Resource separation in dynamic logic of propositional assignments (In Press, Journal Pre-proof)

Joseph Boudou, Andreas Herzig, Nicolas Troquard

▶ **To cite this version:**

Joseph Boudou, Andreas Herzig, Nicolas Troquard. Resource separation in dynamic logic of propositional assignments (In Press, Journal Pre-proof). Journal of Logical and Algebraic Methods in Programming, 2021, pp.100683. 10.1016/j.jlamp.2021.100683 . hal-03210604

HAL Id: hal-03210604
https://hal.science/hal-03210604

Submitted on 28 Apr 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Resource Separation in Dynamic Logic of Propositional Assignments

Joseph Boudou[1], Andreas Herzig[2], Nicolas Troquard[3]

[1] IRIT, University of Toulouse, France
[2] IRIT, CNRS, France
[3] Free University of Bozen-Bolzano, Italy

Abstract. We extend dynamic logic of propositional assignments by adding an operator of parallel composition that is inspired by separation logics. We provide an axiomatisation via reduction axioms, thereby establishing decidability. We also prove that the complexity of both the model checking and the satisfiability problem stay in PSPACE.

Keywords: Dynamic logic, separation logic, propositional assignments, parallel composition, concurrency

1 Introduction

It is notoriously delicate to extend Propositional Dynamic Logic PDL with an operator of parallel composition of programs. Several attempts were made in the literature: Abrahamson as well as Mayer and Stockmeyer studied a semantics in terms of interleaving [Abr80,MS96]; Peleg and Goldblatt modified the interpretation of programs from a relation between possible worlds to a relation between possible worlds and sets thereof [Pel87,Gol92]; Balbiani and Vakarelov studied the interpretation of parallel composition of programs π_1 and π_2 as the intersection of the accessibility relations interpreting π_1 and π_2 [BV03]. However, it seems fair to say that there is still no consensus which of these extensions is the 'right' one.

Dynamic Logic of Propositional Assignments DL-PA [BHT13,BHST14] is a version of Propositional Dynamic Logic (PDL) whose atomic programs are assignments of propositional variables p to true or false, respectively written $+p$ and $-p$. We and coauthors have shown that many knowledge representation concepts and formalisms can be captured in DL-PA, such as update and revision operations [Her14], database repair [FHR19], lightweight dynamic epistemic logics [CS15,CHM+16,CS17], planning [HMNDBW14], and judgment aggregation [NGH18]. The mathematical properties of DL-PA are simpler than those of PDL, in particular, the Kleene star can be eliminated [BHT13] and satisfiability and model checking are both PSPACE complete [BHST14].

In this paper we investigate how dynamic logic can be extended with a program operator of parallel composition $\pi_1 \| \pi_2$ of two programs π_1 and π_2 that is inspired by separation logic. The latter was studied in the literature as an account of concurrency, among others by Brookes and by O'Hearn [O'H04,Bro04,BO16]. Their Concurrent Separation Logic is characterised by two main principles:

1. When two programs are executed in parallel then the state of the system is partitioned ('separated') between the two programs: the perception of the state and its modification is viewed as being local to each of the two parallel programs. Each of them therefore has a partial view of the global state. This entails that parallelism in itself does not modify the state of the system: the parallel execution of two programs that do nothing does not change the state. The formula $\varphi \to [\top?\|\top?]\varphi$ should therefore be valid, where "$\top?$" is the test that $\top$ is true (which always succeeds).
2. The execution of a parallel program $\pi_1\|\pi_2$ should be insensitive to the way the components of π_1 and π_2 are interleaved. Hence "race conditions" [BO16] must be avoided: the execution should not depend on the order of execution of atomic programs in π_1 and π_2 (where we consider tests to be atomic, too). Here we interpret this requirement in a rather radical way: when there is a race condition between two programs then they cannot be executed in parallel. For example, the parallel program $+p\|-p$ where $+p$ makes p true and $-p$ makes p false is inexecutable because there is a conflict: the two possible interleavings $+p;-p$ and $-p;+p$ are not equivalent. We even consider that $+p\|+p$ is inexecutable, which some may consider a bit over-constrained[4].

In formal frameworks for the verification of parallel programs such as the one proposed by Brookes and O'Hearn [Bro04,O'H04], allowing race conditions is a necessary feature for the framework to be able to prove a property of programs, namely that they are race-free. On the contrary, dynamic logics permit to prove properties of formulas, and atomic actions are actually even totally abstracted away in most dynamic logics. In such abstract settings, whether two given atomic actions can be executed concurrently is a semantic detail of each model. For instance, in dynamic logics with a parallel composition based on separation (like in [BdFV11,Bou16,BB18]), the separation relation of the model provides the possibility to forbid race conditions. The race condition issue arises in logics based on DL-PA because atomic actions are concrete: basically, each atomic action potentially changes the valuation of exactly one propositional variable. Hence it is natural to consider access to the propositional variables as the main resource. In this perspective, a decision has to be made on whether the separation of these resources is strict or not, i.e., whether race conditions are allowed or not. In the present work, we have chosen a strict separation semantics because it is the simplest solution satisfying the two principles stated above.

We have not yet said what one should understand by a DL-PA system state. A previous approach of one of us only considered the separation of valuations, i.e., of truth values of propositional variables [Her13]. Two separating conjunctions in the style of separation logic were defined on such models. This however did not allow one to define an adjoint implication as usually done in the separation logic literature, which was somewhat unsatisfactory. Another paper that was coauthored by one of us has richer models: valuations are supplemented by information about writability of variables [HMV19]. It is supposed that a variable can only be assigned by a program when it is writable. Splitting and merging of such models can be defined in a natural way, thus providing a meaningful interpretation of parallel composition. When parallel composition is based

[4] This restriction can be related to the fact that the formula $e \mapsto e' * e \mapsto e'$ is unsatisfiable in Separation Logic [Rey02].

on separation, writability information permits to resolve merge conflicts. Consider for instance the executions of the two programs $\top?\|{+}p$ and $-p\|{+}p$ from a state in which p is false. We argue that intuitively, the former program should lead to a state in which p is true whereas the latter program should either not be executable or non-deterministically lead to two possible states, one where p is true and one where p is false. However, without writability information, the states before the merge of each of these programs turn out to be identical: p is false in the left branch but true in the right branch. Writability information allows the merge operation to distinguish these two situations and to resolve the conflict in the former case.

We here push this program further and consider models having moreover information about readability of variables. We suppose that writability implies readability[5] and that a variable can only be tested if it is readable. Our tests $\varphi?$ therefore differ from the standard tests of PDL and DL-PA in that their executability depends on whether the relevant variables are readable. In particular, while $\langle p? \rangle \top \rightarrow p$, remains valid, its converse $p \rightarrow \langle p? \rangle \top$ becomes invalid in our logic: it may be the case that p is true but cannot be read.

Distinguishing the variables that can be tested permits some useful checks. Consider for instance the execution of the program $(-p; q?) \,\|\, (-q; p?)$ from a state in which both p and q are true. Without readability, this program can be executed and results in a state in which both p and q are false. However, no interleaving of this program can be executed. Adding readability of variables permits to detect this issue: we enforce that a variable cannot be read by one subprogram of a parallel composition if it can be written by the other subprogram.

The paper is organised as follows. In Section 2 we define models and the two ternary relations 'split' and 'merge' on models. In Section 3 we define the language of our logic and in Section 4 we give the interpretation of formulas and programs. In Section 5 we axiomatise the valid formulas by means of reduction axioms and in Section 6 we establish that the satisfiability problem is PSPACE complete. Section 7 sums up our contributions and discusses related work and the application to parallel planning. The annex contains a proof of associativity of parallel composition.[6]

2 Models and Their Splitting and Merging

Let $\mathbb{P}$ be a countable set of propositional variables. We use $p, q, \ldots$ for elements of $\mathbb{P}$. A model (alias a system state) is a triple $\mathsf{m} = \langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V} \rangle$ where Rd, Wr, and V are subsets of $\mathbb{P}$ such that $\mathsf{Wr} \subseteq \mathsf{Rd}$. The intuition is that Rd is the set of readable variables, Wr is the set of writable variables, and V is a valuation: its elements are true, while those of its

[5] As suggested by one of the reviewers of a previous version of the present paper [BHT19], this constraint may be relaxed and one may suppose that a program can modify a variable without being able to read its value. This would simplify the presentation of the logic; however, we believe that our inclusion constraint is natural in most applications.

[6] The present paper is a more elaborate version of [BHT19]. It contains proofs of the results, more motivation and explanations, and furthermore proves that our operator of parallel composition is associative.

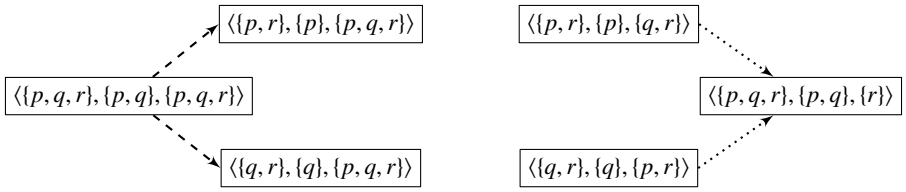

Fig. 1. Examples of split and merge operations: the left side illustrates the split of the model $\langle\{p,q,r\},\{p,q\},\{p,q,r\}\rangle$ into $\langle\{p,r\},\{p\},\{p,q,r\}\rangle$ and $\langle\{q,r\},\{q\},\{p,q,r\}\rangle$; the right side illustrates the merge of the models $\langle\{p,r\},\{p\},\{p,q,r\}\rangle$ and $\langle\{q,r\},\{q\},\{p,q,r\}\rangle$ into $\langle\{p,q,r\},\{p,q\},\{r\}\rangle$.

complement $\mathbb{P} \setminus \mathsf{V}$ are false. The constraint that $\mathsf{Wr} \subseteq \mathsf{Rd}$ means that writability implies readability.

The special case when $\mathsf{Wr} = \mathbb{P}$ is typical when checking the validity or the satisfiability of a formula φ. Lemma 1 in Section 4 will prove that, as expected, this case is equivalent to the case where Wr and Rd are the set of propositional variables occurring in φ. In fact, the writability and readability sets are mostly useful for checking properties of subprograms of parallel compositions. On the other end of the spectrum, the case $\mathsf{Wr} = \emptyset$ is of little interest since the only executable programs will be those without assignments.

Two models $\mathsf{m}_1 = \langle \mathsf{Rd}_1, \mathsf{Wr}_1, \mathsf{V}_1\rangle$ and $\mathsf{m}_2 = \langle \mathsf{Rd}_2, \mathsf{Wr}_2, \mathsf{V}_2\rangle$ are *RW-disjoint* if and only if the writable variables of one model and the readable variables of the other are disjoint, i.e., if and only if $\mathsf{Wr}_1 \cap \mathsf{Rd}_2 = \mathsf{Wr}_2 \cap \mathsf{Rd}_1 = \emptyset$. For example, $\mathsf{m}_1 = \langle\{p\},\{p\},\emptyset\rangle$ and $\mathsf{m}_2 = \langle\{p\},\emptyset,\emptyset\rangle$ fail to be RW-disjoint: in m_1, some program π_1 modifying the value of p may be executable, while for programs executed in m_2, the value of p may differ depending on whether it is read before or after the modification by π_1 took place.

As writability implies readability, RW-disjointness of m_1 and m_2 implies that Wr_1 and Wr_2 are disjoint.

