Propositional Dynamic Logic with Storing, Recovering and Parallel Composition

Mario R. F. Benevides\textsuperscript{1,2}  
\textit{Computer Science Department and Systems and Computer Engineering Program}  
\textit{Federal University of Rio de Janeiro}  
\textit{Brazil}

Renata de Freitas\textsuperscript{1,3}  Petrucio Viana\textsuperscript{1,4}

\textit{Institute of Mathematics}  
\textit{Fluminense Federal University}  
\textit{Brazil}

Abstract

This work extends Propositional Dynamic Logic (PDL) with parallel composition operator and four atomic programs which formalize the storing and recovering of elements in data structures. A generalization of Kripke semantics is proposed that instead of using set of possible states it uses structured sets of possible states. This new semantics allows for representing data structures and using the five new operator one is capable of reasoning about the manipulation of these data structures. The use of the new language (PRSPDL) is illustrated with some examples. We present sound and complete set of axiom schemata and inference rules to prove all the valid formulas for a restricted fragment called RSPDL\textsuperscript{o}.

Keywords: Propositional Dynamic Logic, Parallel Composition, Modal Logic for Program Specification, Concurrency

1 Introduction and Motivation

Propositional Dynamic Logic (PDL) \cite{7,14} plays an important role in formal specification and reasoning about sequential programs and systems. It has been used to describe and verify properties as correctness, termination, fairness, liveness and equivalence of programs.

PDL is a multi-modal logic with one modality $\langle \pi \rangle$ for each program $\pi$. The logic has a set of basic programs and a set of operators (sequential composition,
nondeterministic choice, test and iteration) that are used to inductively build the
set of non-basic programs. A Kripke semantics can be provided, with a frame
\( F = (W, R_\pi) \), where \( W \) is a non-empty set of possible program states and, for each
program \( \pi \), \( R_\pi \) is a binary relation on \( W \) such that \( (s, t) \in R_\pi \) if and only if there
is a computation of \( \pi \) starting in \( s \) and terminating in \( t \).

In modal logics in general and, consequently, in PDL an state has no internal
structure, in the sense that its possible constituents play no role in the process
of evaluating a formula in that state. In the last decade, many logical formalisms
have been proposed to cope with mutable data structures and updates. Separation
Logic \([17,21]\) was proposed to reasoning about imperatives programs with shared
mutable data structures, i.e structures with fields that can be updated and refer-
cenced in different points of its execution. An interesting extension of this logic
was proposed in \([15]\) to deal with concurrency. Moreover, in the field of Epistemic
Logic, many formalism have been proposed to deal with the dynamics of knowl-
edge in situations like, agent based systems, games and social networks. Logics like
Dynamic Epistemic Logic DEL \([11]\) and Public Announcement Logic PAL \([20]\) are
examples. These logics must deal with updates and changes of knowledge as actions
are performed by the agents or by the environment.

Another weakness PDL suffers is the lack of operators for the treatment of
parallelism and concurrence of programs. There are many extensions of PDL to
deal with this kind of operators \([19,18,2,3,12,16,1,6]\). The aim of all these logics is
to reasoning about parallel or concurrent programs.

In this work — although it is not our main concern to reasoning about parallel
execution of programs — we stay close to the above traditions, proposing an ex-
tension of the PDL regular language with a parallel operator and four operators:
two to store data and two to recover data. Our language, which we call PRSPDL,
is endowed with a semantics based on structured sets, in which the parallel oper-
ator together with the projections can be used to represent and manipulate data
structures. This semantics is a generalization of Kripke semantics which instead of
using a set of possible states uses a structured set of states. The idea of providing
structure to a set was inspired in fork algebras \([13,9]\) and has been used in many
formalisms \([8,22,10]\).

We start the study of PRSPDL by presenting the system, exemplify its expressive
power, and show a completeness result for a fragment of it.

The paper is structured as follows. Section 2 presents the basics on syntax and
semantics of the PRSPDL language. Section 3, presents some programs written
in PRSPDL language and provide some motivations. Section 4 presents axioms
and rules as well as soundness for the restricted system, RSPDL\(^0\), obtained from
PRSPDL by excluding the iteration, parallel composition and the test operators.
Section 5 presents a completeness result for RSPDL\(^0\). Finally, section 6 contains
some discussion about our contribution and some future works.
2 Syntax and semantics

The language is the usual PDL language with composition, choice, iteration, and test operators added with four atomic programs \(s_1, s_2, r_1, r_2\) and a binary operator of parallel composition. Intuitively, the semantics of these new operators is as follows. It is important to notice that our states are “ordered pairs”. The intended meaning of the nondeterministic program \(s_1\), called store first, is to store the current state as the first component of the resulting state, i.e., when the program \(s_1\) runs at state \(s\) it finishes its running producing a new state \((s, t)\) whose first coordinate is \(s\). Analogously, \(s_2\), called store second, stores the current state \(s\) as a second coordinate of the resulting state \((t, s)\). The intended meaning of the deterministic program \(r_1\), called recover first, is to recover a data (state) that is stored at the first component of the current state. When the program \(r_1\) runs at state \((s, t)\) it finishes its running at \(s\). Analogously, the program \(r_2\), called recover second, recovers the second component of the current state \((s, t)\) and finishes at state \(t\). Finally, when program \(\pi_1 \parallel \pi_2\) runs at state \((s_1, t_1)\), its effect is to run \(\pi_1\) and \(\pi_2\) in parallel at state \(s_1\) and \(t_1\) respectively, yielding a new state \((s_2, t_2)\).

