Tableaux methods for propositional dynamic logics with separating parallel composition

Philippe Balbiani, Joseph Boudou

Tableaux Methods for Propositional Dynamic Logics with Separating Parallel Composition

Philippe Balbiani and Joseph Boudou(✉)

Institut de Recherche en Informatique de Toulouse,
CNRS — Toulouse University, Toulouse, France
joseph.boudou@irit.fr

Abstract. PRSPDL is a propositional dynamic logic with an operator for parallel compositions of programs. We first give a complexity upper bound for this logic. Then we focus on the class of $\lhd$-deterministic frames and give tableaux methods for two fragments of PRSPDL over this class of frames.

1 Introduction

Propositional dynamic logic (PDL) is a multi-modal logic designed to reason about the behaviors of programs [11]. With each program α is associated a modal operator $[\alpha]$, formulas $[\alpha]\varphi$ being read "all executions of α from the current state lead to a state where φ holds". The set of programs is structured by some operators: the sequential composition $\alpha\,;\beta$ of programs α and β corresponds to the composition of the accessibility relations $R(\alpha)$ and $R(\beta)$; test $\varphi?$ on formula φ corresponds to the identity relation restricted to the states at which φ holds; the iteration α^* corresponds to the reflexive and transitive closure of $R(\alpha)$. PDL has been extensively studied and a great deal is known about its complexity and proof theory [9,11,12,15,16,18]. Moreover, since PDL's programs are abstract, variants of PDL has been devised for different fields, like knowledge representation and linguistic.

A limitation of PDL is the lack of constructs to reason about concurrency. Different extensions of PDL have been devised to overcome this limitation, for instance interleaving PDL [1], PDL with intersection (IPDL) [8] and the concurrent dynamic logic [17]. PDL with storing, recovering and parallel composition (PRSPDL) [4] is another extension of PDL with a construct for parallel composition of programs. The key difference of PRSPDL is that, for the execution of the program $\alpha \parallel \beta$, α and β are executed in parallel *on two different substates* of the initial state. Hence, $\alpha \parallel \beta$ being executable at some state does not imply that α or β is executable at that state. Since states can be separated in substates and merged back, PRSPDL is related to the Boolean logic of bunched implication (BBI) [19]. Indeed, a multiplicative conjunction semantically similar

to the one found in BBI can be defined in PRSPDL. Hence PRSPDL is a modal logics of separation like logics in [5–7], and is closely related to the process logic MBIc [6]. The differences between PRSPDL and MBIc are the lack of sequential compositions in MBIc making it strictly less expressive [2] and the associativity of the separation relation making the satisfiability problem harder [14]. The combination of separation and concurrency provided by PRSPDL suggests interesting applications. For instance, in the field of program verification, a dynamic and concurrent logic on heaps akin to separation logics [10,20] may be envisioned.

PRSPDL has many similarities with IPDL. Like IPDL, PRSPDL lacks the tree model property. Moreover, due to formulas of the form $[\alpha \parallel \beta]\,\varphi$, there is no sets of formulas of the language comparable to the Fischer-Ladner closure for PDL [11] on which the filtration method could be applied. Hence, studying PRSPDL's computability is hard and the only result currently known about the computability of PRSPDL is its undecidability when interpreted over the class of separated frames [3]. This difficulties, added to the usual complications due to the iteration construct, make the conception of a tableaux method for PRSPDL a real challenge. We overcome all this difficulties by adapting techniques from [16]: compound programs are allowed as edge's labels of the constructed model and new atomic formulas are used to identify states reachable by some programs. The added value of our paper consists in the presence of the iteration construct and in a new extended definition of the Fischer-Ladner closure.

In this paper, three variants of PRSPDL are studied. First, PRSPDL, formally defined in Sect. 2, is proved in Sect. 3 to be faithfully translatable into IPDL with converse. This result conveys a 2EXPTIME complexity upper bound. Then the fragment of PRSPDL without the special programs of storing and recovering, interpreted over the class of $\lhd$-deterministic frames, is considered. A Fischer-Ladner closure for an extension of this fragment is defined in Sect. 4 and a sound and complete tableaux method is exhibited in Sect. 5. Finally, an optimal decision procedure for the fragment of PRSPDL without storing, recovering and iteration, interpreted over $\lhd$-deterministic frames, is given in Sect. 6, proving this fragment to be PSPACE-complete.

2 PRSPDL

Let Π_0 a countable set of atomic programs (denoted a, b.....) and Φ_0 be a countable set of propositional variables (denoted p, q.....). The sets Π and Φ of programs and formulas are defined as follows:

$$\begin{aligned} \alpha, \beta &:= a \mid (\alpha\,;\beta) \mid \varphi? \mid \alpha^* \mid (\alpha \parallel \beta) \mid s_1 \mid s_2 \mid r_1 \mid r_2 \\ \varphi &:= p \mid \bot \mid \neg\varphi \mid [\alpha]\,\varphi \end{aligned}$$

We define the abbreviations $\top \doteq \neg\bot$ and $\langle\alpha\rangle\varphi \doteq \neg\,[\alpha]\,\neg\varphi$. The Boolean operators can be defined too, for instance $\varphi \to \psi \doteq [\varphi?]\,\psi$. Moreover, a multiplicative conjunction related to BBI may be defined as $\varphi * \psi \doteq \neg\,[\varphi? \parallel \psi?]\,\bot$. Parentheses may be omitted for clarity, but they are taken into account when counting

occurrences of symbols. Double negations are implicitly eliminated. We write $|\alpha|$ and $|\varphi|$ for the number of occurrences of symbols in any program α and any formula φ respectively. We define two fragments of PRSPDL's language:

- $\mathcal{L}_{;?*\|}$ is the set of PRSPDL's formulas and programs with no occurrences of the symbols s_1, s_2, r_1 and r_2;
- $\mathcal{L}_{;?\|}$ is the set of PRSPDL's formulas and programs with no occurrences of the symbols s_1, s_2, r_1, r_2 and *.

A frame is a tuple $(W, R, \lhd)$ where W is a non-empty set of states, R is a function associating a binary relation over W to each atomic program and $\lhd$ is a ternary relation over W. Intuitively, $x\,R\,(a)\,y$ means that the program a can be executed in state x, reaching state y. Similarly, $x \lhd (y, z)$ means that x can be split into the substates y and z or equivalently that y and z can be merged to obtain x. When the merging of states is functional, the frame is said to be $\lhd$-deterministic. This is a common restriction, for instance in separation logics [10,20]. Formally, a frame is $\lhd$-deterministic iff for all $x, y, w_1, w_2 \in W$, if $x \lhd (w_1, w_2)$ and $y \lhd (w_1, w_2)$ then $x = y$. The class of all frames is denoted by $\mathcal{C}_{\text{all}}$ and the class of $\lhd$-deterministic frames by $\mathcal{C}_{\lhd\text{-det}}$. A model is a tuple $(W, R, \lhd, V)$ where $(W, R, \lhd)$ is a frame and V is a function associating a subset of W to each propositional variable. A model is $\lhd$-deterministic iff its frame is $\lhd$-deterministic. The forcing relation $\vDash$ is defined by parallel induction along with the extension of R to all programs:

$$
\begin{array}{ll}
\mathcal{M}, x \vDash p & \text{iff } x \in V(p) \\
\mathcal{M}, x \vDash \bot & \text{never} \\
\mathcal{M}, x \vDash \neg\varphi & \text{iff } \mathcal{M}, x \nvDash \varphi \\
\mathcal{M}, x \vDash [\alpha]\,\varphi & \text{iff } \forall z \in W,\ xR\,(\alpha)z \text{ implies } \mathcal{M}, z \vDash \varphi \\
xR\,(\alpha\,;\beta)y & \text{iff } \exists z \in W,\ xR\,(\alpha)z \text{ and } zR\,(\beta)y \\
xR\,(\varphi?)y & \text{iff } x = y \text{ and } \mathcal{M}, x \vDash \varphi \\
xR\,(\alpha^*)y & \text{iff } xR\,(\alpha)^*y \\
 & \qquad \text{where} R\,(\alpha)^* \text{is the reflexive transitive closure of} R\,(\alpha) \\
xR\,(\alpha \parallel \beta)y & \text{iff } \exists w_1, w_2, w_3, w_4 \in W, \\
 & \qquad x \lhd (w_1, w_2)\,, w_1R\,(\alpha)w_3, w_2R\,(\beta)w_4 \text{ and } y \lhd (w_3, w_4) \\
xR\,(s_i)y & \text{iff } \exists z_1, z_2 \in W,\ y \lhd (z_1, z_2) \text{ and } x = z_i \\
xR\,(r_i)y & \text{iff } \exists z_1, z_2 \in W,\ x \lhd (z_1, z_2) \text{ and } y = z_i
\end{array}
$$

A formula φ is *satisfiable in a class $\mathcal{C}$ of frames* iff there exists a model $\mathcal{M} = (W, R, \lhd, V)$ and a state $w \in W$ such that $(W, R, \lhd) \in \mathcal{C}$ and $\mathcal{M}, w \vDash \varphi$. The satisfiability problem for a fragment $\mathcal{L}$ of PRSPDL over a class $\mathcal{C}$ of frames is the decision problem answering whether a formula in $\mathcal{L}$ is satisfiable in $\mathcal{C}$.

3 Complexity Upper Bound for PRSPDL

In order to illustrate the close relationship between PRSPDL and IPDL, we provide a faithful translation from PRSPDL to PDL with intersection and

converse (ICPDL) [15]. This translation conveys an upper bound for the complexity of the satisfiability problem of PRSPDL with respect to $\mathcal{C}_{\text{all}}$. Given the same sets Π_0 and Φ_0 as for PRSPDL and three new atomic programs b_0, b_1, b_2, the language of ICPDL is defined by:

$$\begin{aligned}\alpha, \beta &:= a \mid (\alpha\,;\beta) \mid (\alpha \cup \beta) \mid \varphi? \mid \alpha^* \mid \alpha^- \mid (\alpha \cap \beta) \mid b_0 \mid b_1 \mid b_2\\ \varphi &:= p \mid \bot \mid \neg\varphi \mid [\alpha]\,\varphi\end{aligned}$$

A model for ICPDL is a tuple $\mathcal{M} = (W, R, V)$ where W is a non-empty set of states, $R : \Pi_0 \cup \{b_0, b_1, b_2\} \longrightarrow \mathcal{P}\left(W^2\right)$ and $V : \Phi_0 \longrightarrow \mathcal{P}(W)$. See [15] for the definition of the forcing relation $\vDash_{\text{IC}}$. The translation function τ from PRSPDL to ICPDL is defined by inductively replacing all subprograms of the form $\alpha \parallel \beta$ by $b_0\,;\left(\left(b_1\,;\tau(\alpha)\,;b_1^-\right) \cap \left(b_2\,;\tau(\beta)\,;b_2^-\right)\right)\,;b_0^-$, all subprograms of the form s_i for $i \in \{1,2\}$ by $b_0^-\,;b_i^-$ and all subprograms of the form r_i for $i \in \{1,2\}$ by $b_0\,;b_i$. Given a PRSPDL's formula φ, let $\{a_1, \ldots\ldots, a_n\}$ be the set of atomic programs occurring in φ. The ICPDL program $\pi(\varphi)$ is defined by $\pi(\varphi) = \left(a_1 \cup \ldots\ldots \cup a_n \cup b_0 \cup b_0^- \cup b_1 \cup b_1^- \cup b_2 \cup b_2^-\right)^*$.

Proposition 1. *A PRSPDL formulas $\varphi \in \Phi$ is satisfiable if and only if the ICPDL formula $\tau(\varphi) \wedge [\pi(\varphi)]\left(\langle b_1\rangle\top \leftrightarrow \langle b_2\rangle\top\right)$ is satisfiable.*

As a corollary, by [15]:

Corollary 1. *The satisfiability problem of PRSPDL with respect to the class of all frames is in 2EXPTIME.*

4 Fischer-Ladner Closure over $\mathcal{L}_{;?*\parallel}$

In this section, we consider the fragment $\mathcal{L}_{;?*\parallel}$ of PRSPDL. We define the sets $\Pi_{;?*\parallel} = \Pi \cap \mathcal{L}_{;?*\parallel}$ and $\Phi_{;?*\parallel} = \Phi \cap \mathcal{L}_{;?*\parallel}$ of programs and formulas of $\mathcal{L}_{;?*\parallel}$. In traditional tableaux methods for PDL-like logics, the formulas appearing in a tableau all belong to a Fischer-Ladner closure [11]. In the case of PRSPDL, due to the parallel composition construct, the Fischer-Ladner closure must be defined in an extension of the language.

4.1 Placeholders and Marking Functions

In order to decompose formulas of the form $[\alpha \parallel \beta]\,\varphi$ into subformulas, parallel compositions are distinguished using indices and new atomic formulas called placeholders are added. The sets Π_{PH}, Φ_{pure} and Φ_{PH} of *annotated programs*, *pure formulas* and *annotated formulas* respectively, are defined by parallel induction as follows:

$$\begin{aligned}\alpha, \beta &:= a \mid (\alpha\,;\beta) \mid \varphi? \mid \alpha^* \mid (\alpha \parallel_i \beta)\\ \varphi &:= p \mid \bot \mid \neg\varphi \mid [\alpha]\varphi\\ \psi &:= \varphi \mid (i,j) \mid \neg\psi \mid [\alpha]\psi\end{aligned}$$

where i ranges over $\mathbb{N}$ and j over $\{1,2\}$. Moreover, for any $i \in \mathbb{N}$, there must be at most one occurrence of $\|_i$ in any pure formula. The integers below the parallel composition symbols are called *indices*. Formulas of the form (i,j) are called *placeholders*.