We define ternary relations $\triangleleft$ ('split') and $\triangleright$ ('merge') on models as follows:

$$\mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix} \text{ iff } \mathsf{m}_1 \text{ and } \mathsf{m}_2 \text{ are RW-disjoint, } \mathsf{Rd} = \mathsf{Rd}_1 \cup \mathsf{Rd}_2,\ \mathsf{Wr} = \mathsf{Wr}_1 \cup \mathsf{Wr}_2, \text{ and } \mathsf{V} = \mathsf{V}_1 = \mathsf{V}_2;$$

$$\begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix} \triangleright \mathsf{m} \text{ iff } \mathsf{m}_1 \text{ and } \mathsf{m}_2 \text{ are RW-disjoint, } \mathsf{Rd} = \mathsf{Rd}_1 \cup \mathsf{Rd}_2,\ \mathsf{Wr} = \mathsf{Wr}_1 \cup \mathsf{Wr}_2,\ \mathsf{V}_1 \setminus \mathsf{Wr} = \mathsf{V}_2 \setminus \mathsf{Wr}, \text{ and } \mathsf{V} = (\mathsf{V}_1 \cap \mathsf{Wr}_1) \cup (\mathsf{V}_2 \cap \mathsf{Wr}_2) \cup (\mathsf{V}_1 \cap \mathsf{V}_2).$$

For example, for $\mathsf{m} = \langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle$ and $\mathsf{m}_2 = \langle \mathsf{Rd}_2, \mathsf{Wr}_2, \mathsf{V}_2\rangle$ we have $\mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}\\ \mathsf{m}_2\end{smallmatrix}$ if and only if $\mathsf{Wr}_2 = \emptyset$, $\mathsf{V}_2 = \mathsf{V}$, and $\mathsf{Rd}_2 \subseteq \mathsf{Rd} \setminus \mathsf{Wr}$. In particular, $\langle\emptyset,\emptyset,\emptyset\rangle \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix}$ if and only if $\mathsf{m}_1 = \mathsf{m}_2 = \langle\emptyset,\emptyset,\emptyset\rangle$. Contrarily to splitting, merging does not keep the valuation constant: it only keeps constant the non-modifiable part $\mathsf{V} \setminus \mathsf{Wr}$ of the valuation V and puts the results of the allowed modifications of Wr together. These modifications cannot conflict because m_1 and m_2 are RW-disjoint. Figure 1 illustrates each of these two operations by an example. The checks that are performed in the merge operation are reminiscent of the self composition technique in the analysis of secure information flows [DHS05,SG16].[7]

[7] We are grateful to Rainer Hähnle for pointing this out to us.

The set Rd of readable variables of a model m determines which models cannot be distinguished from m:

$$\mathsf{m} \sim \mathsf{m}' \text{ iff } \mathsf{Rd} = \mathsf{Rd}', \mathsf{Wr} = \mathsf{Wr}', \mathsf{V} \cap \mathsf{Rd} = \mathsf{V}' \cap \mathsf{Rd}'.$$

Hence m and m' are indistinguishable if (1) they have the same readable and writable variables and (2) the valuations are identical as far as their readable parts are concerned. This relation will serve to interpret tests: the test $\varphi?$ of a formula φ is conditioned by its truth in all read-indistinguishable models, i.e., in all models where the readable variables have the same truth value.

3 Language

Formulas and programs are defined by the following grammar, where p ranges over the set of propositional variables $\mathbb{P}$:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi,$$
$$\pi ::= +p \mid -p \mid \mathtt{r}{+}p \mid \mathtt{r}{-}p \mid \mathtt{w}{+}p \mid \mathtt{w}{-}p \mid \varphi? \mid \varphi\overset{?}{?} \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \pi\|\pi.$$

The program $+p$ makes p true and $-p$ makes p false, where the executability of these two programs is conditioned by the writability of p. The program $\mathtt{r}{+}p$ makes p readable and $\mathtt{r}{-}p$ makes p unreadable; similarly, $\mathtt{w}{+}p$ makes p writable and $\mathtt{w}{-}p$ makes p non-writable. We suppose that these four programs are always executable. The program $\varphi?$ is the PDL test that φ, that we call *exogenous*; $\varphi\overset{?}{?}$ is the *endogenous* test that φ: it is conditioned by the readability of the relevant variables of φ.

The formula $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$. Given an integer $n \geq 0$, the program π^n is defined inductively by $\pi^0 = \top?$ and $\pi^{n+1} = \pi;\pi^n$. Similarly, $\pi^{\leq n}$ is defined by $\pi^{\leq 0} = \top?$ and $\pi^{\leq n+1} = \top? \cup (\pi;\pi^{\leq n})$. For a finite set of variables $P = \{p_i\}_{1\leq i\leq n}$ and associated programs $\{\pi_i(p_i)\}_{1\leq i\leq n}$, we use the notation $\fatsemi_{p\in P}\, \pi(p)$ to denote the sequence $\pi_1(p_1);\cdots;\pi_n(p_n)$, in some order. We will make use of this notation with care to guarantee that the ordering of the elements of P does not matter.

The set of propositional variables occurring in a formula φ is noted $\mathbb{P}_\varphi$ and the set of those occurring in a program π is noted $\mathbb{P}_\pi$. For example, $\mathbb{P}_{p\vee\langle+q\rangle\neg r} = \{p, q, r\}$.

4 Semantics

Let $\mathsf{m} = \langle\mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle$ be a model. Formulas are interpreted as sets of models:

$\mathsf{m} \models \top$;
$\mathsf{m} \models p$ iff $p \in \mathsf{V}$, for $p \in \mathbb{P}$;
$\mathsf{m} \models \neg\varphi$ iff $\mathsf{m} \not\models \varphi$;
$\mathsf{m} \models \varphi \vee \psi$ iff $\mathsf{m} \models \varphi$ or $\mathsf{m} \models \psi$;
$\mathsf{m} \models \langle\pi\rangle\varphi$ iff there is a model m' such that $\mathsf{m}[\![\pi]\!]\mathsf{m}'$ and $\mathsf{m}' \models \varphi$.

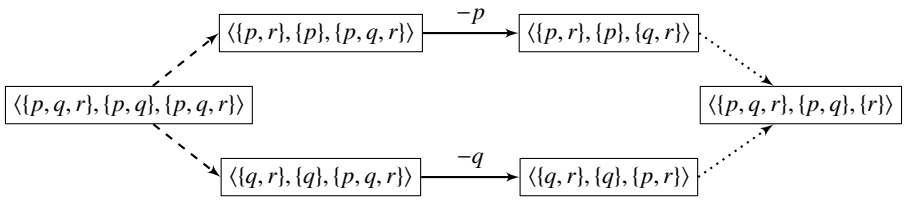

Fig. 2. Illustration of an execution of $-p\|-q$ at the model $\langle\{p,q,r\},\{p,q\},\{p,q,r\}\rangle$.

Programs are interpreted as relations on the set of models:

$$
\begin{array}{ll}
\mathsf{m}[\![+p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd},\ \mathsf{Wr}' = \mathsf{Wr},\ \mathsf{V}' = \mathsf{V} \cup \{p\}, \text{ and } p \in \mathsf{Wr} \\
\mathsf{m}[\![-p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd},\ \mathsf{Wr}' = \mathsf{Wr},\ \mathsf{V}' = \mathsf{V} \setminus \{p\}, \text{ and } p \in \mathsf{Wr} \\
\mathsf{m}[\![\mathtt{r}{+}p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd} \cup \{p\},\ \mathsf{Wr}' = \mathsf{Wr}, \text{ and } \mathsf{V}' = \mathsf{V} \\
\mathsf{m}[\![\mathtt{r}{-}p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd} \setminus \{p\},\ \mathsf{Wr}' = \mathsf{Wr} \setminus \{p\}, \text{ and } \mathsf{V}' = \mathsf{V} \\
\mathsf{m}[\![\mathtt{w}{+}p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd} \cup \{p\},\ \mathsf{Wr}' = \mathsf{Wr} \cup \{p\}, \text{ and } \mathsf{V}' = \mathsf{V} \\
\mathsf{m}[\![\mathtt{w}{-}p]\!]\mathsf{m}' & \text{iff } \mathsf{Rd}' = \mathsf{Rd},\ \mathsf{Wr}' = \mathsf{Wr} \setminus \{p\}, \text{ and } \mathsf{V}' = \mathsf{V} \\
\mathsf{m}[\![\varphi?]\!]\mathsf{m}' & \text{iff } \mathsf{m} = \mathsf{m}' \text{ and } \mathsf{m} \models \varphi \\
\mathsf{m}[\![\varphi\mathbin{?\!?}]\!]\mathsf{m}' & \text{iff } \mathsf{m} = \mathsf{m}' \text{ and } \mathsf{m}'' \models \varphi \text{ for every } \mathsf{m}'' \text{ such that } \mathsf{m}'' \sim \mathsf{m} \\
\mathsf{m}[\![\pi_1;\pi_2]\!]\mathsf{m}' & \text{iff there is an } \mathsf{m}'' \text{ such that } \mathsf{m}[\![\pi_1]\!]\mathsf{m}'' \text{ and } \mathsf{m}''[\![\pi_2]\!]\mathsf{m}' \\
\mathsf{m}[\![\pi_1 \cup \pi_2]\!]\mathsf{m}' & \text{iff } \mathsf{m}[\![\pi_1]\!]\mathsf{m}' \text{ or } \mathsf{m}[\![\pi_2]\!]\mathsf{m}' \\
\mathsf{m}[\![\pi^*]\!]\mathsf{m}' & \text{iff there is an } n \geq 0 \text{ such that } \mathsf{m}[\![\pi]\!]^n\mathsf{m}' \\
\mathsf{m}[\![\pi_1\|\pi_2]\!]\mathsf{m}' & \text{iff there are } \mathsf{m}_1, \mathsf{m}_2, \mathsf{m}'_1, \mathsf{m}'_2 \text{ such that } \mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix}, \begin{smallmatrix}\mathsf{m}'_1\\ \mathsf{m}'_2\end{smallmatrix} \triangleright \mathsf{m}', \\
 & \quad \mathsf{m}_1[\![\pi_1]\!]\mathsf{m}'_1,\ \mathsf{Rd}_1 = \mathsf{Rd}'_1,\ \mathsf{Wr}_1 = \mathsf{Wr}'_1,\ \mathsf{V}_1 \setminus \mathsf{Wr}_1 = \mathsf{V}'_1 \setminus \mathsf{Wr}'_1, \\
 & \quad \mathsf{m}_2[\![\pi_2]\!]\mathsf{m}'_2,\ \mathsf{Rd}_2 = \mathsf{Rd}'_2,\ \mathsf{Wr}_2 = \mathsf{Wr}'_2,\ \mathsf{V}_2 \setminus \mathsf{Wr}_2 = \mathsf{V}'_2 \setminus \mathsf{Wr}'_2
\end{array}
$$

In the interpretation of assignments of atomic formulas we require propositional variables to be modifiable, while readability and writability can be modified unconditionally. When a variable is made writable then it is made readable, too, in order to guarantee the inclusion constraint on models; similarly when a variable is made unreadable. The interpretation of parallel composition $\pi_1\|\pi_2$ is such that both π_1 and π_2 only modify 'their' variables. More precisely, parallel composition $\pi_1\|\pi_2$ of two programs π_1 and π_2 relates two models m and m' when the following conditions are satisfied: (1) m can be split into m_1 and m_2; (2) the execution of π_1 on m_1 may lead to m'_1 and the execution of π_2 on m_2 may lead to m'_2; (3) m'_1 and m'_2 can be merged into m'. Moreover, (4) the modifications are legal: π_1 and π_2 neither change readability nor writability, and each of them only modifies variables that were allocated to it by the split.

Figure 2 illustrates the interpretation of the parallel program $-p\|-q$. Some more examples follow.

Example 1. Suppose $\mathsf{m} = \langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle$ with $\mathsf{Wr} = \mathsf{Rd} = \mathsf{V} = \{p,q,r\}$. Then $\mathsf{m}' = \langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}'\rangle$ with $\mathsf{V}' = \{p,r\}$ is the only model such that $\mathsf{m}[\![+p\|-q]\!]\mathsf{m}'$.

The next example illustrates the last condition in the interpretation of parallel composition.