Formally, we have the following definitions:

**Definition 2.1** Let \(\mathcal{A}ct = \{a, b, c, \ldots\}\) be the set of basic programs, typically denoted by \(\alpha\). The PRSPDL programs, typically denoted \(\pi\), are defined as follows:

\[
\pi ::= \alpha \mid \pi_1; \pi_2 \mid \pi_1 \cup \pi_2 \mid \pi_1^* \mid \pi_1 \parallel \pi_2 \mid ?\phi \mid s_1 \mid s_2 \mid r_1 \mid r_2.
\]

**Definition 2.2** The dynamic modal language with parallel composition, storing and recovering (PRSPDL) is a multi-modal language consisting of a set \(\Phi\) of countably many propositional symbols, typically denoted by \(p, q, r, \ldots\), the boolean connectives \(\neg\) and \(\land\), and a family of modal operators \(\{\langle \pi \rangle : \pi\text{ is a PRSPDL program}\}\). The PRSPDL formulas, typically denoted \(\phi\), are defined as follows:

\[
\phi ::= \bot \mid p \mid \neg \phi \mid \phi_1 \land \phi_2 \mid \langle \pi \rangle \phi.
\]

**Definition 2.3** A frame is a pair \(\mathcal{F} = (W, \{R_\pi : \pi\text{ is a program}\})\), where:

- \(W\) is a non-empty set,
- \(R_\pi \subseteq W \times W\), for each program \(\pi\).

**Definition 2.4** A model is a pair \(\mathcal{M} = (\mathcal{F}, V)\), where \(\mathcal{F}\) is a frame and \(V : \Phi \to 2^W\) is a valuation function mapping proposition symbols into subsets of \(W\).

**Definition 2.5** A model is standard when it satisfies the following conditions:

- \(R_{\pi_1; \pi_2} = R_{\pi_1} ; R_{\pi_2}\),
- \(R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}\),
- \(R_{\pi^*} = (R_\pi)^*\),
- \(R_{?\phi} = \{(w, w) \in W^2 : \mathcal{M}, w \models \phi\}\).

The main semantical difference between PDL and PRSPDL is that in the later formulas are interpreted on sets of structured states [9].
Definition 2.6 A set of structured states is a triple \((S, E, \star)\) where \(S\) is a non-empty set, \(E\) is an equivalence relation on \(S\), and \(\star : S^2 \to S\) is injective, i.e., a binary operation satisfying
\[
s_1 \star s_2 = t_1 \star t_2 \text{ iff } s_1 = t_1 \text{ and } s_2 = t_2,
\]
for every \((s_1, s_2), (t_1, t_2) \in E\).

Definition 2.7 A structured frame is a pair
\[
\mathcal{F} = ((S, E, \star), \{R_\pi : \pi \text{ is a program}\}),
\]
where:
- \((S, E, \star)\) is a non-empty set of structured states,
- \(R_\pi \subseteq E\), for each program \(\pi\),
- \((S, \{R_\pi : \pi \text{ is a program}\})\) is a frame.

A structured model is proper when it is based on a proper structured frame.

Definition 2.8 A structured frame \(\mathcal{F}\) is proper when it satisfies the following conditions:
- \(R_{s_1} = \{(s, s \star t) : s, t \in S\}\),
- \(R_{s_2} = \{(t, s \star t) : s, t \in S\}\),
- \(R_{r_1} = \{(s \star t, s) : s, t \in S\}\),
- \(R_{r_2} = \{(s \star t, t) : s, t \in S\}\),
- \(R_{\pi_1 \parallel \pi_2} = \{(s_1 \star t_1, s_2 \star t_2) : s_1, t_1, s_2, t_2 \in S \text{ and } (s_1, s_2) \in R_{\pi_1} \text{ and } (t_1, t_2) \in R_{\pi_2}\}\).

A structured model is proper when it is based on a proper structured frame.

Definition 2.9 An PRSPDL model is a proper standard model.

Observe that, in proper standard frames, the relations \(R_{s_1}\) and \(R_{r_1}\) are converse of each other. A similar remark applies to the relations \(R_{s_2}\) and \(R_{r_2}\). Besides, in proper standard frames, the following properties are true, where \(I_S = \{(s, s) : s \in S\}\) is the identity relation on \(S\):
\[
\begin{align*}
R_{s_1} ; R_{r_1} &= I_S \quad (1) \\
R_{s_2} ; R_{r_2} &= I_S \quad (2) \\
R_{s_1} ; R_{r_2} &= E \quad (3) \\
(R_{r_1} ; R_{s_1}) \cap (R_{r_2} ; R_{s_2}) &\subseteq I_S \quad (4) \\
R_{r_1} ; E = R_{r_2} ; E \quad (5)
\end{align*}
\]

It is important to notice that there are other properties that hold in proper standard frames, but the ones listed above are used in the proofs in the rest of the paper.