To interpret annotated formulas, if placeholders were simply considered as new propositional variables, it would be impossible to ensure that whenever $w \lhd (x,y)$ and $\mathcal{M}, x \vDash [\alpha \|_i \beta]\,\varphi$ then $\mathcal{M}, x \vDash [\alpha]\,(i,1)$ and $\mathcal{M}, y \vDash [\beta]\,(i,2)$. Therefore we interpret placeholders using *marking functions* which assign subsets of W to placeholders. The set of all such functions is denoted by B_W. The *empty marking function* $m_W^{\varnothing} \in B_W$ binds the empty set to all placeholders. The 4-ary forcing relation $\vDash_{\mathrm{F}}$ is defined on all models $\mathcal{M} = (W, R, \lhd, V)$, all $w \in W$, all $m \in B_W$ and all $\varphi \in \Phi_{PH}$ by parallel induction along with the extension of R to all annotated programs, in a similar way than for PRSPDL except:

$$
\begin{array}{ll}
\mathcal{M}, x, m \vDash_{\mathrm{F}} (i,j) & \text{iff } x \in m(i,j) \\
xR\,(\varphi?)y & \text{iff } x = y \text{ and } \mathcal{M}, x, m_W^{\varnothing} \vDash_{\mathrm{F}} \varphi \\
xR\,(\alpha \|_i \beta)y & \text{iff } \exists w_1, w_2, w_3, w_4 \in W, \\
 & \qquad x \lhd (w_1, w_2)\,, w_1 R\,(\alpha) w_3, w_2 R\,(\beta) w_4 \text{ and } y \lhd (w_3, w_4)
\end{array}
$$

There exists a forgetful epimorphism $\overline{\cdot} : \Phi_{\mathrm{pure}} \longrightarrow \Phi_{;?*\|}$ associating to each pure formula φ the formula $\overline{\varphi}$ obtained by removing all indices in φ. Thanks to the following lemma, which can be easily proved by induction on $|\varphi|$, we will consider satisfiability of pure formulas instead of satisfiability of $\mathcal{L}_{;?*\|}$ formulas.

Lemma 1. *For all $\varphi_0 \in \Phi_{pure}$, $\mathcal{M}, w, m_W^{\varnothing} \vDash_F \varphi_0$ iff $\mathcal{M}, w \vDash \overline{\varphi_0}$. Moreover, for all $m \in B_W$, $\mathcal{M}, w, m_W^{\varnothing} \vDash_F \varphi_0$ iff $\mathcal{M}, w, m \vDash_F \varphi_0$*

4.2 Fischer-Ladner Closure

Following [11], given an annotated formulas φ over Π_0 and Φ_0, we will define the *closure* $\mathrm{FL}(\varphi_0)$ *of* φ by applying the rules in Fig. 1.

Lemma 2. *The cardinality of* $\mathrm{FL}(\varphi_0)$ *is linear in* $|\varphi|$.

We will be mainly interested in closures of pure formulas. We define the set $\mathrm{SP}(\varphi_0) = \{\alpha \mid \exists \varphi, [\alpha]\,\varphi \in \mathrm{FL}(\varphi_0)\}$ of subprograms of any pure formula φ_0.

$$
\frac{[a]\,\varphi}{\varphi}
\qquad
\frac{\varphi}{\neg\varphi}
$$

$$
\frac{[\varphi?]\,\psi}{\varphi \quad \psi}
\qquad
\frac{[\alpha\,;\beta]\,\varphi}{[\alpha]\,[\beta]\,\varphi}
$$

$$
\frac{[\alpha^*]\,\varphi}{[\alpha]\,[\alpha^*]\,\varphi \quad \varphi}
\qquad
\frac{[\alpha \|_i \beta]\,\varphi}{[\alpha](i,1) \quad [\beta](i,2) \quad \varphi}
$$

Fig. 1. Fischer-Ladner closure calculus

Lemma 3. *For any pure formula φ_0 and any $i \in \mathbb{N}$, there is at most one formula of the form $[\alpha \parallel_i \beta]\,\varphi$ in* $\mathrm{FL}(\varphi_0)$.

The function G_{φ_0} is defined such that for all $i \in \mathbb{N}$, if there exists $\alpha, \beta \in \Pi_{PH}$ verifying $[\alpha \parallel_i \beta]\,\varphi \in \mathrm{FL}(\varphi_0)$ then $G_{\varphi_0}(i) = \varphi$, otherwise $G_{\varphi_0}(i) = \top$. When the index φ_0 is obvious from the context, we write G instead of G_{φ_0}.

5 Tableaux Method for $\mathcal{L}_{;?*\parallel}$ over $\mathcal{C}_{\lhd\text{-det}}$

In this section we introduce a tableaux method for pure formulas interpreted over $\mathcal{C}_{\lhd\text{-det}}$. To deal with the merging of states at the end of parallel compositions, we borrow some ideas from [16]. Firstly, non-atomic programs are allowed as label of edges in the built structure. Secondly, placeholders are used in order to ensure that formulas of the form $[\alpha \parallel_i \beta]\,\varphi$ are propagated.

5.1 Rules of the Tableaux Method

Given a set W of states, a judgment about W is either:

- a judgment $x\colon \varphi$ stating that x must satisfy φ;
- a judgment $(x, y)\colon \alpha$ stating that y can be reached from x by α;
- a judgment $(x, y, z) : \Delta$ with $\Delta \in \{F, B\}$, stating that x can be decomposed forwardly (if $\Delta = F$) or backwardly (if $\Delta = B$) into y and z.

A judgment j *involves* a state x iff x appears on the left of j. A structure is a tuple $\mathcal{S} = (W, J, K)$ where W is a set of states, J a set of judgments about W and $K \subseteq J$ a subset of inactive judgments. A tableau $\mathcal{T}$ for a pure formula φ_0 is an ordered, possibly infinite tree whose nodes are labeled with structures, the root being labeled with the initial structure $(\{w_0\}, \{w_0\colon \varphi_0\}, \emptyset)$ for some w_0. Successor nodes are constructed in accordance with the rules from Figs. 2, 3, 4, 5 and 6. The rules have the general form

$$\frac{X_0}{X_1 \mid \ldots\ldots \mid X_\ell} C$$

where X_0 is the set of premises, $(X_k)_{k \in 1 \ldots . \ell}$ are the sets of conclusions, C is the set of side conditions and $\ell > 0$. The rules $(\Box)$, $(\Box \| 1F)$, $(\Box \| 1B)$, $(\Box \| 0\top)$ and $(\Box \| 0\bot)$ are called *universal*. States denoted by n, n_1, n_2, n_3 and n_4 in the conclusions must be fresh. A rule instantiation is applicable to a node η_0 labeled with $\mathcal{S}_0 = (W_0, J_0, K_0)$ if all the following conditions are met:

- the instantiation $\overline{X_0}$ of the set of premises is a subset of $J_0 \setminus K_0$,
- all side conditions' instantiations are satisfied,
- if the rule is universal then for all $k \in 1 \ldots . \ell$, there is a judgment j_k in X'_ks instantiation such that $\mathrm{j}_k \notin J_0$.