Example 2. The programs $+p\|+p$ and $+p\|(\mathtt{w}{+}p; {-}p; \mathtt{r}{-}p)$ cannot be executed on the model $\mathsf{m} = \langle\{p\},\{p\},\{p\}\rangle$. For the second, suppose there are m_1 and m_2 such that $\mathsf{m} \triangleleft^{\mathsf{m}_1}_{\mathsf{m}_2}$ and suppose $+p$ is executed on m_1 and $\mathtt{w}{+}p; {-}p; \mathtt{r}{-}p$ on m_2. For $+p$ to be executable we must have $p \in \mathsf{Wr}_1$, and therefore $p \notin \mathsf{Wr}_2$ (and a fortiori $p \notin \mathsf{Rd}_2$) because of the RW-disjointness condition. Hence $\mathsf{m}_1 = \langle\{p\},\{p\},\{p\}\rangle$ and $\mathsf{m}_2 = \langle\emptyset,\emptyset,\{p\}\rangle$. Then $\mathsf{m}'_1 = \mathsf{m}_1$ is the only model such that $\mathsf{m}_1[\![+p]\!]\mathsf{m}'_1$; and $\mathsf{m}'_2 = \langle\emptyset,\emptyset,\emptyset\rangle$ is the only model such that $\mathsf{m}_2[\![\mathtt{w}{+}p; {-}p; \mathtt{r}{-}p]\!]\mathsf{m}'_2$. These two models cannot be merged because $\mathsf{V}'_2 \setminus \mathsf{Wr}'_2 = \emptyset$ fails to be equal to $\mathsf{V}_2 \setminus \mathsf{Wr}_2 = \{p\}$. However, $+p\|(\mathtt{w}{+}p; {+}p; \mathtt{r}{-}p)$ is executable on m: we have $\mathsf{m} \triangleleft^{\mathsf{m}}_{\langle\emptyset,\emptyset,\{p\}\rangle}$ and $\mathsf{m}[\![+p]\!]\langle\{p\},\{p\},\{p\}\rangle$ and $\langle\emptyset,\emptyset,\{p\}\rangle[\![\mathtt{w}{+}p; {+}p; \mathtt{r}{-}p]\!]\langle\emptyset,\emptyset,\{p\}\rangle$ and $\triangleright^{\langle\emptyset,\emptyset,\{p\}\rangle}_{\langle\{p\},\{p\},\{p\}\rangle} \langle\{p\},\{p\},\{p\}\rangle$.

It is clear that parallel composition satisfies commutativity. It is less obvious that it is also associative. The proof is somewhat involved and can be found in the annex.

Let us finally illustrate the different semantics of the two test operators of our logic.

Example 3. Suppose m is such that $p \notin \mathsf{Rd}$ and $p \in \mathsf{V}$. Then the program $p?$ is executable on m because $p \in \mathsf{V}$. In contrast, there is no m' such that $\mathsf{m}[\![p\mathbf{?}]\!]\mathsf{m}'$, the reason being that there is always an m'' such that $\mathsf{m} \sim \mathsf{m}''$ and $p \notin \mathsf{V}''$, hence $p\mathbf{?}$ is inexecutable.

In practice, parallel programs should only contain endogenous tests in order to avoid that a subprogram accesses the truth value of a variable that is not among its readable variables. Actually we have kept PDL tests for technical reasons only: we could not formulate some of the reduction axioms without them.

Satisfiability and validity of formulas are defined in the expected way.

Example 4. The formulas $\varphi \to [\top\mathbf{?}\|\top\mathbf{?}]\varphi$, $[+p\|{-}p]\bot$, $[+p\|{+}p]\bot$ and $[p\mathbf{?}\|{+}p]\bot$ whose parallel programs were discussed in the introduction are all valid.

The formulas $\langle{+}p\rangle\top$ and $\langle{-}p\rangle\top$ both express that p is writable. Moreover, $\langle p\mathbf{?}\rangle\top$ expresses that p is true and readable, and $\langle\neg p\mathbf{?}\rangle\top$ expresses that p is false and readable; therefore $\langle p\mathbf{?}\rangle\top \vee \langle\neg p\mathbf{?}\rangle\top$ expresses that p is readable. This will be instrumental in our axiomatisation.

Finally, for any model $\mathsf{m} = \langle\mathsf{Rd},\mathsf{Wr},\mathsf{V}\rangle$ and $P \subseteq \mathbb{P}$, we write $\mathsf{m} \cap P$ for the model $\langle\mathsf{Rd} \cap P, \mathsf{Wr} \cap P, \mathsf{V} \cap P\rangle$. This notation along with the following standard lemma will be used several times in the remainder of this work.

Lemma 1. *If there is $P \subseteq \mathbb{P}$ such that $\mathsf{m}_1 \cap P = \mathsf{m}'_1 \cap P$, $\mathbb{P}_\varphi \subseteq P$ and $\mathbb{P}_\pi \subseteq P$ then*

1. *$\mathsf{m}_1 \models \varphi$ implies $\mathsf{m}'_1 \models \varphi$; and*
2. *$\mathsf{m}_1[\![\pi]\!]\mathsf{m}_2$ implies $\mathsf{m}'_1[\![\pi]\!]\mathsf{m}'_2$, for m'_2 such that $\mathsf{m}'_2 \cap P = \mathsf{m}_2 \cap P$ and $\mathsf{m}'_2 \cap (\mathbb{P} \setminus P) = \mathsf{m}'_1 \cap (\mathbb{P} \setminus P)$.*

Proof (sketch). The proof is by a straightforward simultaneous induction on the size of φ and π.

5 Axiomatisation via Reduction Axioms

We axiomatise the validities of our logic by means of reduction axioms, as customary in dynamic epistemic logics [vDvdHK07]. These axioms transform every formula into a boolean combination of propositional variables and formulas of the form $\langle +p\rangle\top$ and $\langle p?\rangle\top \vee \langle \neg p?\rangle\top$. The former expresses that p is writable: we abbreviate it by $\mathtt{w}_p$; the latter expresses that p is readable: we abbreviate it by $\mathtt{r}_p$. Hence we have:

$$\mathtt{w}_p \stackrel{\text{def}}{=} \langle +p\rangle\top$$
$$\mathtt{r}_p \stackrel{\text{def}}{=} \langle p?\rangle\top \vee \langle \neg p?\rangle\top$$

The reduction starts by eliminating all the program operators from formulas, where the elimination of parallel composition is done by sequentialising it while keeping track of the values of the atoms. After that step, the only remaining program operators either occur in formulas of the form $\mathtt{r}_p$ or $\mathtt{w}_p$, or in modal operators of the form $\langle +p\rangle$, $\langle -p\rangle$, $\langle \mathtt{r}{+}p\rangle$, $\langle \mathtt{r}{-}p\rangle$, $\langle \mathtt{w}{+}p\rangle$, or $\langle \mathtt{w}{-}p\rangle$. All these modal operators can be distributed over the boolean operators, taking advantage of the fact that all of them are deterministic modal operators (validating the Alt_1 axiom $\langle\pi\rangle\varphi \to [\pi]\varphi$). Finally, sequences of such modalities facing a propositional variable can be transformed into boolean combinations of readability and writability statements $\mathtt{r}_p$ and $\mathtt{w}_p$. The only logical link between these statements is that writability of p implies readability of p. This is captured by the axiom schema $\mathtt{w}_p \to \mathtt{r}_p$.

The sequentialisation of parallel composition uses copies of variables, so we start by introducing that notion. We then define some programs and formulas that will allow us to formulate the reduction axioms more concisely.

5.1 Copies of Atomic Propositions

Our reduction axioms will introduce fresh copies of each propositional variable, one per occurrence of the parallel composition operator. The interpretation of parallel composition being based on separation, each concurrent program operates on its own model. The copies emulate the separation of models: each concurrent program is executed on a set of copies of propositional variables.

In order to keep things readable we neglect that the copies should be indexed by programs and denote the copies of the variable p by $p^{\mathbf{k}}$, where $\mathbf{k}$ is some integer. In principle we should introduce a bijection between the indexes $\mathbf{k}$ and the subprogram they are attached to; we however do not do so to avoid overly complicated notations.

Given a set of propositional variables $P \subseteq \mathbb{P}$ and an integer $\mathbf{k} \in \{1, 2\}$, we define the set of copies $P^{\mathbf{k}} = \{p^{\mathbf{k}} \ : \ p \in P\}$. Similarly, we define copies of programs and formulas: the program $\pi^{\mathbf{k}}$ and the formula $\varphi^{\mathbf{k}}$ are obtained by replacing all their occurrences of propositional variables p by $p^{\mathbf{k}}$. For example, $(+p; q?)^{\mathbf{k}}$ equals $+p^{\mathbf{k}}; q^{\mathbf{k}}?$.

The following lemma will be instrumental in the soundness proof.

Lemma 2. *Let π be a program, let $\langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle$ be a model, and let $\mathbf{k} \in \{1, 2\}$. Then*

$$\langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle[\![\pi]\!]\langle \mathsf{Rd}', \mathsf{Wr}', \mathsf{V}'\rangle \textit{ iff } \langle \mathsf{Rd}^{\mathbf{k}}, \mathsf{Wr}^{\mathbf{k}}, \mathsf{V}^{\mathbf{k}}\rangle[\![\pi^{\mathbf{k}}]\!]\langle \mathsf{Rd}'^{\mathbf{k}}, \mathsf{Wr}'^{\mathbf{k}}, \mathsf{V}'^{\mathbf{k}}\rangle.$$

$$
\begin{aligned}
\mathsf{split}(P) = \;& ;_{p\in P} \Big(\mathtt{w}{+}p^{\mathbf{1}}; \mathtt{w}{+}p^{\mathbf{2}}; \big((p?; {+}p^{\mathbf{1}}; {+}p^{\mathbf{2}}) \cup (\neg p?; {-}p^{\mathbf{1}}; {-}p^{\mathbf{2}})\big); \mathtt{r}{-}p^{\mathbf{1}}; \mathtt{r}{-}p^{\mathbf{2}}; \\
& \big(\neg \mathtt{r}_p? \cup (\mathtt{w}_p?; (\mathtt{w}{+}p^{\mathbf{1}} \cup \mathtt{w}{+}p^{\mathbf{2}})) \cup \\
& (\neg \mathtt{w}_p \wedge \mathtt{r}_p?; (\mathtt{r}{+}p^{\mathbf{1}} \cup \mathtt{r}{+}p^{\mathbf{2}} \cup (\mathtt{r}{+}p^{\mathbf{1}}; \mathtt{r}{+}p^{\mathbf{2}})))\big)\Big) \\
\mathsf{store}(P) = \;& ;_{p\in P} ;_{k\in\{1,2\}} \big(\mathtt{w}{+}p^{\mathbf{k}}_R; \mathtt{w}{+}p^{\mathbf{k}}_W; \\
& ((\mathtt{r}_{p^{\mathbf{k}}}?; {+}p^{\mathbf{k}}_R) \cup (\neg \mathtt{r}_{p^{\mathbf{k}}}?; {-}p^{\mathbf{k}}_R)); ((\mathtt{w}_{p^{\mathbf{k}}}?; {+}p^{\mathbf{k}}_W) \cup (\neg \mathtt{w}_{p^{\mathbf{k}}}?; {-}p^{\mathbf{k}}_W))\big) \\
\mathsf{check}(P) = \;& \bigwedge_{p\in P} \bigwedge_{k\in\{1,2\}} \big((p^{\mathbf{k}}_R \leftrightarrow \mathtt{r}_{p^{\mathbf{k}}}) \wedge (p^{\mathbf{k}}_W \leftrightarrow \mathtt{w}_{p^{\mathbf{k}}}) \wedge (\neg p^{\mathbf{k}}_W \rightarrow (p \leftrightarrow p^{\mathbf{k}}))\big) \\
\mathsf{merge}(P) = \;& ;_{p\in P} \big((\neg \mathtt{w}_p? \cup \\
& ((\mathtt{w}_{p^{\mathbf{1}}} \wedge p^{\mathbf{1}}) \vee (\mathtt{w}_{p^{\mathbf{2}}} \wedge p^{\mathbf{2}})?; {+}p) \cup \\
& ((\mathtt{w}_{p^{\mathbf{1}}} \wedge \neg p^{\mathbf{1}}) \vee (\mathtt{w}_{p^{\mathbf{2}}} \wedge \neg p^{\mathbf{2}})?; {-}p)); \\
& ;_{k\in\{1,2\}} ({-}p^{\mathbf{k}}_R; {-}p^{\mathbf{k}}_W; \mathtt{w}{+}p^{\mathbf{k}}; {-}p^{\mathbf{k}}; \mathtt{r}{-}p^{\mathbf{k}}_R; \mathtt{r}{-}p^{\mathbf{k}}_W; \mathtt{r}{-}p^{\mathbf{k}})\big) \\
\mathsf{flatten}(\pi_1, \pi_2) = \;& \mathsf{split}\left(\mathbb{P}_{\pi_1\|\pi_2}\right); \mathsf{store}\left(\mathbb{P}_{\pi_1\|\pi_2}\right); {\pi_1}^{\mathbf{1}}; {\pi_2}^{\mathbf{2}}; \mathsf{check}\left(\mathbb{P}_{\pi_1\|\pi_2}\right)?; \mathsf{merge}\left(\mathbb{P}_{\pi_1\|\pi_2}\right)
\end{aligned}
$$

Table 1. Useful programs and formulas, for all finite $P \subseteq \mathbb{P}$ and all programs π_1 and π_2.