Definition 2.10 Let \(\mathcal{M}\) be a model. The notion of satisfaction of a formula \(\phi\) in a model \(\mathcal{M}\) at a state \(s\), notation \(\mathcal{M}, s \models \phi\) is inductively defined as follows:
further investigation.

used in diagrammatic reasoning based on allegories. We also leave this matter for
new useful operators. For instance, we can define the operators in [4,5] which are
composition on proper standard models are as expected:

\begin{itemize}
\item \(M, s \models \varphi \) iff there is \( t \in S \) such that \( sR_{s_1} t \) and \( M, t \models \varphi \) iff there are \( t, s_2 \in S \) such that \( t = s \ast s_2 \), and \( M, t \models \varphi \) iff there is \( s_2 \in S \) such that \( M, s \ast s_2 \models \varphi \). In another words, \( M, s \models \langle s_1 \rangle \varphi \) iff \( s \) is the first coordinate of an element standing for an ordered pair in which \( \varphi \) is true.
\item \( M, s \models \langle s_2 \rangle \varphi \) iff there is \( t \in S \) such that \( sR_{s_2} t \) and \( M, t \models \varphi \) iff there are \( t, s_1 \in S \) such that \( t = s_1 \ast s \), and \( M, t \models \varphi \) iff there is \( s_1 \in S \) such that \( M, s_1 \ast s \models \varphi \). In another words, \( M, s \models \langle s_2 \rangle \varphi \) iff \( s \) is the second coordinate of an element standing for an ordered pair in which \( \varphi \) is true.
\item \( M, s \models \langle r_1 \rangle \varphi \) iff there is \( t \in S \) such that \( sR_{r_1} t \) and \( M, t \models \varphi \) iff there are \( t, s_1, s_2 \in S \) such that \( s = s_1 \ast s_2 \), \( t = s_1 \) and \( M, s_1 \models \varphi \) iff there are \( s_1, s_2 \in S \) such that \( s = s_1 \ast s_2 \) and \( M, s_1 \models \varphi \). In another words, \( M, s \models \langle r_1 \rangle \varphi \) iff \( s \) stands for an ordered pair in whose first coordinate \( \varphi \) is true.
\item \( M, s \models \langle r_2 \rangle \varphi \) iff there is \( t \in S \) such that \( sR_{r_2} t \) and \( M, t \models \varphi \) iff there are \( t, s_1, s_2 \in S \) such that \( s = s_1 \ast s_2 \), \( t = s_2 \) and \( M, s_2 \models \varphi \) iff there are \( s_1, s_2 \in S \) such that \( s = s_1 \ast s_2 \) and \( M, s_2 \models \varphi \). In another words, \( M, s \models \langle r_2 \rangle \varphi \) iff \( s \) stands for an ordered pair in whose second coordinate \( \varphi \) is true.
\item \( M, s \models \langle \pi_1 \parallel \pi_2 \rangle \varphi \) iff there is \( t \in S \) such that \( sR_{\pi_1 \parallel \pi_2} t \) and \( M, t \models \varphi \) iff there are \( s_1, s_2, t_1, t_2 \in S \) such that \( s = (s_1 \ast s_2) \) and \( t = (t_1 \ast t_2) \) and \( s_1 R_{\pi_1} s_2 \) and \( t_1 R_{\pi_2} t_2 \), and \( M, t \models \varphi \). In another words, \( M, s \models \langle \pi_1 \parallel \pi_2 \rangle \varphi \) iff programs \( \pi_1 \) and \( \pi_2 \) are executed in parallel in \( s \) and reach a state \( t \) where \( \varphi \) holds.
\end{itemize}

By applying the PRSPDL operators to these basic programs we can define some new useful operators. For instance, we can define the operators in [4,5] which are used in diagrammatic reasoning based on allegories. We also leave this matter for further investigation.

If \( M, w \models \varphi \) for every state \( w \), we say that \( \varphi \) is globally satisfied in the model \( M \), notation \( M \models \varphi \). If \( \varphi \) is globally satisfied in all models \( M \) of a frame \( \mathcal{F} \), we say that \( \varphi \) is valid in \( \mathcal{F} \), notation \( \mathcal{F} \models \varphi \). Finally, if \( \varphi \) is valid in all frames, we say that \( \varphi \) is valid, notation \( \models \varphi \). Two formulas \( \varphi \) and \( \psi \) are semantically equivalent if \( \models \varphi \leftrightarrow \psi \).

3 Examples

In order to illustrate the usage of the PRSPDL language we present four examples. In all of them, we take advantage of the operations of storing and recovering, to
store some data and then recover this data during the computation. One powerful mechanism is the combination of these operations of store/recover with test, it allows for reasoning about properties that holds at previous states in the computation and use this information at the current state. In what follows, we abbreviate $\neg \perp$ as 1.

3.1 Example 1

In this example, we present a program $\pi_1$ which when start to run on input $u$, stores the initial state $u$ at the second coordinate of an ordered pair, then executes actions $\alpha$ and $\beta$ over the first coordinate of the pair, successively and, after that, returns to the initial state by restoring the second coordinate. This sequence of actions is displayed at the following diagram:

\[
\begin{align*}
\text{u} & \xrightarrow{s_2} (v_0, u) \xrightarrow{\alpha \parallel 1} (v_1, u) \xrightarrow{\beta \parallel 1} (v_2, u) \\
& \text{r}_2
\end{align*}
\]

When written as a PRSPDL program, $\pi_1$ can be specified as:

\[\pi_1 \equiv s_2; (\alpha \parallel 1); (\beta \parallel 1); r_2.\]

3.2 Example 2

In this example, we present a program $\pi_2$ which when start to run on input $u$, stores the initial state $u$ at the second coordinate of an ordered pair, then executes action $\alpha$ on the first coordinate of the current pair, until property $\phi$ is true, then after that, returns to the initial state by restoring the second coordinate of the current pair.