$$\Box \ \frac{x\colon [a]\varphi \qquad (x,y)\colon a}{y\colon \varphi}$$

$$\Diamond 1 \ \frac{x\colon \langle\alpha\rangle\varphi}{(x,n)\colon \alpha \qquad n\colon \varphi} \ \mathrm{size}(\alpha) = 1 \qquad \Diamond 0 \ \frac{x\colon \langle\alpha\rangle\varphi}{(x,x)\colon \alpha \qquad x\colon \varphi} \ \mathrm{size}(\alpha) = 0$$

$$\Diamond * \ \frac{x\colon \langle\alpha\rangle\varphi}{(x,n)\colon \alpha \qquad n\colon \varphi \ \mid \ (x,x)\colon \alpha \qquad x\colon \varphi} \ \mathrm{size}(\alpha) = *$$

Fig. 2. Basic rules of $\mathcal{L}_{;?*\|}$'s tableaux calculus

$$\Box ? \ \frac{x\colon [\varphi?]\psi}{x\colon \neg\varphi \ \mid \ x\colon \psi} \qquad \Diamond ? \ \frac{(x,x)\colon \varphi?}{x\colon \varphi}$$

Fig. 3. Test rules of $\mathcal{L}_{;?*\|}$'s tableaux calculus

$$\Box; \ \frac{x\colon [\alpha\ ;\ \beta]\varphi}{x\colon [\alpha][\beta]\varphi} \qquad \Diamond;00 \ \frac{(x,x)\colon \alpha\ ;\ \beta}{(x,x)\colon \alpha \qquad (x,x)\colon \beta}$$

$$\Diamond;0\mathbf{.} \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{0\mathbf{.}}} \ \mathrm{size}(\alpha) = 0, x \neq y \qquad \Diamond;\mathbf{.}0 \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{\mathbf{.}0}} \ \mathrm{size}(\beta) = 0, x \neq y$$

$$\Diamond;11 \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{11}} \ \mathrm{size}(\alpha) = \mathrm{size}(\beta) = 1$$

$$\Diamond;1* \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{\mathbf{.}0} \ \mid \ Y_{11}} \ \mathrm{size}(\alpha) = 1, \mathrm{size}(\beta) = *$$

$$\Diamond;*1 \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{0\mathbf{.}} \ \mid \ Y_{11}} \ \mathrm{size}(\alpha) = *, \mathrm{size}(\beta) = 1$$

$$\Diamond;** \ \frac{(x,y)\colon \alpha\ ;\ \beta}{Y_{0\mathbf{.}} \ \mid \ Y_{\mathbf{.}0} \ \mid \ Y_{11}} \ \mathrm{size}(\alpha) = \mathrm{size}(\beta) = *, x \neq y$$

$$Y_{0\mathbf{.}} = \{(x,x)\colon \alpha, \ (x,y)\colon \beta\}$$
$$Y_{\mathbf{.}0} = \{(x,y)\colon \alpha, \ (y,y)\colon \beta\}$$
$$Y_{11} = \{(x,n)\colon \alpha, \ (n,y)\colon \beta\}$$

Fig. 4. Sequence rules of $\mathcal{L}_{;?*\|}$'s tableaux calculus

$$\Box * \ \frac{x\colon [\alpha^*]\varphi}{x\colon \varphi \qquad x\colon [\alpha][\alpha^*]\varphi}$$

$$\Diamond *\neq \ \frac{(x,y)\colon \alpha^*}{(x,y)\colon \alpha \ \mid \ (x,n)\colon \alpha \qquad (n,y)\colon \alpha^*} \ x \neq y$$

Fig. 5. Iteration rules of $\mathcal{L}_{;?*\|}$'s tableaux calculus

$$\Box\|1F\ \ \frac{x\colon [\alpha\ \|_i\ \beta]\varphi \qquad (x,y,z)\colon F}{y\colon [\alpha](i,1) \qquad z\colon [\beta](i,2)}\ \ \mathrm{size}\,(\alpha\ \|_i\ \beta) \neq 0$$

$$\Box\|1B\ \ \frac{y\colon (i,1) \qquad z\colon (i,2) \qquad (x,y,z)\colon B}{x\colon G(i)}$$

$$\Box\|0\top\ \ \frac{x\colon [\alpha\ \|_i\ \beta]\varphi}{x\colon \varphi \ \mid\ x\colon [\mathrm{desiter}\,(\alpha\ \|_i\ \beta)]\,\bot}\ \ \mathrm{size}\,(\alpha\ \|_i\ \beta) \neq 1$$

$$\Box\|0\bot\ \ \frac{x\colon [\alpha\ \|_i\ \beta]\bot \qquad (x,y,z)\colon \Delta}{y\colon [\alpha]\bot \ \mid\ z\colon [\beta]\bot}\ \ \mathrm{size}\,(\alpha\ \|_i\ \beta) = 0$$

$$\Diamond\|00\ \ \frac{(x,x)\colon \alpha\ \|_i\ \beta}{(x,n_1,n_2)\colon F \qquad (n_1,n_1)\colon \alpha \qquad (n_2,n_2)\colon \beta}$$

$$\Diamond\|0\bullet\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{0\bullet}}\ \ \mathrm{size}(\alpha) = 0, x \neq y \qquad \Diamond\|\bullet 0\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{\bullet 0}}\ \ \mathrm{size}(\beta) = 0, x \neq y$$

$$\Diamond\|11\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{11}}\ \ \mathrm{size}(\alpha) = 1, \mathrm{size}(\beta) = 1$$

$$\Diamond\|1*\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{\bullet 0} \ \mid\ Z_{11}}\ \ \mathrm{size}(\alpha) = 1, \mathrm{size}(\beta) = *$$

$$\Diamond\|*1\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{0\bullet} \ \mid\ Z_{11}}\ \ \mathrm{size}(\alpha) = *, \mathrm{size}(\beta) = 1$$

$$\Diamond\|**\ \ \frac{(x,y)\colon \alpha\ \|_i\ \beta}{Z_{0\bullet} \ \mid\ Z_{\bullet 0} \ \mid\ Z_{11}}\ \ \mathrm{size}(\alpha) = \mathrm{size}(\beta) = *, x \neq y$$

$$Z_{0\bullet} = \{(x,n_1,n_2)\colon F,\ (y,n_1,n_4)\colon B,\ (n_1,n_1)\colon \alpha,\ (n_2,n_4)\colon \beta\}$$
$$Z_{\bullet 0} = \{(x,n_1,n_2)\colon F,\ (y,n_3,n_2)\colon B,\ (n_1,n_3)\colon \alpha,\ (n_2,n_2)\colon \beta\}$$
$$Z_{11} = \{(x,n_1,n_2)\colon F,\ (y,n_3,n_4)\colon B,\ (n_1,n_3)\colon \alpha,\ (n_2,n_4)\colon \beta\}$$

Fig. 6. Parallel composition rules of $\mathcal{L}_{;?*\|}$'s tableaux calculus

When applying a rule instantiation, the ℓ child nodes $\eta_1, \ldots\ldots, \eta_\ell$ of η_0, labeled with $\mathcal{S}_1, \ldots\ldots, \mathcal{S}_\ell$, are created such that $\mathcal{S}_k = (W_0 \cup F_k, J_0 \cup \overline{X_k}, K_0 \cup Q)$ where F_k is the set of fresh states corresponding to n, n_1, n_2, n_3 or n_4 in X_k, $\overline{X_k}$ is the instantiation of X_k and $Q = \overline{X_0}$ except for the universal rules for which $Q = \emptyset$.