Proof (sketch). Just as for Lemma 1, the proof is by simultaneous induction on the form of programs and formulas, where the induction hypothesis for the latter is that $\langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle \models \varphi$ if and only if $\langle \mathsf{Rd}^{\mathbf{k}}, \mathsf{Wr}^{\mathbf{k}}, \mathsf{V}^{\mathbf{k}}\rangle \models \varphi^{\mathbf{k}}$. It relies on the fact that DL-PA$^{\|}$ satisfies the substitution rule.

Moreover, in order to simulate the semantics of the parallel composition operator our reduction axioms associate to each copy $p^{\mathbf{k}}$ two fresh variables $p^{\mathbf{k}}_R$ and $p^{\mathbf{k}}_W$, denoting whether $p^{\mathbf{k}}$ was respectively readable and writable just after the split. Given a set of propositional variables P, we define

$$\mathsf{St}(P) = \{p^{\mathbf{k}}_R \;:\; k \in \{1,2\}, p \in P\} \cup \{p^{\mathbf{k}}_W \;:\; k \in \{1,2\}, p \in P\}$$

as the set of all these fresh variables.

5.2 Useful Programs and formulas

Let $P \subseteq \mathbb{P}$ be some finite set of propositional variables. Table 1 lists programs and formulas that will be useful to concisely formulate the reduction axioms. Observe that the order of the variables in the sequential compositions $;_{p\in P}(\cdots)$ occurring in the above programs does not matter. Observe also that the only endogenous tests on the right hand side occur in readability statements $\mathtt{r}_p$. (Remember that $\mathtt{r}_p$ abbreviates $\langle p?\rangle\top \vee \langle \neg p?\rangle\top$.)

The $\mathsf{split}(P)$ program simulates the split operation by (1) assigning the truth value of every $p \in P$ to its copies $p^{\mathbf{1}}$ and $p^{\mathbf{2}}$ and (2) non-deterministically assigning two copies of each read and write variable in a way such that a counterpart of the RW-disjointness condition $\mathsf{Wr}_1 \cap \mathsf{Rd}_2 = \mathsf{Wr}_2 \cap \mathsf{Rd}_1 = \emptyset$ is guaranteed. Note that the assignments $\mathtt{w}{+}p^{\mathbf{k}}$ also

make $p^{\mathbf{k}}$ readable. The following lemma formally states the main property of $\mathsf{split}(P)$. It can easily be proved by following the previous observations.

Lemma 3. *For all $P \subseteq \mathbb{P}$, and all models m, m_1 and m_2 such that $\mathsf{m} \cap P = \mathsf{m}$,*

$$\mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix} \textit{ if and only if } \mathsf{m}[\![\mathsf{split}(P)]\!]\mathsf{m}'$$

with $\mathsf{Rd}' = \mathsf{Rd} \cup \mathsf{Rd}_1{}^{\mathbf{1}} \cup \mathsf{Rd}_2{}^{\mathbf{2}}$, $\mathsf{Wr}' = \mathsf{Wr} \cup \mathsf{Wr}_1{}^{\mathbf{1}} \cup \mathsf{Wr}_2{}^{\mathbf{2}}$, *and* $\mathsf{V}' = \mathsf{V} \cup \mathsf{V}_1{}^{\mathbf{1}} \cup \mathsf{V}_2{}^{\mathbf{2}}$.

The $\mathsf{store}(P)$ program stores the readability and writability states of the copies of the propositional variable into some fresh variables. These variables are then used only in $\mathsf{check}(P)$.

The formula $\mathsf{check}(P)$ compares the current state with the state just after the split. It is true if and only if (1) readability values are identical, (2) writability values are identical, and (3) truth values are identical for non-writable variables.

The $\mathsf{merge}(P)$ program simulates the merge operation by reinstating all those read- and write-atoms that had been allocated to the first subprogram in the sequentialisation. The following lemma formally states the main property of $\mathsf{merge}(P)$. This lemma is weaker than Lemma 3 for $\mathsf{split}(P)$. The additional hypotheses are guaranteed to hold by the interplay of programs $\mathsf{split}(P)$ and $\mathsf{store}(P)$, and formula $\mathsf{check}(P)$.

Lemma 4. *Let m, m_1 and m_2 be models, and P a set of propositional variables such that $\mathsf{m} \cap P = \mathsf{m}$, m_1 and m_2 are RW-disjoint, $\mathsf{Rd} = \mathsf{Rd}_1 \cup \mathsf{Rd}_2$, $\mathsf{Wr} = \mathsf{Wr}_1 \cup \mathsf{Wr}_2$, and $\mathsf{V}_1 \setminus \mathsf{Wr} = \mathsf{V}_2 \setminus \mathsf{Wr}$. Then*

$$\begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix} \triangleright \mathsf{m} \textit{ if and only if } \mathsf{m}'[\![\mathsf{merge}(P)]\!]\mathsf{m}$$

with $\mathsf{Rd}' = \mathsf{Rd} \cup \mathsf{Rd}_1{}^{\mathbf{1}} \cup \mathsf{Rd}_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, $\mathsf{Wr}' = \mathsf{Wr} \cup \mathsf{Wr}_1{}^{\mathbf{1}} \cup \mathsf{Wr}_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, *and* $\mathsf{V}' = \mathsf{V}^\sharp \cup \mathsf{V}_1{}^{\mathbf{1}} \cup \mathsf{V}_2{}^{\mathbf{2}}$ *where $\mathsf{V}^\sharp$ is any subset of $P \cup \mathsf{St}(P)$ such that $\mathsf{V}^\sharp \setminus \mathsf{Wr} = \mathsf{V} \setminus \mathsf{Wr}$.*

Proof (sketch). It suffices to prove that $\mathsf{m}'[\![\mathsf{merge}(P)]\!]\mathsf{m}$ if and only if $\mathsf{V} = (\mathsf{V}_1 \cap \mathsf{Wr}_1) \cup (\mathsf{V}_2 \cap \mathsf{Wr}_2) \cup (\mathsf{V}_1 \cap \mathsf{V}_2)$, which is straightforward.

Finally, the $\mathsf{flatten}(\pi_1, \pi_2)$ program emulates the execution of the program $\pi_1 \| \pi_2$ as a sequential composition of the previous programs. Notice that there is no occurrence in $\mathsf{flatten}(\pi_1, \pi_2)$ of the parallel composition operator, except possibly inside π_1 and π_2. Lemma 6 below states that the emulation is faithful. We first need Lemma 5, which can be seen as an adaptation of Lemma 1 to $\mathsf{flatten}(P)$.

Lemma 5. *For all models m and m', and all programs π_1 and π_2,*

$$\mathsf{m}[\![\mathsf{flatten}(\pi_1, \pi_2)]\!]\mathsf{m}' \textit{ iff } (\mathsf{m} \cap \mathbb{P}_{\pi_1 \| \pi_2})[\![\mathsf{flatten}(\pi_1, \pi_2)]\!](\mathsf{m}' \cap \mathbb{P}_{\pi_1 \| \pi_2}).$$

Proof (sketch). It suffices to observe that all variables in $\mathbb{P}_{\mathsf{flatten}(\pi_1, \pi_2)} \setminus \mathbb{P}_{\pi_1 \| \pi_2}$ are (1) made writable and initialized by split or store, and (2) set to false and made unreadable by merge.

We can now prove our main lemma.

Lemma 6. *For all models* m *and* m'*, and all programs* π_1 *and* π_2*,*

$$\mathsf{m}[\![\pi_1 || \pi_2]\!]\mathsf{m}' \text{ if and only if } \mathsf{m}[\![\mathsf{flatten}(\pi_1, \pi_2)]\!]\mathsf{m}'.$$

Proof (sketch). Let $P = \mathbb{P}_{\pi_1||\pi_2}$. By Lemmas 1 and 5, we can assume that $\mathsf{m} \cap P = \mathsf{m}$ and $\mathsf{m}' \cap P = \mathsf{m}'$.

For the left-to-right direction, suppose there are models $\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}'_1$ and m'_2 such that $\mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix}, \begin{smallmatrix}\mathsf{m}'_1\\ \mathsf{m}'_2\end{smallmatrix} \triangleright \mathsf{m}'$, $\mathsf{m}_1[\![\pi_1]\!]\mathsf{m}'_1$, $\mathsf{m}_2[\![\pi_2]\!]\mathsf{m}'_2$, $\mathsf{Rd}_1 = \mathsf{Rd}'_1$, $\mathsf{Wr}_1 = \mathsf{Wr}'_1$, $\mathsf{V}_1 \setminus \mathsf{Wr}_1 = \mathsf{V}'_1 \setminus \mathsf{Wr}'_1$, $\mathsf{Rd}_2 = \mathsf{Rd}'_2$, $\mathsf{Wr}_2 = \mathsf{Wr}'_2$, and $\mathsf{V}_2 \setminus \mathsf{Wr}_2 = \mathsf{V}'_2 \setminus \mathsf{Wr}'_2$. The following statements can be proved:

1. $\mathsf{m}[\![\mathsf{split}(P)]\!]\mathsf{m}^+_1$ with $\mathsf{Rd}^+_1 = \mathsf{Rd} \cup \mathsf{Rd}_1{}^{\mathbf{1}} \cup \mathsf{Rd}_2{}^{\mathbf{2}}$, $\mathsf{Wr}^+_1 = \mathsf{Wr} \cup \mathsf{Wr}_1{}^{\mathbf{1}} \cup \mathsf{Wr}_2{}^{\mathbf{2}}$, and $\mathsf{V}^+_1 = \mathsf{V} \cup \mathsf{V}_1{}^{\mathbf{1}} \cup \mathsf{V}_2{}^{\mathbf{2}}$. The proof relies on Lemma 3.
2. $\mathsf{m}^+_1[\![\mathsf{store}(P)]\!]\mathsf{m}^+_2$ with $\mathsf{Rd}^+_2 = \mathsf{Rd}^+_1 \cup \mathsf{St}(P)$, $\mathsf{Wr}^+_2 = \mathsf{Wr}^+_1 \cup \mathsf{St}(P)$, $\mathsf{V}^+_2 = \mathsf{V}^+_1 \cup \mathsf{V}_{\mathsf{St}}$, and $\mathsf{V}_{\mathsf{St}} = \{p^{\mathbf{k}}_R \ : \ p^{\mathbf{k}} \in \mathsf{Rd}^+_1\} \cup \{p^{\mathbf{k}}_W \ : \ p^{\mathbf{k}} \in \mathsf{Wr}^+_1\}$. Notice that $\mathsf{V}_{\mathsf{St}} = \{p^{\mathbf{k}}_R \ : \ p \in \mathsf{Rd}_k\} \cup \{p^{\mathbf{k}}_W \ : \ p \in \mathsf{Wr}_k\}$.
3. $\mathsf{m}^+_2[\![\pi_1{}^{\mathbf{1}}]\!]\mathsf{m}^+_3$ with $\mathsf{Rd}^+_3 = \mathsf{Rd} \cup \mathsf{Rd}'_1{}^{\mathbf{1}} \cup \mathsf{Rd}_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, $\mathsf{Wr}^+_3 = \mathsf{Wr} \cup \mathsf{Wr}'_1{}^{\mathbf{1}} \cup \mathsf{Wr}_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, and $\mathsf{V}^+_3 = \mathsf{V} \cup \mathsf{V}'_1{}^{\mathbf{1}} \cup \mathsf{V}_2{}^{\mathbf{2}} \cup \mathsf{V}_{\mathsf{St}}$. The proof relies on Lemmas 1 and 2.
4. $\mathsf{m}^+_3[\![\pi_2{}^{\mathbf{2}}]\!]\mathsf{m}^+_4$ with $\mathsf{Rd}^+_4 = \mathsf{Rd} \cup \mathsf{Rd}'_1{}^{\mathbf{1}} \cup \mathsf{Rd}'_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, $\mathsf{Wr}^+_4 = \mathsf{Wr} \cup \mathsf{Wr}'_1{}^{\mathbf{1}} \cup \mathsf{Wr}'_2{}^{\mathbf{2}} \cup \mathsf{St}(P)$, and $\mathsf{V}^+_4 = \mathsf{V} \cup \mathsf{V}'_1{}^{\mathbf{1}} \cup \mathsf{V}'_2{}^{\mathbf{2}} \cup \mathsf{V}_{\mathsf{St}}$.
5. $\mathsf{m}^+_4 \models \mathsf{check}(P)$.
6. $\mathsf{m}^+_4[\![\mathsf{merge}(P)]\!]\mathsf{m}'$. The proof relies on Lemma 4.
7. $\mathsf{m}[\![\mathsf{flatten}(\pi_1, \pi_2)]\!]\mathsf{m}'$.

