle \in b$ (because b is saturated under R[α]). Then, $V^{\sigma\alpha} \models \psi_1$ (by Induction Hypothesis, because $\text{len}(\sigma) + \text{len}(\alpha) + \text{len}(\psi_1) < \text{len}(\sigma) + \text{len}([\alpha]\psi_1) = \text{len}(\sigma) + 1 + \text{len}(\alpha) + \text{len}(\psi_1)$). Therefore, $V^\sigma \models [\alpha]\psi_1$ (by definition).
– Let $\psi = [\psi_1?]\psi_2$. If $\langle \sigma, [\psi_1?]\psi_2 \rangle \in b$ then, because b is saturated under rule R[?], we consider two sub-cases. Either (1) $\langle \sigma, \neg\psi_1 \rangle \in b$ or (2) $\langle \sigma, \psi_2 \rangle \in b$. In both sub-cases, we have $V^\sigma \models \psi_1$ implies $V^\sigma \models \psi_2$ (by Induction Hypothesis). Therefore, $V^\sigma \models [\psi_2?]\psi_2$ (by definition).
– Let $\psi = [\pi_1; \pi_2]\psi_1$. If $\langle \sigma, [\pi_1; \pi_2]\psi_1 \rangle \in b$ then $\langle \sigma, [\pi_1][\pi_2]\psi_2 \rangle \in b$ (because b is saturated under rule R[;]). Then it is easy to see that $\langle \sigma\sigma_1, [\pi_2]\psi_1 \rangle \in b$, for all execution traces $\sigma_1 \in \text{exe}(\pi_1)$ Then $V^{\sigma\sigma_1} \models [\pi_2]\psi_1$ (by Induction Hypothesis, since $\text{len}(\sigma) + \text{len}(\sigma_1) + 1 + \text{len}(\pi_2) + \text{len}(\psi_1) = \text{len}(\sigma\sigma_1) + \text{len}([\pi_2]\psi_1) < \text{len}(\sigma) + \text{len}([\pi_1; \pi_2]\psi_1) = \text{len}(\sigma) + 1 + \text{len}(\pi_1) + 1 + \text{len}(\pi_2) + \text{len}(\psi_1)$). The latter means that $V^\sigma \models [\pi_1; \pi_2]\psi_1$.
– Let $\psi = [\pi_1 \cup \pi_2]\psi_1$. If $\langle \sigma, [\pi_1 \cup \pi_2]\psi_1 \rangle \in b$ then $\langle \sigma, [\pi_1]\psi_1 \rangle, \langle \sigma, [\pi_2]\psi_1 \rangle \in b$ (because b is saturated under rule R[$\cup$]). Then, $V^\sigma \models [\pi_1]\psi_1$ and $V^\sigma \models [\pi_2]\psi_1$ (by Induction Hypothesis). Therefore, $V^\sigma \models [\pi_1 \cup \pi_2]\psi_1$ (by definition).
– The cases where $\psi = \neg[\pi]\psi_1$ are analogous to the last ones. □

Theorem 3 (Termination). The algorithm in Table 7 halts for every input (V, φ_0).

Proof. It is enough to show that function mcTableau is eventually called with an argument b which is either a closed or a saturated branch. Assume that, during the execution, branch b is passed as argument to a call of function mcTableau. Assume that b contains a witness λ to one of the tableau rules. Then the function will be called recursively with a new branch b_1 wherein λ is marked ‘non-applicable’, so it will never be a witness again. Moreover, b_1 differs from b by some additional labelled formulas that are shorter than λ. Therefore, by an easy induction on the length of labelled formulas, we show that function mcTableau will eventually generate a branch b_1 which is either closed or saturated. The details are omitted. □

Theorem 4 (Complexity). The amount of memory used by the algorithm in Table 7 is a polynomial function of the length of the input (V, φ_0).