The size function in the side conditions is defined by:

$$\mathrm{size}(\varphi?) = 0 \qquad \mathrm{size}(\alpha\,;\beta) = \mathrm{size}(\alpha \parallel \beta) = \begin{cases} 0 & \text{if} \mathrm{size}(\alpha) = \mathrm{size}(\beta) = 0 \\ 1 & \text{if} \mathrm{size}(\alpha) = 1 \text{ or } \mathrm{size}(\beta) = 1 \\ * & \text{otherwise} \end{cases}$$
$$\mathrm{size}(a) = 1$$

$$\mathrm{size}(\alpha*) = \begin{cases} 0 & \text{if} \mathrm{size}(\alpha) = 0 \\ * & \text{otherwise} \end{cases}$$

The rules ensure that for any judgment $(x, y)\colon \alpha \in J$, if $\text{size}(\alpha) = 0$ then $x = y$ and if $\text{size}(\alpha) = 1$ then $x \neq y$. When $\text{size}(\alpha) = *$, both cases must be considered. For instance rule $(\Diamond *)$ may be seen as the disjunction of the rules $(\Diamond 1)$ and $(\Diamond 0)$. When a program α of size $*$ is considered as having size 0, it is implicitly replaced by $\text{desiter}(\alpha)$. The function $\text{desiter} : \Pi_{PH} \longrightarrow \Pi_{PH}$ substitutes each occurrence of subprograms of the form α^* with $\top$? The replacement is made explicit in the right-hand side conclusion of rule $(\Box \| 0\top)$ in order to enable the application of rule $(\Box \| 0\bot)$ afterward. Obviously, if $\text{size}(\alpha) \neq 1$ then $\text{size}\,(\text{desiter}\,(\alpha)) = 0$.

For judgments of the form $(x, y, z)\colon \Delta$, we distinguish forward $(\Delta = F)$ and backward $(\Delta = B)$ decompositions. The rules ensure that if $(x, y, z)\colon \Delta \in J$, $(x', y', z')\colon \Delta' \in J$ and y' and z' are reachable from y and z respectively, then either $(y, z) = (y', z')$ or $\Delta = F$ and $\Delta' = B$. This property is used in rules $(\Box \| 1F)$ and $(\Box \| 1B)$ to ensure that no new judgments about a state is added after all successors of that state have been added (see Lemma 7 on page 12). Rules $(\Box \| 1F)$ and $(\Box \| 1B)$ ensure that if $x\colon [\alpha \parallel_i \beta]\,\varphi \in J$ then for any state $y \neq x$ reachable from x by $\alpha \parallel_i \beta$, $y\colon \varphi \in J$. They make use of placeholders and function G from Sect. 4. Similarly, rules $(\Box \| 0\top)$ and $(\Box \| 0\bot)$ ensure that if $x\colon [\alpha \parallel_i \beta]\,\varphi \in J$ then either $x\colon \varphi \in J$ or x is not reachable from x by $\alpha \parallel_i \beta$. When $\text{size}(\alpha \parallel_i \beta) = *$, since the rules $(\Box \| 1F)$ and $(\Box \| 0\top)$ are both universal, they could be both applied on the same judgment $x\colon [\alpha \parallel_i \beta]\,\varphi$.

In a tableau, a maximal path from the root is called a *branch*. For any branch $\mathcal{B}$, we write $W_\mathcal{B}$ (resp. $J_\mathcal{B}$) for the union of the W (resp. J) such that there exists a node in $\mathcal{B}$ labeled with (W, J, K) for some J (resp. W) and K. A structure $\mathcal{S} = (W, J, K)$ is *inconsistent* if there exists $x \in W$ such that $x\colon \bot \in J$ or both $x\colon \varphi \in J$ and $x\colon \neg\varphi \in J$ for some $\varphi \in \Phi_{PH}$. A branch is *open* if its nodes are all labeled with a consistent structure. A branch $\mathcal{B}$ is *saturated* iff for any node $\eta \in \mathcal{B}$ labeled with $\mathcal{S} = (W, J, K)$ and any rule's instantiation π applicable on $\mathcal{S}$, there exists a node η' in $\mathcal{B}$ labeled with $\mathcal{S}' = (W', J', K')$ and such that one of π's conclusions sets is a subset of J'. A branch $\mathcal{B}$ is *demand-satisfied* iff for any node $\eta \in \mathcal{B}$ labeled with $\mathcal{S} = (W, J, K)$ and any judgment in J of the form $(x, y)\colon \alpha^*$ there is a node $\eta' \in \mathcal{B}$ labeled with $\mathcal{S}' = (W', J', K')$ and a list $x_0, \ldots\ldots, x_m \in W'$ such that $x_0 = x$, $x_m = y$ and for all $i < m$, $(x_i, x_{i+1})\colon \alpha \in J'$. A tableau is *satisfying* if it has an open saturated demand-satisfied branch. We will prove that for any pure formula φ_0, there exists a satisfying tableau for φ_0 if and only if φ_0 is satisfiable.

5.2 Soundness

We prove the soundness of the tableaux method by interpreting branches into a satisfying model. The use of placeholders necessitates the selection of marking functions to interpret judgments. We introduce the notion of *twines* to select those marking functions. Intuitively, a twine corresponds to an equivalence class of states by the transitive and symmetric closure of the relation obtained as the union of the accessibility relation (by any program) and the relation linking two states iff they are mergeable (by $\lhd$).

Formally, Let $\mathcal{B}$ be a branch from a tableau for φ_0. The set Θ of twines of $\mathcal{B}$ is defined as $\Theta = W_{\mathcal{B}}^2 \cup \{\theta_0\}$ with θ_0 not being a member of $W_{\mathcal{B}}^2$. A twine function t assigns a twine to each state in $W_{\mathcal{B}}$. The function t is constructed from the root of $\mathcal{B}$ as follows:

1. If x is the unique state in the label of the root, then $t(x) = \theta_0$.
2. If x has been added by an application of a rule which did not add a judgment of the form $(z, w_1, w_2)\colon F$ (rules $(\Diamond 1)$, $(\Diamond *)$, $(\Diamond; 11)$, $(\Diamond; 1*)$, $(\Diamond; *1)$, $(\Diamond; **)$ and $(\Diamond * \neq)$) then $t(x) = t(y)$, y being any state involved in the premises of the rule instantiation. A careful analysis of the rules shows that the choice of y does not matter, because whenever $(y_1, y_2)\colon \alpha \in J_{\mathcal{B}}$ then $t(y_1) = t(y_2)$.
3. If x has been added by an application of a rule which did add a judgment of the form $(z, w_1, w_2)\colon F$ (rules $(\Diamond \| 00)$, $(\Diamond \| 0\bullet)$, $(\Diamond \| \bullet 0)$, $(\Diamond \| 11)$, $(\Diamond \| 1*)$, $(\Diamond \| *1)$ and $(\Diamond \| **)$), then $t(x) = (w_1, w_2)$.