For the right-to-left direction, let us suppose that there are models m^+_1, m^+_2, m^+_3 and m^+_4 such that $\mathsf{m}[\![\mathsf{split}(P)]\!]\mathsf{m}^+_1[\![\mathsf{store}(P)]\!]\mathsf{m}^+_2[\![\pi_1]\!]\mathsf{m}^+_3[\![\pi_2; \mathsf{check}(P)?]\!]\mathsf{m}^+_4[\![\mathsf{merge}(P)]\!]\mathsf{m}'$. The following statements can be proved:

1. $\mathsf{m} \triangleleft \begin{smallmatrix}\mathsf{m}_1\\ \mathsf{m}_2\end{smallmatrix}$ with $\mathsf{Rd}_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{Rd}^+_1\}$, $\mathsf{Wr}_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{Wr}^+_1\}$, $\mathsf{V}_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{V}^+_1\}$, $\mathsf{Rd}_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{Rd}^+_1\}$, $\mathsf{Wr}_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{Wr}^+_1\}$, and $\mathsf{V}_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{V}^+_1\}$. The proof relies on Lemma 3.
2. $\mathsf{m}_1[\![\pi_1]\!]\mathsf{m}'_1$ with $\mathsf{Rd}'_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{Rd}^+_3\}$, $\mathsf{Wr}'_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{Wr}^+_3\}$, and $\mathsf{V}'_1 = \{p \ : \ p^{\mathbf{1}} \in \mathsf{V}^+_3\}$. The proof relies on Lemmas 1 and 2.
3. $\mathsf{m}_2[\![\pi_2]\!]\mathsf{m}'_2$ with $\mathsf{Rd}'_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{Rd}^+_4\}$, $\mathsf{Wr}'_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{Wr}^+_4\}$, and $\mathsf{V}'_2 = \{p \ : \ p^{\mathbf{2}} \in \mathsf{V}^+_4\}$.
4. $\mathsf{Rd}_1 = \mathsf{Rd}'_1$, $\mathsf{Wr}_1 = \mathsf{Wr}'_1$, $\mathsf{V}_1 \setminus \mathsf{Wr}_1 = \mathsf{V}'_1 \setminus \mathsf{Wr}'_1$, $\mathsf{Rd}_2 = \mathsf{Rd}'_2$, $\mathsf{Wr}_2 = \mathsf{Wr}'_2$, and $\mathsf{V}_2 \setminus \mathsf{Wr}_2 = \mathsf{V}'_2 \setminus \mathsf{Wr}'_2$. The proof relies on the fact that $\mathsf{m}^+_4 \models \mathsf{check}(P)$, and $\mathsf{V}^+_4 \cap \mathsf{St}(P) = \{p^{\mathbf{k}}_R \ : \ p \in \mathsf{Rd}_k\} \cup \{p^{\mathbf{k}}_W \ : \ p \in \mathsf{Wr}_k\}$.
5. $\begin{smallmatrix}\mathsf{m}'_1\\ \mathsf{m}'_2\end{smallmatrix} \triangleright \mathsf{m}'$. The proof relies on Lemma 4.
6. $\mathsf{m}[\![\pi_1 || \pi_2]\!]\mathsf{m}'$.

5.3 Reduction Axioms for Program Operators

The reduction axioms for program operators are in Table 2. Those for sequential and non-deterministic composition and for exogenous tests (PDL tests) are as in PDL. The one for endogenous tests $\varphi?$ checks whether φ remains true for any possible value of

$$\langle \varphi ? \rangle \psi \leftrightarrow \psi \wedge \varphi$$
$$\langle \varphi ?\!\!? \rangle \psi \leftrightarrow \psi \wedge [\mathop{;}_{p \in \mathbb{P}_\varphi} (\mathtt{r}_p ? \cup (\neg \mathtt{r}_p ?; (+p \cup -p)))] \varphi$$
$$\langle \pi_1 ; \pi_2 \rangle \varphi \leftrightarrow \langle \pi_1 \rangle \langle \pi_2 \rangle \varphi$$
$$\langle \pi_1 \cup \pi_2 \rangle \varphi \leftrightarrow \langle \pi_1 \rangle \varphi \vee \langle \pi_2 \rangle \varphi$$
$$\langle \pi^* \rangle \varphi \leftrightarrow \langle \pi^{\leq 2^{|\mathbb{P}_\varphi|}} \rangle \varphi$$
$$\langle \pi_1 \| \pi_2 \rangle \varphi \leftrightarrow \langle \mathsf{flatten}(\pi_1, \pi_2) \rangle \varphi$$

Table 2. Reduction axioms for program operators

the non-readable variables of φ. That for the Kleene star is familiar from DL-PA. That for parallel composition $\pi_1 \| \pi_2$ executes π_1 and π_2 in sequence: it starts by splitting up readability and writability between the two programs, then executes π_1, checks whether π_1 didn't change the readability and writability variables and whether all truth value changes it brought about are legal, and finally executes π_2 followed by the same checks for π_2.

Observe that the validity of the reduction axiom for endogenous tests relies on the fact that the copies $p^{\mathbf{1}}$ and $p^{\mathbf{2}}$ that are introduced by the program $\mathsf{split}(\mathbb{P}_{\pi_1 \| \pi_2})$ are fresh. The length of the right hand side can be shortened by restricting $\mathbb{P}_{\pi_1 \| \pi_2}$ to the propositional variables that are assigned by $\mathbb{P}_{\pi_1 \| \pi_2}$, i.e., to elements $p \in \mathbb{P}$ such that $+p$ or $-p$ occurs in $\mathbb{P}_{\pi_1 \| \pi_2}$.

The exhaustive application of the equivalences of Table 2 from the left to the right results in formulas whose program operators are either endogenous tests occurring in a readability statement $\mathtt{r}_p = \langle p ?\!\!? \rangle \top \vee \langle \neg p ?\!\!? \rangle \top$, or assignments of the form $\mathtt{r}{+}p$, $\mathtt{r}{-}p$, $\mathtt{w}{+}p$, $\mathtt{w}{-}p$, $+p$, or $-p$.

5.4 Reduction Axioms for Boolean Operators

We now turn to modal operators $\langle \pi \rangle$ where π is an atomic assignment, i.e., π is of the form $\mathtt{r}{+}p$, $\mathtt{r}{-}p$, $\mathtt{w}{+}p$, $\mathtt{w}{-}p$, $+p$, or $-p$. They are deterministic and can therefore be distributed over the boolean operators. The corresponding reduction axioms are in Table 3.

In the first equivalence $\langle +p \rangle \top \leftrightarrow \mathtt{w}_p$, the right hand side is nothing but an abbreviation of the left hand side. We nevertheless state it in order to highlight that the exhaustive application of these reduction axioms results in sequences of atomic assignments facing either $\mathtt{w}_p$ or $\mathtt{r}_q$. These sequences are going to be reduced in the next step.

5.5 Reduction Axioms for Assignments

When atomic programs face propositional variables or readability and writability statements then the modal operator can be eliminated (sometimes introducing a writability statement $\mathtt{w}_p$). The reduction axioms doing that are in Table 4.

As announced, the exhaustive application of the above axioms results in boolean combinations of propositional variables and readability and writability statements.

$$
\begin{array}{rlrl}
\langle +p\rangle\top &\leftrightarrow \mathtt{w}_p & \langle -p\rangle\top &\leftrightarrow \mathtt{w}_p \\
\langle \mathtt{r}{+}p\rangle\top &\leftrightarrow \top & \langle \mathtt{r}{-}p\rangle\top &\leftrightarrow \top \\
\langle \mathtt{w}{+}p\rangle\top &\leftrightarrow \top & \langle \mathtt{w}{-}p\rangle\top &\leftrightarrow \top \\
\langle +p\rangle\neg\varphi &\leftrightarrow \mathtt{w}_p \wedge \neg\langle +p\rangle\varphi & \langle -p\rangle\neg\varphi &\leftrightarrow \mathtt{w}_p \wedge \neg\langle -p\rangle\varphi \\
\langle \mathtt{r}{+}p\rangle\neg\varphi &\leftrightarrow \neg\langle \mathtt{r}{+}p\rangle\varphi & \langle \mathtt{r}{-}p\rangle\neg\varphi &\leftrightarrow \neg\langle \mathtt{r}{-}p\rangle\varphi \\
\langle \mathtt{w}{+}p\rangle\neg\varphi &\leftrightarrow \neg\langle \mathtt{w}{+}p\rangle\varphi & \langle \mathtt{w}{-}p\rangle\neg\varphi &\leftrightarrow \neg\langle \mathtt{w}{-}p\rangle\varphi \\
\langle +p\rangle(\varphi\vee\psi) &\leftrightarrow \langle +p\rangle\varphi \vee \langle +p\rangle\psi & \langle -p\rangle(\varphi\vee\psi) &\leftrightarrow \langle -p\rangle\varphi \vee \langle -p\rangle\psi \\
\langle \mathtt{r}{+}p\rangle(\varphi\vee\psi) &\leftrightarrow \langle \mathtt{r}{+}p\rangle\varphi \vee \langle \mathtt{r}{+}p\rangle\psi & \langle \mathtt{r}{-}p\rangle(\varphi\vee\psi) &\leftrightarrow \langle \mathtt{r}{-}p\rangle\varphi \vee \langle \mathtt{r}{-}p\rangle\psi \\
\langle \mathtt{w}{+}p\rangle(\varphi\vee\psi) &\leftrightarrow \langle \mathtt{w}{+}p\rangle\varphi \vee \langle \mathtt{w}{+}p\rangle\psi & \langle \mathtt{w}{-}p\rangle(\varphi\vee\psi) &\leftrightarrow \langle \mathtt{w}{-}p\rangle\varphi \vee \langle \mathtt{w}{-}p\rangle\psi
\end{array}
$$

Table 3. Reduction axioms for boolean operators

5.6 Soundness, Completeness, and Decidability

Let us call DL-PA$^{\|}$ our extension of DL-PA with parallel composition. Its axiomatisation is made up of

- an axiomatisation of propositional logic;
- the equivalences of Sections 5.3, 5.4, and 5.5;
- the inclusion axiom schema $\mathtt{w}_p \to \mathtt{r}_p$, which is an abbreviation of the formula $\langle +p\rangle\top \to (\langle p?\rangle\top \vee \langle \neg p?\rangle\top)$;
- the rule of equivalence for the modal operator "from $\varphi \leftrightarrow \psi$ infer $\langle\pi\rangle\varphi \leftrightarrow \langle\pi\rangle\psi$".

Theorem 1. *The axiomatisation of* DL-PA$^{\|}$ *is sound: if φ is provable with the axiomatics of* DL-PA$^{\|}$ *then it is* DL-PA$^{\|}$ *valid.*

Proof (sketch). We have to show that the inference rules preserve validity and the axioms are valid. For the reduction axiom for endogenous test, it suffices to observe that $\mathsf{m}[\![+p \cup -p]\!]\mathsf{m}'$ if and only if $\mathsf{m} \sim \mathsf{m}'$. The proof of validity of the reduction axiom for Kleene star can easily be adapted from the one in [BHT13]. The case of the reduction axiom for parallel composition is handled by Lemma 6. All other cases are straightforward.