Proof. Each call of function mcTableau generates a new tableau branch. This branch remains in memory during the recursive calls and is released once the present call of the function finishes its execution returning **true** or **false**. Therefore, to prove our claim, it is enough to show that the amount of memory used by each tableau branch is a polynomial function of $\text{len}(\varphi_0)$ and that the number of successive recursive calls to mcTableau is a polynomial function of $\text{len}(\varphi_0)$ as well.

First, we observe that the initial branch b_0 contains only formulas from $\mathrm{cl}^+(\varphi_0)$.

Second, each time mcTableau is called with branch b as argument, all the labelled formulas in b have the same label. Since the amount of memory used by a branch is bounded by the number of different labelled formulas it contains, it then follows from Lemma 1 that the number of different labelled formulas in b is bounded by $2\,\mathrm{len}(\varphi_0)$.

Third, the number of successive recursive calls during the local saturation of the tableau is bounded by the number of different labelled formulas a successor may contain. This number is $2\,\mathrm{len}(\varphi_0)$, again by Lemma 1. Now, recall that the list of successors created by the algorithm during successive recursive calls of mcTableau corresponds to one execution trace from input formula φ_0. The length of each execution trace is bounded by $\mathrm{len}(\varphi_0)$, by Lemma 2. Then the number of successive recursive calls that create new successors is bounded by $\mathrm{len}(\varphi_0)$. Finally, the total number of successive recursive calls to mcTableau is bounded by $2\,\mathrm{len}(\varphi_0)^2$.

We then conclude that the amount of memory used by the algorithm is proportional to $4\,\mathrm{len}(\varphi_0)^3$. $\square$

Lemma 4 (Satisfiability Preservation). For each tableau rule ρ, if branch b is consistent, then the set of branches B generated by the application of ρ to b contains a consistent branch.

Proof. For the rules that are already part of in the method for star-free DL-PA the proof is the same as in the proof of Lemma 3. For the other cases, we have:

– Rule R$[*]$: If $V^\sigma \models [\pi^*]\varphi$ then, by Proposition 1, $V^\sigma \models \varphi$ and $V^\sigma \models [\pi][\pi^*]\varphi$.
– Rule R$\langle * \rangle$: If $V^\sigma \models \neg[\pi^*]\varphi$ then $V^\sigma \not\models [\pi^*]\varphi$, and the latter is the case iff $V^\sigma \models \neg\varphi$ or $V^\sigma \models \neg[\pi][\pi^*]\varphi$, again due to Proposition 1. $\square$

Theorem 5 (Soundness). If $V \models \varphi_0$ then there is no closed tableau for (V, φ_0).

Proof. Assume that $V \models \varphi_0$. Then, the initial tableau for (V, φ_0) is consistent. It follows from Lemma 4 that all tableaux for (V, φ_0) have at least one consistent branch b. Now, towards a contradiction, assume that b is closed. Then, either (1) b contains both $\langle \sigma, \varphi \rangle$ and $\langle \sigma, \neg\varphi \rangle$, for some σ and φ; or (2) b is saturated and contains an unfulfilled eventuality $\langle \sigma, \neg[\pi^*]\varphi \rangle$. In the first case, (because b is consistent) $V^\sigma \models \varphi$ and $V^\sigma \models \neg\varphi$, which is a contradiction. In the second case, (again because b is consistent) $V^\sigma \models \neg[\pi^*]\varphi$. Moreover, b contains $\langle \sigma\sigma', \varphi \rangle$, for all execution traces $\sigma' \in \mathrm{exe}(\pi^*)$, by the saturation of R$\langle * \rangle$ and because the eventuality is not fulfilled. Then, $V^{\sigma\sigma'} \models \varphi$, for all execution traces $\sigma' \in \mathrm{exe}(\pi^*)$ (because the branch is consistent). Then, $V^\sigma \models [\pi^n]\varphi$, for all $n \geq 0$. The latter implies $V^\sigma \models [\pi^*]\varphi$, which contradicts the hypothesis. So b is not closed, and therefore the tableau containing b cannot be closed. $\square$

Theorem 6 (Completeness). If there is no closed tableau for (V, φ_0) then $V \models \varphi_0$.