\[
\begin{align*}
\text{u} & \xrightarrow{s_2} (v_0, u) \xrightarrow{\neg \phi?; \alpha \parallel 1}^* (v_1, u) \xrightarrow{\phi? \parallel 1} (w, u) \\
& \text{r}_2
\end{align*}
\]

When written as a PRSPDL program, $\pi_2$ can be specified as:

\[\pi_2 \equiv s_2; (\neg \phi?; \alpha \parallel 1)^*; (\phi? \parallel 1); r_2.\]

3.3 Example 3

In this example, we present a program $\pi_3$ which when start to run on input $u$, stores the initial state $u$ at second coordinate of an ordered pair, then executes action $\alpha$ over the first coordinate of the pair and, after that, if property $\phi$ is true at the initial state then it performs action $\beta$ over the first coordinate of the current pair, else it performs action $\gamma$ over it. It is important to notice that formula $\phi$ is tested at the initial state and not at current state.

\[
\begin{align*}
\text{u} & \xrightarrow{s_2} (v_0, u) \xrightarrow{\alpha \parallel 1} (v_1, u) \xrightarrow{(r_2; \neg \phi?) \top?} (v_1, u) \xrightarrow{\beta \parallel 1} (v_2, u) \\
& \xrightarrow{(r_2; \neg \phi?) \top?} (v_1, u) \xrightarrow{\gamma \parallel 1} (v_3, u)
\end{align*}
\]
When written as a PRSPDL program, \( \pi_3 \) can be specified as:
\[
\pi_3 \equiv s_2; (\alpha \parallel 1); ((\langle r_2; \phi? \rangle \top)?; \beta \parallel 1) \cup (\langle r_2; \neg \phi? \rangle \top)?; (\gamma \parallel 1).
\]

### 3.4 Example 4

In this example, we present a program \( \pi_4 \) which when start to run on input \( u \), stores the initial state \( u \) at second coordinate of an ordered pair, then executes action \( \alpha \) over the first coordinate of the pair and, after that, it either stores the current state as the second coordinate of an ordered pair, executes action \( \alpha \) over the new first coordinate, and returns to the second pair obtained in the computation; or executes action \( \beta \) over the first coordinate of the pair and returns to the initial state.

When written as a PRSPDL program, \( \pi_4 \) can be specified as:
\[
\pi_4 \equiv s_2; (\alpha \parallel 1); ((s_2; (\alpha \parallel 1); r_2) \cup (\beta \parallel 1)); r_2.
\]

### 4 Axiomatic System for RSPDL\(^0\)

In this section, we restrict the language presented in Section 2 to a fragment called RSPDL\(^0\). In this fragment, we do not allow the use of the operators of test (\(?\)), iteration (\(\star\)) and parallel composition (\(\parallel\)). We intend to use the work reported in this fragment as a basis for the investigation of the whole language.

Our objective is to present a set of axioms for RSPDL\(^0\) and prove soundness and completeness for it with respect to the semantics on structured sets. We use the standard Boolean abbreviations \(\top\), \(\lor\), \(\rightarrow\) and \(\leftrightarrow\), and the modal abbreviations \([\pi]\phi := \neg\langle\pi\rangle\neg\phi\), for every program \(\pi\).

Let RSPDL\(^0\) be the modal logic defined by the schemata and rules in Table 1.

Axiom 2 is the standard \(K\) axiom of distributivity. Axioms 3 and 4 are the standard PDL axioms for composition and non-deterministic choice, respectively. Axiom 5 expresses that the relations \(R_{r_1}\) and \(R_{r_2}\), interpreting \(r_1\) and \(r_2\), respectively, are functional. Axiom 6 is the standard temporal axiom expressing that the relations \(R_{s_1}\) and \(R_{r_1}\) (also \(R_{s_2}\) and \(R_{r_2}\)), interpreting \(s_1\) and \(r_1\) (also \(s_2\) and \(r_2\)), respectively, are the converse of each other. Axiom 7 expresses that the relations \(R_{r_1}\) and \(R_{r_2}\), interpreting \(r_1\) and \(r_2\), respectively, have the same domain. Axiom 8 warrants unicity of ordered pairs. Axioms 9, 10 and 11, in conjunct, express that the relation \(R_{s_1};r_2\), interpreting the composite program \(s_1;r_2\), is an equivalence relation on its field.

**Theorem 4.1 (Soundness)** If \(\vdash \varphi\), then \(\varphi\) is valid in all RSPDL\(^0\) frames.

**Proof.** The proof of the soundness of the first four axioms are usual.
Axioms

1. All tautologies

2. \([\pi](\varphi \rightarrow \psi) \rightarrow ([\pi]\varphi \rightarrow [\pi]\psi)\)

3. \([\pi_1; \pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi\)

4. \([\pi_1 \cup \pi_2]\varphi \leftrightarrow [\pi_1]\varphi \land [\pi_2]\varphi\)

5. \(\langle r_1 \rangle \varphi \rightarrow [r_1]\varphi\)
   \(\langle r_2 \rangle \varphi \rightarrow [r_2]\varphi\)

6. \(\varphi \rightarrow [s_1]\langle r_1 \rangle \varphi\)
   \(\varphi \rightarrow [r_1]\langle s_1 \rangle \varphi\)
   \(\varphi \rightarrow [s_2]\langle r_2 \rangle \varphi\)
   \(\varphi \rightarrow [r_2]\langle s_2 \rangle \varphi\)