Theorem 2. *The axiomatisation of* DL-PA$^{\|}$ *is complete: if φ is* DL-PA$^{\|}$ *valid then it is provable in the axiomatics of* DL-PA$^{\|}$.

Proof. The reduction axioms of Sections 5.3, 5.4, and 5.5 allow us to transform any formula into an equivalent boolean combination of propositional variables and readability and writability statements. (Their application requires the rule of replacement of equivalents, which is derivable because we have rules of equivalence for all the connectives of the language, in particular the above $RE(\langle\pi\rangle)$.) Let φ be the resulting formula. Then φ has a DL-PA$^{\|}$ model if and only if

$$\varphi \wedge \bigwedge_{p\in\mathbb{P}} (\mathtt{w}_p \to \mathtt{r}_p)$$

$$\langle +p\rangle q \leftrightarrow \begin{cases} \mathtt{w}_p & \text{if } q = p \\ \mathtt{w}_p \wedge q & \text{otherwise} \end{cases} \qquad \langle -p\rangle q \leftrightarrow \begin{cases} \bot & \text{if } q = p \\ \mathtt{w}_p \wedge q & \text{otherwise} \end{cases}$$

$$\langle \mathtt{r}{+}p\rangle q \leftrightarrow q \qquad \langle \mathtt{r}{-}p\rangle q \leftrightarrow q$$

$$\langle \mathtt{w}{+}p\rangle q \leftrightarrow q \qquad \langle \mathtt{w}{-}p\rangle q \leftrightarrow q$$

$$\langle +p\rangle \mathtt{r}_q \leftrightarrow \mathtt{w}_p \wedge \mathtt{r}_q \qquad \langle -p\rangle \mathtt{r}_q \leftrightarrow \mathtt{w}_p \wedge \mathtt{r}_q$$

$$\langle \mathtt{r}{+}p\rangle \mathtt{r}_q \leftrightarrow \begin{cases} \top & \text{if } q = p \\ \mathtt{r}_q & \text{otherwise} \end{cases} \qquad \langle \mathtt{r}{-}p\rangle \mathtt{r}_q \leftrightarrow \begin{cases} \bot & \text{if } q = p \\ \mathtt{r}_q & \text{otherwise} \end{cases}$$

$$\langle \mathtt{w}{+}p\rangle \mathtt{r}_q \leftrightarrow \begin{cases} \top & \text{if } q = p \\ \mathtt{r}_q & \text{otherwise} \end{cases} \qquad \langle \mathtt{w}{-}p\rangle \mathtt{r}_q \leftrightarrow \mathtt{r}_q$$

$$\langle +p\rangle \mathtt{w}_q \leftrightarrow \mathtt{w}_p \wedge \mathtt{w}_q \qquad \langle -p\rangle \mathtt{w}_q \leftrightarrow \mathtt{w}_p \wedge \mathtt{w}_q$$

$$\langle \mathtt{r}{+}p\rangle \mathtt{w}_q \leftrightarrow \mathtt{w}_q \qquad \langle \mathtt{r}{-}p\rangle \mathtt{w}_q \leftrightarrow \begin{cases} \bot & \text{if } q = p \\ \mathtt{w}_q & \text{otherwise} \end{cases}$$

$$\langle \mathtt{w}{+}p\rangle \mathtt{w}_q \leftrightarrow \begin{cases} \top & \text{if } q = p \\ \mathtt{w}_q & \text{otherwise} \end{cases} \qquad \langle \mathtt{w}{-}p\rangle \mathtt{w}_q \leftrightarrow \begin{cases} \bot & \text{if } q = p \\ \mathtt{w}_q & \text{otherwise} \end{cases}$$

Table 4. Reduction axioms for assignments

has a model in propositional logic, where in propositional logic, $\mathtt{r}_p$ and $\mathtt{w}_p$ are considered to be arbitrary propositional variables; so there is a priori no connection between them nor with the propositional variable p.

Based on the reduction of DL-PA$^{\|}$ formulas to boolean formulas (and the transformation of $\mathtt{r}_p$ and $\mathtt{w}_p$ from abbreviations into propositional variables), we may check the satisfiability of DL-PA$^{\|}$ formulas by means of propositional logic SAT solvers. This is however suboptimal because the reduction may result in a formula that is super-exponentially longer than the original formula. In the next section we explore another route.

6 Complexity via Translation into DL-PA

We establish PSPACE complexity of DL-PA$^{\|}$ satisfiability and model checking by translating formulas and programs to Dynamic Logic of Propositional Assignments DL-PA. The language of the latter is the fragment of that of DL-PA$^{\|}$: it has neither endogenous tests, nor readability and writability assignments, nor parallel composition. Hence the language of DL-PA is built by the following grammar:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \langle\pi\rangle\varphi$$
$$\pi ::= +p \mid -p \mid \varphi? \mid (\pi;\pi) \mid (\pi \cup \pi) \mid \pi^*$$

None of the operators of the language refers to the Rd-component or the Wr-component of models. The interpretation of DL-PA formulas and programs therefore only requires a valuation V.

$$
\begin{aligned}
\mathsf{split}(P) = {} & ;_{p\in P} \Big(\big((p?;+p^{\mathbf{1}};+p^{\mathbf{2}}) \cup (\neg p?;-p^{\mathbf{1}};-p^{\mathbf{2}})\big);\\
& \quad -\mathtt{w}_{p^{\mathbf{1}}};-\mathtt{r}_{p^{\mathbf{1}}};-\mathtt{w}_{p^{\mathbf{2}}};-\mathtt{r}_{p^{\mathbf{2}}};\\
& \quad \big(\neg \mathtt{r}_p? \cup\\
& \quad\quad (\mathtt{w}_p?;((+\mathtt{r}_{p^{\mathbf{1}}};+\mathtt{w}_{p^{\mathbf{1}}}) \cup (+\mathtt{r}_{p^{\mathbf{2}}};+\mathtt{w}_{p^{\mathbf{2}}}))) \cup\\
& \quad\quad (\neg \mathtt{w}_p \wedge \mathtt{r}_p?;(+\mathtt{r}_{p^{\mathbf{1}}} \cup +\mathtt{r}_{p^{\mathbf{2}}} \cup (+\mathtt{r}_{p^{\mathbf{1}}};+\mathtt{r}_{p^{\mathbf{2}}})))\big)\Big)\\
\mathsf{store}(P) = {} & ;_{p\in P} ;_{k\in\{1,2\}} \Big(((\mathtt{r}_{p^{\mathbf{k}}}?;+p^{\mathbf{k}}_R) \cup (\neg \mathtt{r}_{p^{\mathbf{k}}}?;-p^{\mathbf{k}}_R));\\
& \quad ((\mathtt{w}_{p^{\mathbf{k}}}?;+p^{\mathbf{k}}_W) \cup (\neg \mathtt{w}_{p^{\mathbf{k}}}?;-p^{\mathbf{k}}_W))\Big)\\
\mathsf{check}(P) = {} & \bigwedge_{p\in P} \bigwedge_{k\in\{1,2\}} \Big((p^{\mathbf{k}}_R \leftrightarrow \mathtt{r}_{p^{\mathbf{k}}}) \wedge (p^{\mathbf{k}}_W \leftrightarrow \mathtt{w}_{p^{\mathbf{k}}}) \wedge (\neg p^{\mathbf{k}}_W \rightarrow (p \leftrightarrow p^{\mathbf{k}}))\Big)\\
\mathsf{merge}(P) = {} & ;_{p\in P} \Big((\neg \mathtt{w}_p? \cup\\
& \quad ((\mathtt{w}_{p^{\mathbf{1}}} \wedge p^{\mathbf{1}}) \vee (\mathtt{w}_{p^{\mathbf{2}}} \wedge p^{\mathbf{2}})?;+p) \cup\\
& \quad ((\mathtt{w}_{p^{\mathbf{1}}} \wedge \neg p^{\mathbf{1}}) \vee (\mathtt{w}_{p^{\mathbf{2}}} \wedge \neg p^{\mathbf{2}})?;-p));\\
& ;_{k\in\{1,2\}} (-p^{\mathbf{k}}_R;-p^{\mathbf{k}}_W;-p^{\mathbf{k}};-\mathtt{w}_{p^{\mathbf{k}}};-\mathtt{r}_{p^{\mathbf{k}}})\Big)\\
\mathsf{flatten}(\pi_1,\pi_2) = {} & \mathsf{split}\left(\mathbb{P}_{\pi_1\|\pi_2}\right);\mathsf{store}\left(\mathbb{P}_{\pi_1\|\pi_2}\right);{\pi_1}^{\mathbf{1}};{\pi_2}^{\mathbf{2}};\mathsf{check}\left(\mathbb{P}_{\pi_1\|\pi_2}\right)?;\mathsf{merge}\left(\mathbb{P}_{\pi_1\|\pi_2}\right)
\end{aligned}
$$

Table 5. Adaptation of the programs and formulas of Table 1 to the translation into DL-PA.

Our translation from DL-PA$^{\|}$ to DL-PA eliminates endogenous tests and parallel composition. This is done in a way that is similar to their reduction axioms of Table 2. It moreover transforms readability and writability statements into special propositional variables $\mathtt{r}_p$ and $\mathtt{w}_p$, similar to the reduction axioms of Table 4.

To make this formal, let the set of *atomic formulas* be

$$\mathbb{X} = \mathbb{P} \cup \{\mathtt{w}_p \ : \ p \in \mathbb{P}\} \cup \{\mathtt{r}_p \ : \ p \in \mathbb{P}\}.$$

Given a set of propositional variables $P \subseteq \mathbb{P}$, $\mathbb{R}_P = \{\mathtt{r}_p \ : \ p \in P\}$ is the associated set of read-variables and $\mathbb{W}_P = \{\mathtt{w}_p \ : \ p \in P\}$ is the associated set of write-variables. Hence $\mathbb{X} = \mathbb{P} \cup \mathbb{R}_{\mathbb{P}} \cup \mathbb{W}_{\mathbb{P}}$. As before, the set of propositional variables occurring in a formula φ is noted $\mathbb{P}_\varphi$ and the set of those occurring in a program π is noted $\mathbb{P}_\pi$. This now includes the p's in $\mathtt{r}_p$ and $\mathtt{w}_p$. For example, $\mathbb{P}_{p\wedge\langle +w_q\rangle\neg \mathtt{r}_p} = \{p, q\}$.

We translate the DL-PA$^{\|}$ programs $\mathtt{r}{+}p$, $\mathtt{r}{-}p$, $\mathtt{w}{+}p$, and $\mathtt{w}{-}p$ into the DL-PA programs $+\mathtt{r}_p$, $-\mathtt{r}_p$, $+\mathtt{w}_p$ and $-\mathtt{w}_p$. Moreover, we have to 'spell out' that $-\mathtt{w}_p$ has side effect $-\mathtt{r}_p$ and that $+\mathtt{r}_p$ has side effect $+\mathtt{w}_p$. Hence the programs and formulas of Table 1 become the DL-PA programs and formulas listed in Table 5. Notice that readability and writability statements are no longer DL-PA$^{\|}$ abbreviations, but are now DL-PA propositional variables.

Given a DL-PA$^{\|}$ program or formula, its translation into DL-PA basically follows the reduction axiom for endogenous tests $?$ and parallel composition $\|$ of Table 2. We replace:

1. all occurrences of $\varphi?$ with $\left[;_{p\in\mathbb{P}_\varphi}\left(\mathtt{r}_p? \cup (\neg\mathtt{r}_p?; (+p \cup -p))\right)\right]\varphi?$,
2. all occurrences of $\pi_1 \| \pi_2$ with $\mathsf{flatten}(\pi_1, \pi_2)$,
3. all occurrences of $+p$ with $\mathtt{w}_p?; +p$,
4. all occurrences of $-p$ with $\mathtt{w}_p?; -p$,
5. all occurrences of $\mathtt{r}{+}p$ with $+\mathtt{r}_p$,
6. all occurrences of $\mathtt{r}{-}p$ with $-\mathtt{w}_p; -\mathtt{r}_p$,
7. all occurrences of $\mathtt{w}{+}p$ with $+\mathtt{r}_p; +\mathtt{w}_p$,
8. all occurrences of $\mathtt{w}{-}p$ with $-\mathtt{w}_p$.