Proof. The proof is essentially the same as for Theorem 2. We only add the induction step case for the Kleene star operator here:

- Let $\psi = [\pi^*]\psi_1$. If $\langle \sigma, [\pi^*]\psi_1 \rangle \in b$ then $\langle \sigma\sigma', \psi_1 \rangle \in b$, for all execution traces $\sigma' \in \mathrm{exe}(\pi^*)$ (because b is saturated, in particular, under rule R$[*]$). Then, $V^{\sigma\sigma'} \models \psi$, for all execution traces $\sigma' \in \mathrm{exe}(\pi^*)$ (by Induction Hypothesis), iff $V^\sigma \models [\pi^n]\psi$, for all $n \in \mathbb{N}_0$, iff $V^\sigma \models [\pi^*]\psi$.

For the case where $\psi = \neg[\pi^*]\psi$ we use the fact that the branch b is not closed, which means that the eventuality is fulfilled in b, by definition. $\square$

Theorem 7 (Termination). The algorithm in Table 9 halts for every input (V, φ_0).

Proof. It is enough to show that the algorithm eventually generates a tableau such that all its branches are either closed or saturated. The algorithm has two parts: local saturation and successor creation.

First, assume that the latest generated tableau T contains an open and unsaturated branch b with a witness λ to one of the tableau rules of the local saturation part. Then the algorithm updates T by marking λ as 'non-applicable', so it will never be a witness again. Moreover, the new branches of the updated tableau T differ from the old ones by somme additional labelled formulas that are either shorter than λ or (in the case of rules $\mathrm{R}[*]$ and $\mathrm{R}\langle * \rangle$) that can no longer be witnesses to these rules any more. Therefore, by an easy induction on the length of labelled formulas, we show that the algorithm will eventually generate a tableau such that all its branches are either closed or saturated for these rules. The details are omitted.

Second, assume that the latest generated tableau T contains an open and unsaturated branch b with a witness λ to one of the tableau rules $\mathrm{R}[\alpha]$ and $\mathrm{R}\langle\alpha\rangle$. Then it marks λ as 'non-applicable' and the updated T contains new branches with somme additional labelled formulas $\langle\sigma, \psi\rangle$, where σ is a new label and $\psi \in \mathrm{cl}^+(\varphi_0)$. Since $\mathrm{cl}^+(\varphi_0)$ is finite, there cannot be an infinite number of different labelled formulas whose labels different from another label of the tableau. Thus, the equality test will eventually succeeds and new successors won't be created indefinitely. $\square$

Theorem 8 (Complexity). The amount of time used by the algorithm in Table 9 is an exponential function of the length of the input (V, φ_0).

Proof. The amount of time used by the algorithm in Table 9 is bounded by the number of rule applications during the execution and the time spent on the equality tests.

First, for each successor, the local saturation part performs at most $2\,\mathrm{len}(\varphi_0)$ rule applications, because it is the maximum size of $\mathrm{cl}^+(\varphi_0)$ (by Lemma 1). Second, there can be at most $2^{2\,\mathrm{len}(\varphi_0)}$ different labels in the entire tableau T, because it is the maximum size of $\mathcal{P}(\mathrm{cl}^+(\varphi_0))$. Then the successor creation part can generate at most $2^{2\,\mathrm{len}(\varphi_0)}$ different labels until the equality test succeeds. Moreover, the equality test itself takes time proportional to $2^{2\,\mathrm{len}(\varphi_0)}$, by the same reasons.

Overall, the amount of time used by the algorithm is bounded by $2\,\mathrm{len}(\varphi_0) \times 2^{2\,\mathrm{len}(\varphi_0)} \times 2^{2\,\mathrm{len}(\varphi_0)}$ equals to $2^{4\,\mathrm{len}(\varphi_0)+1}\,\mathrm{len}(\varphi_0)$, which is an exponential function of the length of the input formula φ_0. $\square$