7. \(\langle r_1 \rangle \top \leftrightarrow \langle r_2 \rangle \top\)
   \(\langle s_1 \rangle \top \leftrightarrow \langle s_2 \rangle \top\)

8. \(\langle s_1; r_1 \rangle \varphi \rightarrow [s_1; r_1] \varphi\)
   \(\langle s_2; r_2 \rangle \varphi \rightarrow [s_2; r_2] \varphi\)

9. \([s_1; r_2] \varphi \rightarrow \varphi\)

10. \(\varphi \rightarrow [s_1; r_2]\langle s_1; r_2 \rangle \varphi\)

11. \([s_1; r_2] \varphi \rightarrow [s_1; r_2][s_1; r_2] \varphi\)

Inference Rules

\[\text{MP)} \quad \frac{\phi \quad \phi \rightarrow \psi}{\psi}\]

\[\text{Nec}_\pi) \quad \frac{\phi}{[\pi]\phi}\]

Table 1
RSPDL\(^{d}\) axiomatics.
Axiom 5. We treat just the instances relative to r₁. The instances relative to r₂ can be treated in a similar way.

Suppose \( \mathcal{M}, s \models (r₁)\varphi \). So, there are \( s₁ \) and \( s₂ \) such that \( s = s₁ \star s₂ \) and \( \mathcal{F}, V, s₁ \models \varphi \). Let \( u \in S \) be any state for which \( sR₁u \). By definition, there are \( s'₁ \) and \( s'₂ \) such that \( s = s'₁ \star s'₂ \) and \( u = s'₁' \). From \( s = s₁ \star s₂ = s'₁ \star s'₂ \) and the injectivity of \( \star \), we have \( s₁ = s'₁' \). Since, \( \mathcal{M}, s₁ \models \varphi \) and \( u = s'₁ = s₁ \), we have \( \mathcal{M}, u \models \varphi \). So, we can conclude \( \mathcal{M}, u \models \varphi \), for every \( u \in S \) such that \( sR₁u \), i.e., \( \mathcal{M}, s \models [s₁]\varphi \).

Axiom 6. We treat just the instances relative to \( s₁ \) and \( r₁ \). The instances relative to \( s₂ \) and \( r₂ \) can be treated in a similar way.

Suppose \( \mathcal{M}, s \models \varphi \). Let \( t \) be such that \( sR₁t \). Hence, there is a \( s₂ \in S \) such that \( t = s \star s₂ \). So, there are \( s₁' \), \( s₂' \) such that \( t = s₁' \star s₂' \) and \( \mathcal{M}, s₁' \models \varphi \), which is the same as \( \mathcal{M}, t \models (r₁)\varphi \). So, we can conclude \( \mathcal{M}, t \models (r₁)\varphi \), for every \( t \in S \) such that \( sR₁t \), i.e., \( \mathcal{M}, s \models [s₁](r₁)\varphi \).

Now, suppose again that \( \mathcal{M}, s \models \varphi \) and let \( t \) be such that \( sR₁t \). Hence, there are \( s₁ \), \( s₂ \) in \( S \) such that \( s = s₁ \star s₂ \) and \( t = s₁ \star s₂ \). By definition, \( s₁R₁s₁ \star s₂ \) and \( \mathcal{M}, s₁ \star s₂ \models \varphi \). So, there is a \( u \in S \) such that \( tR₁u \) and \( \mathcal{M}, u \models \varphi \), which is the same as \( \mathcal{M}, t \models (s₁)\varphi \). So, we can conclude \( \mathcal{M}, t \models (s₁)\varphi \), for every \( t \in S \) such that \( sR₁t \), i.e., \( \mathcal{M}, s \models [r₁](s₁)\varphi \).

Axiom 7. We treat just the instance relative to \( r₁ \) and \( r₂ \). The instance relative to \( s₁ \) and \( s₂ \) can be treated in a similar way.

We have that \( \mathcal{M}, s \models (r₁)T \) iff there is some \( t \in S \) such that \( sR₁t \) and \( \mathcal{M}, t \models T \), iff there is some \( t \in S \) such that \( sR₁t \), iff there are \( t, t' \in S \) such that \( s = t \star t' \), iff there is some \( t' \in S \) such that \( sR₁t' \), iff there is some \( t' \in S \) such that \( sR₂t' \) and \( \mathcal{M}, t' \models T \), iff \( \mathcal{M}, s \models (r₂)T \).

Axiom 8. We treat just the instance relative to \( s₁ \) and \( r₁ \). The instance relative to \( s₂ \) and \( r₂ \) can be treated in a similar way.

Suppose \( \mathcal{M}, s \models (s₁;r₁)\varphi \). Hence, there are \( u, v \in S \) such that \( sR₁u \) and \( \mathcal{M}, v \models \varphi \). Hence, there are \( u, v, v', s' \in S \) such that \( u = s \star s' \), \( u = v \star v' \), and \( \mathcal{M}, v \models \varphi \). By the injectivity of \( \star \), we have \( s = v \). Hence, \( \mathcal{M}, s \models \varphi \). Now, let \( t \in S \) be such that \( sR₁t \). Hence, there are \( u, t', s' \) in \( S \) such that \( s = s \star s' \) and \( u = t \star t' \). By the injectivity of \( \star \), we have \( s = t \). Hence, \( \mathcal{M}, t \models \varphi \).