Let $t(\pi)$ be the translation of the DL-PA$^{\|}$ program π and $t(\varphi)$ the translation of the DL-PA$^{\|}$ formula φ. Remember that when $t(\pi)$ and $t(\varphi)$ are interpreted in DL-PA, the variables $\mathtt{r}_p$ and $\mathtt{w}_p$ are considered to be arbitrary propositional variables.

Lemma 7. *For all* DL-PA$^{\|}$ *programs* π *and formula* φ*, and all* DL-PA$^{\|}$ *models* m *and* m' *such that* $\mathsf{m} \cap P = \mathsf{m}$*,*

$$\mathsf{m}[\![\pi]\!]\mathsf{m}' \text{ if and only if } \mathsf{V}^+[\![t(\pi)]\!]^{\text{DL-PA}}\mathsf{V}'^+, \text{ and}$$
$$\mathsf{m} \models \varphi \text{ if and only if } \mathsf{V}^+ \models_{\text{DL-PA}} t(\varphi)$$

where $\mathsf{V}^+ = \mathsf{V} \cup \mathbb{R}_{\mathsf{Rd}} \cup \mathbb{W}_{\mathsf{Wr}}$ *and* $\mathsf{V}'^+ = \mathsf{V}' \cup \mathbb{R}_{\mathsf{Rd}'} \cup \mathbb{W}_{\mathsf{Wr}'}$.

Proof (sketch). The proof is by simultaneous induction on the size of π or φ. The cases for endogenous test and parallel composition are similar to those in the proof of Theorem 1. The other cases are straightforward.

The following theorem is a direct corollary of the previous lemma.

Theorem 3. *A* DL-PA$^{\|}$ *formula* φ *is* DL-PA$^{\|}$*-satisfiable if and only if the* DL-PA *formula* $t(\varphi) \wedge \bigwedge_{p\in\mathbb{P}_\varphi}(\mathtt{w}_p \to \mathtt{r}_p)$ *is* DL-PA *satisfiable.*

We can now state our complexity results.

Lemma 8. *The translation t is polynomial.*

Proof. It can easily be checked that for all P, $\mathsf{split}(P)$, $\mathsf{store}(P)$, $\mathsf{check}(P)$ and $\mathsf{merge}(P)$ are linear in the size of P. Therefore, $\mathsf{flatten}(\pi_1, \pi_2)$ is linear in the size of π_1 plus the size of π_2. Similarly, $\left[;_{p\in\mathbb{P}_\varphi}\left(\mathtt{r}_p? \cup (\neg\mathtt{r}_p?; (+p \cup -p))\right)\right]\varphi?$ is linear in the size of φ. All other translation expressions are clearly linear too. Hence applying t from the root of the syntax tree to its leaves, $t(\varphi)$ can be computed in time polynomial in the size of φ.

Theorem 4. DL-PA$^{\|}$ *model and satisfiability checking are both PSPACE complete.*

Proof. First, PSPACE membership of DL-PA$^{\|}$ model and satisfiability checking follows from Lemmas 7 and 8. Second, since the language of DL-PA$^{\|}$ contains that of DL-PA and since model and satisfiability checking are PSPACE hard for the latter [BHST14], it follows that DL-PA$^{\|}$ model and satisfiability checking are PSPACE hard, too.

7 Discussion and Conclusion

We have added to Dynamic Logic of Propositional Assignments DL-PA a parallel composition operator in the spirit of separation logics. Our semantics augments DL-PA valuations by readability and writability information. We have provided an axiomatisation in terms of a complete set of reduction axioms. Our reduction to DL-PA ensures decidability. We have also proved PSPACE complexity via a polynomial translation to DL-PA.

We have adopted a stricter stance on race conditions than in [HMV19] where e.g. the program $+p\|+p$ is executable. Let us briefly compare these two semantics. In our case, the intuition is that $\pi_1\|\pi_2$ is executable if any interleaving of the components of π_1 and π_2 is executable, and that any of these interleavings leads to the same outcome. This is not guaranteed in the approach of [HMV19], which is motivated by parallel planning. There, it is generally considered that two actions that are executed in parallel should not interfere [BF97]: they should not have conflicting effects and there should be no cross-interaction, where the second condition means that the effect of one action should not destroy the precondition of the other, and vice versa. For example, in a world of blocks the actions

$$\mathsf{liftLeft}(b) = \mathsf{OnTable}(b)?; -\mathsf{OnTable}(b); +\mathsf{HoldsLeft}(b),$$
$$\mathsf{liftRight}(b) = \mathsf{OnTable}(b)?; -\mathsf{OnTable}(b); +\mathsf{HoldsRight}(b)$$

of lifting block b with the left robot arm and with the right robot arm have cross-interaction because one of the effects of $\mathsf{liftLeft}(b)$ is $\neg\mathsf{OnTable}(b)$, which makes the precondition $\mathsf{pre}(\mathsf{liftRight}(b)) = \mathsf{OnTable}(b)$ of $\mathsf{lifRight}(b)$ false. The more liberal semantics of [HMV19] makes that it is not enough to describe the parallel execution of actions $\pi_1, \ldots, \pi_n$ as a step of a parallel plan by $\pi_1\| \ldots \|\pi_n$. Instead, the absence of cross-interactions has to be checked 'by hand', namely by explicitly inserting a DL-PA test after each π_i that the preconditions of the other actions are not violated: a step of a parallel plan is described by the program.

$$(\pi_1; \bigwedge_{j\neq 1} \mathsf{pre}(\pi_j)?)\| \ldots \|(\pi_n; \bigwedge_{j\neq n} \mathsf{pre}(\pi_j)?).$$

This is not necessary in our semantics where the absence of cross interaction between actions that are performed in parallel is 'built-in'. Indeed, the parallel composition $\mathsf{liftLeft}(b)\|\mathsf{liftRight}(b)$ is not executable in DL-PA$^{\|}$: each subprogram requires writability of $\mathsf{OnTable}(b)$ to be executable. Our logic DL-PA$^{\|}$ can therefore be expected to provide a more appropriate base for parallel planning.

The language of the extension of DL-PA of [HMV19] also contains an operator of inclusive nondeterministic composition, noted $\sqcup$. While the standard inclusive nondeterministic composition $\pi_1 \cup \pi_2$ of PDL and DL-PA$^{\|}$ is read "do either π_1 or π_2", the program $\pi_1 \sqcup \pi_2$ is read "do π_1 or π_2 *or both*". It has the same semantics as $\pi_1 \cup \pi_2 \cup (\pi_1\|\pi_2)$ and hence does not add expressivity. It is shown in [HMV19] that it does not increase succinctness either. This is proved by a polynomial reduction that uses the same 'flattening' programs as the reduction of parallel composition. These programs are similar

to our programs in Table 5, we therefore expect that inclusive nondeterministic composition does not increase the succinctness of the language of DL-PA$^{\|}$ either.

The mathematical properties of DL-PA$^{\|}$ compare favourably with the high complexity or even undecidability of the other extensions of dynamic logic by a separating parallel composition operator that were proposed in the literature [BT14,Bou16]. Just as ours, the latter line of work is in the spirit of separation logic, having splitting and merging operations that are defined on system states. The axiomatisation that was introduced and studied in [BB18] is restricted to the star-free fragment and the authors had to add propositional quantifiers in order to make parallel composition definable. This contrasts with the simplicity of our axiomatisation of DL-PA$^{\|}$ that we obtained by adding reduction axioms to the axiomatisation of DL-PA. This can be related to the fact that propositional quantifiers can be expressed in DL-PA: $\exists p\varphi$ is equivalent to $\langle +p \cup -p\rangle\varphi$ and $\forall p\varphi$ is equivalent to $[+p \cup -p]\varphi$. Just as DL-PA can be viewed as an instance of PDL—the interpretation of atomic programs moves from PDL's abstract relation between states to concrete updates of valuations—, DL-PA$^{\|}$ can be viewed as an instance of the logic of [BdFV11] where the interpretation of parallel composition no longer resorts to an abstract relation $\star$ associating three states, but instead has concrete functions that split and merge valuations and that are constrained by readability and writability information.

8 Acknowledgements

The paper benefited from comments and remarks from the reviewers as well as from the attendees of DaLí 2019, in particular Alexandru Baltag, Raul Fervari, Rainer Hähnle and Dexter Kozen. We would like to particularly thank the three reviewers of JLAMP who provided detailed and well-informed reviews. We did our best take all these comments into account.

Andreas Herzig was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. Nicolas Troquard was supported by UNIBZ CRC 2019 project IN2092, Computations in Resource Aware Systems (CompRAS).

A Appendix: Associativity of Parallel Composition

Alongside commutativity, associativity is a desirable property of a parallel composition operator. While the operator defined in Section 4 is clearly commutative, whether it is associative does not strike the eye. In fact, a detailed proof of it is rather cumbersome. We present it here.

Proposition 1. $\mathsf{m}[\![\pi_1\|(\pi_2\|\pi_3)]\!]\mathsf{m}'$ *iff* $\mathsf{m}[\![(\pi_1\|\pi_2)\|\pi_3]\!]\mathsf{m}'$.

Proof. Figure 3 illustrates precisely what we are going to show. Suppose $\mathsf{m} = \langle \mathsf{Rd}, \mathsf{Wr}, \mathsf{V}\rangle$ and $\mathsf{m}' = \langle \mathsf{Rd}', \mathsf{Wr}', \mathsf{V}'\rangle$.

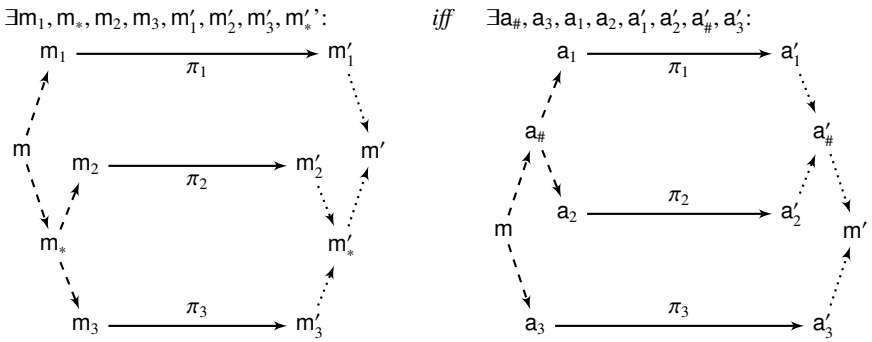

Fig. 3. Illustration of associativity. Visual aid for the proof of Proposition 1.

Left-hand side. $\mathsf{m}[\![\pi_1 \| (\pi_2 \| \pi_3)]\!]\mathsf{m}'$.