The set Θ^+ of *active twines of* $\mathcal{B}$ is the image of the twine function. For any twine $\theta \in \Theta^+ \setminus \{\theta_0\}$ there exists a unique tuple $(x, w_1, w_2) \in W_{\mathcal{B}}^3$ such that $\theta = (w_1, w_2)$ and $(x, w_1, w_2)\colon F \in J_{\mathcal{B}}$. In that case, we write ${}^{\lhd}\theta$ for $t(x)$.

Given a branch $\mathcal{B}$ with twine function t, a structure $\mathcal{S} = (W, J, K)$ labeling a node in $\mathcal{B}$ and a model $\mathcal{M}' = (W', R', \lhd', V')$, a pair (f, g) is an *interpretation of* $\mathcal{S}$ *into* $\mathcal{M}'$ *with respect to* $\mathcal{B}$ if f is a function from W to W' and g a function from Θ^+ to $B_{W'}$ such that for all $x, y, z \in W$, $x', y', z' \in W'$, $\varphi \in \Phi_{PH}$, $\alpha \in \Pi_{PH}$, $\Delta \in \{F, B\}$, $\theta \in \Theta^+ \setminus \{\theta_0\}$ and $i \in \mathbb{N}$:

$$x\colon \varphi \in J \Rightarrow \mathcal{M}', f(x), g\left(t(x)\right) \vDash_{\mathrm{F}} \varphi \tag{1}$$

$$(x, y)\colon \alpha \in J \Rightarrow f(x) R'\left(\alpha\right) f(y) \tag{2}$$

$$(x, x)\colon \alpha \in J \Rightarrow f(x) R'\left(\mathrm{desiter}(\alpha)\right) f(x) \tag{3}$$

$$(x, y)\colon \alpha \in J, x \neq y \text{ and } \mathrm{size}(\alpha) = * \Rightarrow (f(x), f(y)) \notin R'(\mathrm{desiter}(\alpha)) \tag{4}$$

$$(x, y, z)\colon \Delta \in J \Rightarrow f(x) \lhd' \left(f(y), f(z)\right) \tag{5}$$

$$x' \lhd' (y', z') \wedge y' \in g(\theta)(i, 1) \wedge z' \in g(\theta)(i, 2) \Rightarrow \mathcal{M}', x', g({}^{\lhd}\theta) \vDash_{\mathrm{F}} G(i) \tag{6}$$

If there is such an interpretation, $\mathcal{S}$ is said to be interpretable in $\mathcal{M}'$ with respect to $\mathcal{B}$. If the label of each node in $\mathcal{B}$ is interpretable in $\mathcal{M}'$ with respect to $\mathcal{B}$, then $\mathcal{B}$ is interpretable in $\mathcal{M}'$.

Obviously, interpretable branches are open and the rules preserve the interpretability. By ordering the applicable rule instantiations in a queue, a strategy for rule applications can be easily defined such that a saturated tableau is obtained for all pure formulas. Then, to prove Proposition 2 below, it suffices to prove the following lemma, which is done by selecting the interpretable branch where the leftmost child of nodes on which rule $(\Diamond * \neq)$ is applied is chosen whenever possible.

Lemma 4. *If φ_0 is satisfiable and $\mathcal{T}$ is a tableau for φ_0 in which all open branches are saturated, then $\mathcal{T}$ has an open saturated demand-satisfied branch.*

Proposition 2. *If $\varphi_0 \in \Phi_{pure}$ is satisfiable, there exists a tableaux for φ_0 with an open saturated demand-satisfied branch.*

5.3 Completeness

We now consider a satisfying tableau $\mathcal{T}$ for φ_0. We will construct a model satisfying φ_0. Since $\mathcal{T}$ is satisfying, it has an open saturated demand-satisfied branch $\mathcal{B}$. The model $\mathcal{M} = (W, R, \lhd, V)$ and the marking function m are defined as follows:

$$\begin{aligned} W &= W_{\mathcal{B}} \\ R(a) &= \left\{(x,y) \in W^2 \mid (x,y) : a \in J_{\mathcal{B}}\right\},\ \forall a \in \Pi_0 \\ \lhd &= \left\{(x,y,z) \in W^3 \mid \exists \Delta \in \{F,B\},\ (x,y,z) : \Delta \in J_{\mathcal{B}}\right\} \\ V(p) &= \left\{x \in W \mid x : p \in J_{\mathcal{B}}\right\},\ \forall p \in \Phi_0 \\ m(i,j) &= \left\{x \in W \mid x : (i,j) \in J_{\mathcal{B}}\right\},\ \forall (i,j) \in \mathbb{N} \times \{1,2\} \end{aligned}$$

By construction of $\mathcal{T}$, $\mathcal{M}$ is $\lhd$-deterministic. By induction on $|\varphi|$ and $|\alpha|$, the following truth lemma can be proved.

Lemma 5. *For all $x, y \in W$, $\varphi \in \Phi_{PH}$ and $\alpha \in \Pi_{PH}$,*

$$x \colon \varphi \in J_{\mathcal{B}} \Rightarrow \mathcal{M}, x, m \vDash_F \varphi \tag{7}$$

$$(x,y) \colon \alpha \in J_{\mathcal{B}} \Rightarrow xR(\alpha)y \tag{8}$$

The proof of Lemma 5 necessitates various properties of function R:

- If $xR(\alpha \parallel_i \beta)y$ and $x \neq y$, then there exists $w_1, w_2, w_3, w_4 \in W$ such that $(x, w_1, w_2) \colon F \in J_{\mathcal{B}}$, $(y, w_3, w_4) \colon B \in J_{\mathcal{B}}$, $w_1 R(\alpha) w_3$ and $w_2 R(\beta) w_4$.
- For all $x \in W$, $\alpha, \beta \in \Pi_{PH}$ and $i \in \mathbb{N}$, if $xR(\alpha \parallel_i \beta)x$ then there exists $w_1, w_2 \in W$ such that $x \lhd (w_1, w_2)$, $w_1 R(\alpha) w_1$ and $w_2 R(\beta) w_2$.
- For all $x \in W$ and $\alpha \in \Pi_{PH}$, if $xR(\alpha)x$ then $\mathrm{size}(\alpha) \neq 1$.
- For all $x \in W$ and $\alpha \in \Pi_{PH}$, if $xR(\alpha)x$ then $xR(\mathrm{desiter}(\alpha))x$.

Our completeness result immediately follows from Lemma 5.