Axiom 9. Suppose \( \mathcal{M}, s \models [s₁;r₂]\varphi \). Hence, for every \( t \in S \) if \( sR₁t \), then \( \mathcal{M}, t \models \varphi \). Now, since we have \( sR₁s₁ \star s₁r₂s \) and \( s \star sR₂s \), we have \( sR₁r₂s \). So, we conclude \( \mathcal{M}, s \models [s₁;r₂]\varphi \).

Axiom 10. Suppose \( \mathcal{M}, s \models \varphi \). Let \( t \in S \) be such that \( sR₁r₂t \). Now, since we have \( tR₁t \) and \( t \star sR₂s \), we have \( tR₁r₂s \). This, together with \( \mathcal{M}, s \models \varphi \), gives us \( \mathcal{M}, t \models (s₁;r₂)\varphi \). So, we can conclude that \( \mathcal{M}, t \models (s₁;r₂)\varphi \), for every \( t \in S \) such that \( sR₁r₂ \), i.e., \( \mathcal{M}, s \models [s₁;r₂]\varphi \).

Axiom 11. Suppose \( \mathcal{M}, s \models [s₁;r₂]\varphi \). Hence, for every \( t \in S \) if \( sR₁r₂t \), then \( \mathcal{M}, t \models \varphi \). Let \( u, v \in S \) such that \( sR₁u \) and \( uR₁v \). Since we have \( sR₁s \star u \) and \( s \star uR₁v \), we have \( sR₁r₂ \), which gives us \( \mathcal{M}, v \models \varphi \). Hence, we have \( \mathcal{M}, v \models \varphi \) for every \( v \in S \) such that \( uR₁r₂v \), i.e., \( \mathcal{M}, u \models [s₁;r₂]\varphi \). Moreover, we have \( \mathcal{M}, u \models [s₁;r₂]\varphi \) for every \( u \in S \) such that \( sR₁r₂u \), i.e., \( \mathcal{M}, s \models [s₁;r₂][s₁;r₂]\varphi \), as required.
5 Completeness of RSPDL\(^0\)

System RSPDL\(^0\) is the restriction of PRSPDL obtained by the exclusion of the iteration operator *, the test operator and the parallel composition operator. A proof system for RSPDL\(^0\) was presented in Section 4. In this section, we prove the completeness of this proof system for RSPDL\(^0\).

**Theorem 5.1** If \(\not\models_{\text{RSPDL}\(^0\)} \varphi\), then there is a model in which \(\varphi\) is not valid.

**Proof.** The canonical model is the structure

\[\mathcal{M}^c = (W^c, \{R^c_{\pi} : \pi \text{ is a program}\}, V^c),\]

defined as usual:

- \(W^c\) is the set of all maximal consistent sets of formulas,
- \(sR^c_{\pi}t\) iff formula \(\varphi\) is in \(t\) for every formula \([\pi]\varphi\) in \(s\),
- \(V^c_p\) is the set of all maximal consistent sets of formulas containing \(p\).

The canonical frame is the structure \(\mathcal{F}^c = (W^c, \{R^c_{\pi} : \pi \text{ is a program}\})\).

Observe that we do not have neither that the canonical frame nor the canonical model is a proper frame or model. This is because we do not have that \(W^c\) is a structured set. Anyway, by a standard modal logic reasoning, we have:

**Lemma 5.2** If a formula \(\varphi\) is such that \(\not\models \varphi\), then there is some state \(w\) in the canonical model \(\mathcal{M}^c\) such that \(\mathcal{M}^c, w \not\models \varphi\).

Axioms 3 and 4 warrant that the canonical model is standard.

**Lemma 5.3** \(\mathcal{M}^c\) is standard.

**Proof.** First, we shall prove \(R^c_{\pi_1;\pi_2} = R^c_{\pi_1}; R^c_{\pi_2}\).

To prove the inclusion from left to right, suppose \(\Sigma\) and \(\Sigma'\) are MCS satisfying \(\Sigma R^c_{\pi_1;\pi_2} \Sigma'\). First, we prove that the set of formulas \(\Gamma = \{\varphi : [\pi_1]\varphi \in \Sigma\} \cup \{\neg[\pi_2]\psi : \psi \not\in \Sigma'\}\) is consistent. In fact, if \(\Gamma \vdash \bot\), then there are \(\varphi_1, \ldots, \varphi_n\) such that \([\pi_1]\varphi_1, \ldots, [\pi_1]\varphi_n \in \Sigma\) and there are \(\psi_1, \ldots, \psi_m \not\in \Sigma'\) such that \(\vdash \varphi_1 \land \cdots \land \varphi_n \land \neg[\pi_2]\psi_1 \land \cdots \land \neg[\pi_2]\psi_m \rightarrow \bot\). By normality, we obtain \(\vdash \varphi_1 \land \cdots \land \varphi_n \rightarrow ([\pi_2]\psi_1 \lor \cdots \lor [\pi_2]\psi_m)\). Hence, \(\vdash [\pi_1]\varphi_1 \land \cdots \land [\pi_1]\varphi_n \neg \rightarrow [\pi_1]([\pi_2]\psi_1 \lor \cdots \lor [\pi_2]\psi_m)\). So, \(\Sigma \vdash [\pi_1][\pi_2]\psi_1 \lor \cdots \lor [\pi_1][\pi_2]\psi_m\). Now, by applying Axiom 3, we have \(\Sigma \vdash [\pi_1;\pi_2]\psi_1 \lor \cdots \lor [\pi_1;\pi_2]\psi_m\), which, since \(\Sigma R^c_{\pi_1;\pi_2} \Sigma'\), implies \(\psi_1 \in \Sigma'\) or \(\cdots\) or \(\psi_m \in \Sigma'\), a contradiction.

Now, let \(\Sigma''\) be a maximal consistent set such that \(\Gamma \subseteq \Sigma''\). For each \(\varphi\) such that \([\pi_1]\varphi \in \Sigma\), we have \(\varphi \in \Gamma \subseteq \Sigma''\). Hence, \(\Sigma R^c_{\pi_1} \Sigma''\). Besides, let \(\psi \in \Sigma'\), that is, \(\neg \psi \not\in \Sigma'\). We have \(\neg [\pi_2] \neg \psi \in \Gamma \subseteq \Sigma''\), which is the same as \(\langle \pi_2 \rangle \psi \in \Sigma''\). Hence, \(\Sigma'' R^c_{\pi_2} \Sigma'\). From these, we obtain \(\Sigma R^c_{\pi_1}; R^c_{\pi_2} \Sigma'\).

To prove the other inclusion, let \(\Sigma\) and \(\Sigma'\) be MCS satisfying \(\Sigma R^c_{\pi_1}; R^c_{\pi_2} \Sigma'\). Let \(\Sigma''\) be a MCS such that \(\Sigma R^c_{\pi_1} \Sigma''\) and \(\Sigma'' R^c_{\pi_2} \Sigma'\). Let \(\varphi \in \Sigma'\). Hence, \(\langle \pi_2 \rangle \varphi \in \Sigma''\), and so \(\langle \pi_1 \rangle \langle \pi_2 \rangle \varphi \in \Sigma\). Now, by Axiom 3, we have \(\langle \pi_1;\pi_2 \rangle \varphi \in \Sigma\) which proves \(\Sigma R^c_{\pi_1;\pi_2} \Sigma'\).
The proof that $R_{\pi_1 \cup \pi_2}^c = R_{\pi_1}^c \cup R_{\pi_2}^c$ is trivial, using Axiom 4. \qed

Axioms 5–11 warrant that relations have the required properties to make $W^c$ a structured set.

**Lemma 5.4** (i) Relations $R_{r_1}^c$ and $R_{r_2}^c$ are functional.

(ii) Relations $(R_{r_1}^c)^{-1}; R_{r_1}^c$ and $(R_{r_2}^c)^{-1}; R_{r_2}^c$ are injective.

(iii) Relations $R_{r_1}^c$ and $R_{r_2}^c$ have the same domain.

(iv) Relations $R_{s_1}^{s_2^c}$ and $R_{s_2}^{s_1^c}$ are the converse of each other. Also, relations $R_{s_2}^{s_1^c}$ and $R_{s_2}^{s_2^c}$ are the converse of each other.

**Proof.** To prove $R_{r_1}^c$ is functional, we proceed as follows. Suppose $\Sigma$, $\Sigma_1$, and $\Sigma_2$ are MCSs satisfying $\Sigma R_{r_1}^c \Sigma_1$ and $\Sigma R_{r_1}^c \Sigma_2$. Let $\phi \in \Sigma_1$. Since $\Sigma R_{r_1}^c \Sigma_1$, we have $\langle r_1 \rangle \phi \in \Sigma$. Now, by applying Axiom 5, we obtain $[r_1] \phi \in \Sigma$, and since $\Sigma R_{r_1}^c \Sigma_2$, we have $\phi \in \Sigma_2$. Hence, $\Sigma_1 \subseteq \Sigma_2$. The inclusion $\Sigma_2 \subseteq \Sigma_1$ can be proved analogously.

The proof that $R_{r_2}^c$ is functional is entirely similar.

To prove $(R_{r_1}^c)^{-1}; R_{r_2}^c$ is injective, we proceed as follows. Suppose $\Sigma$, $\Sigma_1$, and $\Sigma_2$ are MCSs satisfying $\Sigma (R_{r_1}^c)^{-1}; R_{r_1}^c \Sigma_1$ and $\Sigma (R_{r_1}^c)^{-1}; R_{r_1}^c \Sigma_2$. Let $\phi \in \Sigma_1$. Since $\Sigma (R_{r_1}^c)^{-1}; R_{r_1}^c \Sigma$, there is some MCS $\Sigma'$ such that $\Sigma' R_{r_1}^c \Sigma_1$ and $\Sigma' R_{r_1}^c \Sigma_2$. Since $\phi \in \Sigma_1$, and $\Sigma' R_{r_1}^c \Sigma_2$, we have $\langle r_1 \rangle \phi \in \Sigma'$. Now, by applying Axiom 5, we obtain $[r_1] \phi \in \Sigma'$, and since $\Sigma' R_{r_1}^c \Sigma$, we have $\phi \in \Sigma$. Besides, since $\Sigma 2 (R_{r_1}^c)^{-1}; R_{r_1}^c \Sigma$, there is some MCS $\Sigma''$ such that $\Sigma'' R_{r_1}^c \Sigma_1$ and $\Sigma'' R_{r_1}^c \Sigma_2$. Since $\Sigma'' R_{r_1}^c \Sigma$ and $\phi \in \Sigma$, we have $\langle r_1 \rangle \phi \in \Sigma''$. Again, by applying Axiom 5, we obtain $[r_1] \phi \in \Sigma''$, and since $\Sigma'' R_{r_1}^c \Sigma_2$, we have $\phi \in \Sigma_2$. Hence, $\Sigma_1 \subseteq \Sigma_2$. The inclusion $\Sigma_2 \subseteq \Sigma_1$ can be proved analogously.