Proposition 3. *For any pure formula φ_0, if there exists a satisfying tableau for φ_0, then φ_0 is satisfiable.*

6 Optimal Decision Procedure for $\mathcal{L}_{;?\parallel}$ Over $\mathcal{C}_{\lhd\text{-det}}$

In this section we establish the complexity of the satisfiability problem of $\mathcal{L}_{;?\parallel}$ over $\mathcal{C}_{\lhd\text{-det}}$. The fragment $\mathcal{L}_{;?\parallel}$ is the iteration-free fragment of $\mathcal{L}_{;?*\parallel}$. Therefore, we reuse the constructions from the previous sections. We write $\Pi_{0,\mathrm{PH}}$ for the set of iteration-free annotated programs, $\Phi_{0,\mathrm{PH}}$ for the set of iteration-free annotated formulas and $\Phi_{0,\mathrm{pure}}$ for the set of iteration-free pure formulas. It can be easily checked that for all $\varphi_0 \in \Phi_{0,\mathrm{pure}}$, $\mathrm{FL}(\varphi_0) \subseteq \Phi_{0,\mathrm{PH}}$.

6.1 Semantic Tableaux Method

The rules of $\mathcal{L}_{;?\|}$'s tableaux calculus are the rules $(\Box)$, $(\Diamond 1)$, $(\Diamond 0)$, $(\Box?)$, $(\Diamond?)$, $(\Box;)$, $(\Diamond;00)$, $(\Diamond;11)$, $(\Diamond;0\bullet)$, $(\Diamond;\bullet 0)$, $(\Box\|1F)$, $(\Box\|1B)$, $(\Box\|0\top)$, $(\Box\|0\bot)$, $(\Diamond\|00)$, $(\Diamond\|0\bullet)$, $(\Diamond\|\bullet 0)$ and $(\Diamond\|11)$ from Figs. 2, 3, 4 and 6 along with the rule $(\Diamond;D)$ from Fig. 7. The rule $(\Diamond;D)$ is needed to ensure local saturation in the strategy of Sect. 6.2. Using the same techniques as in Sect. 5, we can prove that for any pure formula $\varphi_0 \in \Phi_{0,\text{pure}}$, there exists a tableau for φ_0 with an open saturated branch if and only if φ_0 is satisfiable.

$$\Diamond;D \quad \frac{x\colon \langle \alpha\,;\,\beta\rangle\varphi}{x\colon \langle\alpha\rangle\langle\beta\rangle\varphi}$$

Fig. 7. Additional rule for $\mathcal{L}_{;?\|}$'s tableaux calculus

Let $\mathrm{FL}^+(\varphi_0) = \mathrm{FL}(\varphi_0) \cup \{[\alpha]\bot \mid \alpha \in \mathrm{SP}(\varphi_0)\}$. The following lemma can be proved by induction on the length of the path from the root of the tableau to the node η.

Lemma 6. *Given any structure $\mathcal{S} = (W, J, K)$ labeling a node in a tableau for φ_0, for any judgment $x\colon \varphi \in J$, $\varphi \in \mathrm{FL}(\varphi_0)[\varphi_0][^+]$ and for any judgment $(x, y)\colon \alpha \in J$, $\alpha \in SP(\varphi_0)$.*

6.2 Optimal Decision Procedure

We will prove that the nondeterministic procedure DECISION defined on the next page solves the satisfiability problem of $\mathcal{L}_{;?\|}$ over $\mathcal{C}_{\lhd\text{-det}}$ in polynomial space. Called with a pure formula φ_0, this procedure constructs a branch of a tableau for φ_0 and returns SAT if this branch is open and saturated. In order to reduce memory usage, the procedure ensures that after any application of an instantiation π of the rules $(\Diamond 1)$ and $(\Diamond\|00)$, no new judgments can be added which involve only the states in π's premises, see Lemma 7 below. A *local rule* is a rule which is neither $(\Diamond 1)$ nor $(\Diamond\|00)$. A structure is *locally saturated* iff no local rule instantiations can be applied to it. A rule's instantiation π is *appropriate to $\mathcal{S}$ and x* iff π is applicable to $\mathcal{S}$ and either π is an instantiation of a local rule or the instantiations of the premises involve only x.

Lemma 7. *Let $\mathcal{S} = (W, J, K)$ be a locally saturated structure labeling a node η in a branch of a tableau. For any descendent node η' of η with label $\mathcal{S}' = (W', J', K')$ and any judgment $\mathfrak{j} \in J'$ involving only one state $x \in W'$, if $x \in W$ then $\mathfrak{j} \in J$.*

DECISION first creates the structure for the root node (line 1). Then it locally saturates this structure without adding any new state (line 2–3). Finally it calls the recursive EXTEND procedure and check whether it returns the empty structure. The empty structure is used by EXTEND as a marker for a closed branch.

Procedure 1. DECISION

Input: A pure iteration-free formula $\varphi_0 \in \Phi_{0,\mathrm{pure}}$.
Output: SAT or UNKNOWN.
Data: A structure $\mathcal{S} = (W, J, K)$.

1 $\mathcal{S} \leftarrow (\{w_0\}, \{w_0 : \varphi_0\}, \emptyset)$
2 **while** *there is a local rule's instantiation* π *applicable to* $\mathcal{S}$ **do**
3 $\quad \mathcal{S} \leftarrow$ a nondeterministically chosen successor of $\mathcal{S}$ by π
4 $\mathcal{S} \leftarrow \textsc{Extend}(\mathcal{S}, w_0)$
5 **if** $W \neq \emptyset$ **then** return SAT
6 **else** return UNKNOWN

Procedure 2. EXTEND

Input: A locally saturated structure $\mathcal{S} = (W, J, K)$ and a state $x \in W$.
Output: A (possibly empty) structure $\mathcal{S}_f = (W_f, J_f, K_f)$.
Data: A set J_0 of judgments and a structure $\mathcal{S}' = (W', J', K')$.

7 $J_0 \leftarrow \{\mathrm{j} \in J \mid \mathrm{j}$ involves only $x\}$
8 $\mathcal{S}' \leftarrow (W, J_0, K \cap J_0)$
9 **while** *there is a rule's instantiation* π *appropriate to* $\mathcal{S}'$ *and* x **do**
10 $\quad \mathcal{S}' \leftarrow$ a nondeterministically chosen successor of $\mathcal{S}'$ by π
11 **if** $\mathcal{S}'$ *is inconsistent* **then**
12 $\quad \mathcal{S}_f \leftarrow (\emptyset, \emptyset, \emptyset)$
13 **else**
14 $\quad \mathcal{S}_f \leftarrow (W', J \cup J', K \cup K')$
15 $\quad$ **foreach** $y \in W' \setminus W$ **do**
16 $\quad\quad \mathcal{S}_f \leftarrow \textsc{Extend}\,(\mathcal{S}_f, y)$
17 **return** $\mathcal{S}_f$

EXTEND operates in two steps. Firstly, in the *existential loop* (lines 9–10), successors of x are added and the structure is locally saturated. Secondly, in the *universal loop* (lines 15–16), EXTEND is recursively called for each state created by the existential loop. The following properties can be proved:

Lemma 8. *1. At each run of* EXTEND*, the existential loop adds a number of new states bounded by a polynomial in* $|\varphi_0|$*.*
2. During a call to DECISION(φ_0)*, the recursion depth of the calls to* EXTEND *is bounded by a polynomial in* $|\varphi_0|$*.*
3. DECISION(φ_0) *returns SAT only if a saturated branch for* φ_0 *has been constructed.*

By Lemmas 2 and 6, the number of judgments added by the loop at lines 2–3 is polynomial in $|\varphi_0|$. Hence this loop terminates. By Lemma 8, the number of new states added by the existential loop (lines 9–10) is polynomial in $|\varphi_0|$. Therefore, by Lemmas 2 and 6, the cardinality of J' is always bounded by a polynomial in $|\varphi_0|$. Hence the existential loop terminates. By Lemmas 8 and

Köning's lemma, the execution tree of Extend is finite, hence the whole procedure terminates. Moreover, each call to Extend$(\mathcal{S}, x)$ needs to keep track of the judgments involving only states in $\{x\} \cup (W' \setminus W)$ and the number of this judgments is polynomial in $|\varphi_0|$. Finally, by Lemma 8, only a polynomial number of such configurations have to be stored. Consequently,

Proposition 4. *All executions of* Decision(φ_0) *terminate and* Decision *can be implemented using polynomial space.*

Given a pure formula $\varphi_0 \in \Phi_{0,\text{pure}}$, the set of executions of Decision(φ_0) corresponds to a collection Γ of tableaux for φ_0 where each execution corresponds to a branch of a tree. If φ_0 is satisfiable, by soundness of the tableaux method, there is an open branch in each tree of Γ. Since Decision returns UNKOWN only when the corresponding branch is close, there is an execution of Decision(φ_0) returning SAT. Conversely, by Lemma 8, if an execution of Decision(φ_0) returns SAT, the corresponding branch $\mathcal{B}$ is saturated. Since $\mathcal{B}$ is open too, by completeness of the tableaux method, φ_0 is satisfiable. As a result:

Proposition 5. *The nondeterministic procedure* Decision *is a decision procedure for the satisfiability problem of* $\mathcal{L}_{;?\|}$ *with respect to* $\mathcal{C}_{\lhd\text{-}det}$.

By Propositions 4 and 5 and Savitch's Theorem, the satisfiability problem of the fragment $\mathcal{L}_{;?\|}$ with respect to $\mathcal{C}_{\lhd\text{-det}}$ is in PSPACE. PSPACE-hardness is given by the obvious embedding of the modal logic K. Hence:

Proposition 6. *The satisfiability problem of the fragment* $\mathcal{L}_{;?\|}$ *over the class* $\mathcal{C}_{\lhd\text{-}det}$ *is PSPACE-complete.*

7 Conclusion and Future Works

We have given a complexity upper bound for the satisfiability of PRSPDL over the class of all frames, a sound and complete tableaux method for the fragment of PRSPDL without special programs interpreted over $\lhd$-deterministic frames and an optimal decision procedure for an iteration-free fragment of PRSPDL over $\lhd$-deterministic frames. Both our complexity results answer questions left open in [2–4]. Moreover, we proved the addition of $\lhd$-deterministic parallel composition to PDL without choice and iteration does not increase its complexity. We leave for future works to prove this interesting property for the full language.

Because of the characteristics PRSPDL shares with IPDL (for which there is no tableaux method to date), our tableaux method has some peculiarities borrowed from [16] (mainly compound programs as edge's labels) which are difficult to combine with the iteration construct. Hence, because of the rules $(\Box*)$ and $(\Diamond* \neq)$, our tableaux method for $\mathcal{L}_{;?*\|}$ over $\mathcal{C}_{\lhd\text{-det}}$ does not terminate. But we believe this tableaux method can be modified to become a decision procedure. For instance, by adding dynamic blocking as in [13], it might be possible to represent infinite saturated tableaux by finite unsaturated ones. A finite model

property would be helpful. The notion of twines and the Fischer-Ladner closure introduced in this paper could be useful to prove this property.

In PDL, nondeterministic choice adds technical difficulties without changing neither the expressive power nor the complexity of the logic. It has been omitted in this paper for the sake of simplicity, leaving the study of PRSPDL with nondeterministic choice for future works.

References

1. Abrahamson, K.: Modal logic of concurrent nondeterministic programs. In: Kahn, G. (ed.) Semantics of Concurrent Computation. LNCS, vol. 70, pp. 21–33. Springer, Heidelberg (1979)
2. Balbiani, P., Boudou, J.: Iteration-free PDL with storing, recovering and parallel composition: a complete axiomatization, J. Logic Comput. (2015)
3. Balbiani, P., Tinchev, T.: Definability and computability for PRSPDL. In: Advances in Modal Logic, pp. 16–33. College Publications (2014)
4. Benevides, M.R.F., de Freitas, R.P., Viana, J.P.: Propositional dynamic logic with storing, recovering and parallel composition. ENTCS **269**, 95–107 (2011)
5. Brochenin, R., Demri, S., Lozes, É.: Reasoning about sequences of memory states. Ann. Pure Appl. Logic **161**(3), 305–323 (2009)
6. Collinson, M., Pym, D.J.: Algebra and logic for resource-based systems modelling. Math. Struct. Comput. Sci. **19**(5), 959–1027 (2009)
7. Courtault, J.R., Galmiche, D.: A Modal BI logic for dynamic resource properties. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 134–148. Springer, Heidelberg (2013)
8. Danecki, R.: Nondeterministic propositional dynamic logic with intersection is decidable. In: Skowron, A. (ed.) Computation Theory. LNCS, vol. 208, pp. 34–53. Springer, Heidelberg (1984)
9. De Giacomo, G., Massacci, F.: Combining deduction and model checking into tableaux and algorithms for converse-PDL. Inf. Comput. **162**(1–2), 117–137 (2000)
10. Demri, S., Deters, M.: Separation logics and modalities: a survey. J. Appl. Non-Class. Logics **25**(1), 50–99 (2015)
11. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979)
12. Goré, R., Widmann, F.: An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 437–452. Springer, Heidelberg (2009)
13. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. Log. Comput. **9**(3), 385–410 (1999)
14. Kurucz, Á., Németi, I., Sain, I., Simon, A.: Decidable and undecidable logics with a binary modality. J. Logic Lang. Inf. **4**(3), 191–206 (1995)
15. Lutz, C.: PDL with intersection and converse is decidable. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 413–427. Springer, Heidelberg (2005)
16. Massacci, F.: Decision procedures for expressive description logics with intersection, composition, converse of roles and role identity. In: IJCAI, pp. 193–198. Morgan Kaufmann (2001)
17. Peleg, D.: Concurrent dynamic logic. J. ACM **34**(2), 450–479 (1987)

18. Pratt, V.R.: A near-optimal method for reasoning about action. J. Comput. Syst. Sci. **20**(2), 231–254 (1980)
19. Pym, D.J.: The semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series, vol. 26. Kluwer Academic Publishers, Dordrecht (2002)
20. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, pp. 55–74. IEEE Computer Society (2002)