The proof that $(R_{r_1}^c)^{-1}; R_{r_2}^c$ is injective is entirely similar.

To prove that $R_{r_1}^c$ and $R_{r_2}^c$ have the same domain, we proceed as follows. Suppose $\Sigma$ and $\Sigma'$ are MCSs satisfying $\Sigma R_{r_1}^c \Sigma'$. Since $\top \in \Sigma'$, we have $\langle r_1 \rangle \top \in \Sigma$. Now, by applying Axiom 7, we obtain $[r_1] \top \in \Sigma$ and, then there is some MCS $\Sigma''$ such that $\Sigma R_{r_1}^c \Sigma''$. Hence, the domain of $R_{r_1}^c$ is included in the domain of $R_{r_2}^c$.

The proof of the other inclusion is entirely similar.

The proofs that $R_{s_1}^{s_2^c}$ and $R_{s_2}^{s_1^c}$ are converses, as well the proof of the same property for $R_{s_2}^{s_1^c}$ and $R_{s_2}^{s_2^c}$, are as usual in temporal logic, using Axiom 6. \qed

Given $\Sigma \in W^c$, let $\mathcal{M}^\Sigma$ be the sub-model of $\mathcal{M}^c$, generated by $\Sigma$ and $(R_{r_1}^c)^{-1}; R_{r_2}^c$. By the Generated Sub-model Lemma, we have:

**Lemma 5.5** (i) For any $\Sigma \in W^c$, $\mathcal{M}^\Sigma \models \text{RSPDL}^0$.

(ii) If $\not\models_{\text{RSPDL}^0} \varphi$, then there is a $\Sigma \in W^c$ such that $\mathcal{M}^\Sigma, \Sigma \not\models \varphi$.

(iii) Relation $(R_{r_1}^c)^{-1}; R_{r_2}^c$ is total.

Now we see prove that, given $\Sigma \in W^c$, the model $\mathcal{M}^\Sigma$ has enough nice properties to be the counter-model we need. In fact, by applying a reasoning similar to that employed in [9], we can prove that $\mathcal{M}^\Sigma$ is indeed a model of $\text{RSPDL}^0$, i.e. that $W^\Sigma$ is a structured set. More specifically, since $R_{s_1}, R_{r_1}, R_{s_2}$ and $R_2$ are programs of $\text{RSPDL}^0$ and since they satisfy Lemmas 5.4 and 5.5, we have that $R_{s_1}, R_{r_1}, R_{s_2}$ and $R_2$ are functional relations sharing the same domain, covering $W^\Sigma \times W^\Sigma$, and
warranting unicity of ordered pairs. These conditions suffice to define an injective function \( \star : W^\Sigma \times W^\Sigma \to W^\Sigma \) for which \( R_{s1} = \{(w, w \star v) : w, v \in W^\Sigma\} \), \( R_{s2} = \{(v, w \star v) : w, v \in W^\Sigma\} \), \( R_{r1} = \{(w \star v, w) : w, v \in W^\Sigma\} \), and \( R_{r2} = \{(w \star v, v) : w, v \in W^\Sigma\} \), as follows.

Define \( f \subseteq (W^\Sigma \times W^\Sigma) \times W^\Sigma \) in the following way, for all \( w, v, u \in W^\Sigma \):

\[
((w, v), u) \in f \iff (u, w) \in R_{r1} \text{ and } (u, v) \in R_{r2}.
\]

We have that \( f \) is an injective functional relation. All these facts together allow us to define \( \star : W^\Sigma \times W^\Sigma \to W^\Sigma \) by setting:

\[
w \star v = f(w, v),
\]

for any pair \( (w, v) \in W^\Sigma \times W^\Sigma \). From this definition it is obvious that

\[
w \star v = u \iff (u, w) \in R_{r1} \text{ and } (u, v) \in R_{r2},
\]

for any \( (w, v) \in W^\Sigma \times W^\Sigma \) and \( u \in W^\Sigma \), and this gives us \( R_{s1} = \{(w, w \star v) : w, v \in W^\Sigma\} \), \( R_{s2} = \{(v, w \star v) : w, v \in W^\Sigma\} \), \( R_{r1} = \{(w \star v, w) : w, v \in W^\Sigma\} \), and \( R_{r2} = \{(w \star v, v) : w, v \in W^\Sigma\} \).

To conclude the proof just observe that \( f \) is functional and injective, and \( \text{Dom} f = W^\Sigma \times W^\Sigma \). This is what we need to show that \( M^\Sigma \) is a proper model.

### 6 Final Remarks

This paper starts the study of PRSPDL, an extension of the PDL regular language with a parallel operator and four operators: two to store data and two to recover data. More specifically, we exemplify the expressive power of PRSPDL and present an axiomatization for a restrict fragment, RSPDL\(^0\), without parallel composition, iteration and test and provide a proof of completeness for this fragment.

The semantics of PRSPDL is a generalization of Kripke semantics with a notion of structured set of possible states instead of sets of states. Structured sets allows one to represent structured data in a very natural way, as we show in some specific examples.

There are many possibilities for future work, we just list the most prominent. First, we would like to establish decidability and complexity issues for the fragment RSPDL\(^0\). Second, we would like to provide an axiomatization for PRSPDL with parallel composition, iteration and test, provide a proof of completeness, and investigate decidability and complexity questions for it. Finally, it would be interesting to have some application of the PRSPDL language in specification of properties of programs with mutable data structures and updates.

### References