There are $\mathsf{m}_1, \mathsf{m}_*, \mathsf{m}'_1, \mathsf{m}'_*$ such that (with, $\mathsf{m}_1 = \langle \mathsf{Rd}_1, \mathsf{Wr}_1, \mathsf{V}_1 \rangle$, $\mathsf{m}'_1 = \langle \mathsf{Rd}'_1, \mathsf{Wr}'_1, \mathsf{V}'_1 \rangle$, $\mathsf{m}_* = \langle \mathsf{Rd}_*, \mathsf{Wr}_*, \mathsf{V}_* \rangle$, $\mathsf{m}'_* = \langle \mathsf{Rd}'_*, \mathsf{Wr}'_*, \mathsf{V}'_* \rangle$):

1. $\mathsf{m} \triangleleft \begin{smallmatrix} \mathsf{m}_1 \\ \mathsf{m}_* \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{m}'_1 \\ \mathsf{m}'_* \end{smallmatrix} \triangleright \mathsf{m}'$
2. $\mathsf{m}_1[\![\pi_1]\!]\mathsf{m}'_1$
3. $\mathsf{Rd}_1 = \mathsf{Rd}'_1$ and $\mathsf{Wr}_1 = \mathsf{Wr}'_1$ and $\mathsf{V}_1 \setminus \mathsf{Wr}_1 = \mathsf{V}'_1 \setminus \mathsf{Wr}'_1$
4. $\mathsf{m}_*[\![\pi_2 \| \pi_3]\!]\mathsf{m}'_*$
5. $\mathsf{Rd}_* = \mathsf{Rd}'_*$ and $\mathsf{Wr}_* = \mathsf{Wr}'_*$ and $\mathsf{V}_* \setminus \mathsf{Wr}_* = \mathsf{V}'_* \setminus \mathsf{Wr}'_*$

Item 4 is equivalent to: there are $\mathsf{m}_2, \mathsf{m}_3, \mathsf{m}'_2, \mathsf{m}'_3$ such that (with $\mathsf{m}_2 = \langle \mathsf{Rd}_2, \mathsf{Wr}_2, \mathsf{V}_2 \rangle$, $\mathsf{m}'_2 = \langle \mathsf{Rd}'_2, \mathsf{Wr}'_2, \mathsf{V}'_2 \rangle$, $\mathsf{m}_3 = \langle \mathsf{Rd}_3, \mathsf{Wr}_3, \mathsf{V}_3 \rangle$, $\mathsf{m}'_3 = \langle \mathsf{Rd}'_3, \mathsf{Wr}'_3, \mathsf{V}'_3 \rangle$):

6. $\mathsf{m}_* \triangleleft \begin{smallmatrix} \mathsf{m}_2 \\ \mathsf{m}_3 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{m}'_2 \\ \mathsf{m}'_3 \end{smallmatrix} \triangleright \mathsf{m}'_*$,
7. $\mathsf{m}_2[\![\pi_2]\!]\mathsf{m}'_2$,
8. $\mathsf{Rd}_2 = \mathsf{Rd}'_2$ and $\mathsf{Wr}_2 = \mathsf{Wr}'_2$ and $\mathsf{V}_2 \setminus \mathsf{Wr}_2 = \mathsf{V}'_2 \setminus \mathsf{Wr}'_2$,
9. $\mathsf{m}_3[\![\pi_3]\!]\mathsf{m}'_3$,
10. $\mathsf{Rd}_3 = \mathsf{Rd}'_3$ and $\mathsf{Wr}_3 = \mathsf{Wr}'_3$ and $\mathsf{V}_3 \setminus \mathsf{Wr}_3 = \mathsf{V}'_3 \setminus \mathsf{Wr}'_3$.

Item 1 is $\mathsf{m} \triangleleft \begin{smallmatrix} \mathsf{m}_1 \\ \mathsf{m}_* \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{m}'_1 \\ \mathsf{m}'_* \end{smallmatrix} \triangleright \mathsf{m}'$ iff:

11. $\mathsf{Wr}_1 \cap \mathsf{Rd}_* = \mathsf{Wr}_* \cap \mathsf{Rd}_1 = \emptyset$ (m_1 and m_* are RW-compatible),
12. $\mathsf{Rd} = \mathsf{Rd}_1 \cup \mathsf{Rd}_*$, and $\mathsf{Wr} = \mathsf{Wr}_1 \cup \mathsf{Wr}_*$, and $\mathsf{V} = \mathsf{V}_1 = \mathsf{V}_*$,
13. $\mathsf{Wr}'_1 \cap \mathsf{Rd}'_* = \mathsf{Wr}'_* \cap \mathsf{Rd}'_1 = \emptyset$ (m'_1 and m'_* are RW-compatible),
14. $\mathsf{Rd}' = \mathsf{Rd}'_1 \cup \mathsf{Rd}'_*$, $\mathsf{Wr}' = \mathsf{Wr}'_1 \cup \mathsf{Wr}'_*$, and $\mathsf{V}'_1 \setminus \mathsf{Wr}' = \mathsf{V}'_* \setminus \mathsf{Wr}'$, and $\mathsf{V}' = (\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_* \cap \mathsf{Wr}'_*) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_*)$.

Item 6 is $\mathsf{m} \triangleleft \begin{smallmatrix} \mathsf{m}_2 \\ \mathsf{m}_3 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{m}'_2 \\ \mathsf{m}'_3 \end{smallmatrix} \triangleright \mathsf{m}'$ iff:

15. $\mathsf{Wr}_2 \cap \mathsf{Rd}_3 = \mathsf{Wr}_3 \cap \mathsf{Rd}_2 = \emptyset$ (m_2 and m_3 are RW-compatible),
16. $\mathsf{Rd}_* = \mathsf{Rd}_2 \cup \mathsf{Rd}_3$, and $\mathsf{Wr}_* = \mathsf{Wr}_2 \cup \mathsf{Wr}_3$, and $\mathsf{V}_* = \mathsf{V}_2 = \mathsf{V}_3$,
17. $\mathsf{Wr}'_2 \cap \mathsf{Rd}'_3 = \mathsf{Wr}'_3 \cap \mathsf{Rd}'_2 = \emptyset$ (m'_2 and m'_3 are RW-compatible),
18. $\mathsf{Rd}'_* = \mathsf{Rd}'_2 \cup \mathsf{Rd}'_3$, $\mathsf{Wr}'_* = \mathsf{Wr}'_2 \cup \mathsf{Wr}'_3$, and $\mathsf{V}'_2 \setminus \mathsf{Wr}'_* = \mathsf{V}'_3 \setminus \mathsf{Wr}'_*$, and $\mathsf{V}'_* = (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_2 \cap \mathsf{V}'_3)$.

Right-hand side. $\mathsf{m}[\![(\pi_1 \| \pi_2) \| \pi_3]\!]\mathsf{m}'$.

There are $\mathsf{a}_\# = \langle \mathsf{Rda}_\#, \mathsf{Wra}_\#, \mathsf{Va}_\# \rangle$, $\mathsf{a}'_\# = \langle \mathsf{Rda}'_\#, \mathsf{Wra}'_\#, \mathsf{Va}'_\# \rangle$, $\mathsf{a}_3 = \langle \mathsf{Rda}_3, \mathsf{Wra}_3, \mathsf{Va}_3 \rangle$ and $\mathsf{a}'_3 = \langle \mathsf{Rda}'_3, \mathsf{Wra}'_3, \mathsf{Va}'_3 \rangle$ such that:

19. $\mathsf{m} \triangleleft \begin{smallmatrix} \mathsf{a}_\# \\ \mathsf{a}_3 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{a}'_\# \\ \mathsf{a}'_3 \end{smallmatrix} \triangleright \mathsf{m}'$,
20. $\mathsf{a}_\#[\![\pi_1 \| \pi_2]\!]\mathsf{a}'_\#$,
21. $\mathsf{Rda}_\# = \mathsf{Rda}'_\#$ and $\mathsf{Wra}_\# = \mathsf{Wra}'_\#$ and $\mathsf{Va}_\# \setminus \mathsf{Wra}_\# = \mathsf{Va}'_\# \setminus \mathsf{Wra}'_\#$,
22. $\mathsf{a}_3[\![\pi_3]\!]\mathsf{a}'_3$,
23. $\mathsf{Rda}_3 = \mathsf{Rda}'_3$ and $\mathsf{Wra}_3 = \mathsf{Wra}'_3$ and $\mathsf{Va}_3 \setminus \mathsf{Wra}_3 = \mathsf{Va}'_3 \setminus \mathsf{Wra}'_3$.

Item 20 is equivalent to: there are $\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}'_1, \mathsf{a}'_2$ such that (with, $\mathsf{a}_1 = \langle \mathsf{Rda}_1, \mathsf{Wra}_1, \mathsf{Va}_1 \rangle$, $\mathsf{a}'_1 = \langle \mathsf{Rda}'_1, \mathsf{Wra}'_1, \mathsf{Va}'_1 \rangle$, $\mathsf{a}_2 = \langle \mathsf{Rda}_2, \mathsf{Wra}_2, \mathsf{Va}_2 \rangle$, $\mathsf{a}'_2 = \langle \mathsf{Rda}'_2, \mathsf{Wra}'_2, \mathsf{Va}'_2 \rangle$):

24. $\mathsf{a}_\# \triangleleft \begin{smallmatrix} \mathsf{a}_1 \\ \mathsf{a}_2 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{a}'_1 \\ \mathsf{a}'_2 \end{smallmatrix} \triangleright \mathsf{a}'_\#$,
25. $\mathsf{a}_1[\![\pi_1]\!]\mathsf{a}'_1$,
26. $\mathsf{Rda}_1 = \mathsf{Rda}'_1$, and $\mathsf{Wra}_1 = \mathsf{Wra}'_1$ and $\mathsf{Va}_1 \setminus \mathsf{Wra}_1 = \mathsf{Va}'_1 \setminus \mathsf{Wra}'_1$,
27. $\mathsf{a}_2[\![\pi_2]\!]\mathsf{a}'_2$,
28. $\mathsf{Rda}_2 = \mathsf{Rda}'_2$ and $\mathsf{Wra}_2 = \mathsf{Wra}'_2$ and $\mathsf{Va}_2 \setminus \mathsf{Wra}_2 = \mathsf{Va}'_2 \setminus \mathsf{Wra}'_2$.

Item 19 is $\mathsf{m} \triangleleft \begin{smallmatrix} \mathsf{a}_\# \\ \mathsf{a}_3 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{a}'_\# \\ \mathsf{a}'_3 \end{smallmatrix} \triangleright \mathsf{m}'$ iff:

29. $\mathsf{Wra}_\# \cap \mathsf{Rda}_3 = \mathsf{Wra}_3 \cap \mathsf{Rda}_\# = \emptyset$ ($\mathsf{a}_\#$ and a_3 are RW-compatible),
30. $\mathsf{Rd} = \mathsf{Rda}_\# \cup \mathsf{Rda}_3$, and $\mathsf{Wr} = \mathsf{Wra}_\# \cup \mathsf{Wra}_3$, and $\mathsf{V} = \mathsf{Va}_\# = \mathsf{Va}_3$,
31. $\mathsf{Wra}'_\# \cap \mathsf{Rda}'_3 = \mathsf{Wra}'_3 \cap \mathsf{Rda}'_\# = \emptyset$ ($\mathsf{a}'_\#$ and a'_3 are RW-compatible),
32. $\mathsf{Rd}' = \mathsf{Rda}'_\# \cup \mathsf{Rda}'_3$, $\mathsf{Wr}' = \mathsf{Wra}'_\# \cup \mathsf{Wra}'_3$, and $\mathsf{Va}'_\# \setminus \mathsf{Wr}' = \mathsf{Va}'_3 \setminus \mathsf{Wr}'$, and $\mathsf{V}' = (\mathsf{Va}'_\# \cap \mathsf{Wra}'_\#) \cup (\mathsf{Va}'_3 \cap \mathsf{Wra}'_3) \cup (\mathsf{Va}'_\# \cap \mathsf{Va}'_3)$.

Item 24 is $\mathsf{a}_\# \triangleleft \begin{smallmatrix} \mathsf{a}_1 \\ \mathsf{a}_2 \end{smallmatrix}$ and $\begin{smallmatrix} \mathsf{a}'_1 \\ \mathsf{a}'_2 \end{smallmatrix} \triangleright \mathsf{a}'_\#$ iff:

33. $\mathsf{Wra}_1 \cap \mathsf{Rda}_2 = \mathsf{Wra}_2 \cap \mathsf{Rda}_1 = \emptyset$ (a_1 and a_2 are RW-compatible),
34. $\mathsf{Rda}_\# = \mathsf{Rda}_1 \cup \mathsf{Rda}_2$, and $\mathsf{Wra}_\# = \mathsf{Wra}_1 \cup \mathsf{Wra}_2$, and $\mathsf{Va}_\# = \mathsf{Va}_1 = \mathsf{Va}_2$,
35. $\mathsf{Wra}'_1 \cap \mathsf{Rda}'_2 = \mathsf{Wra}'_2 \cap \mathsf{Rda}'_1 = \emptyset$ (a'_1 and a'_2 are RW-compatible),
36. $\mathsf{Rda}'_\# = \mathsf{Rda}'_1 \cup \mathsf{Rda}'_2$, $\mathsf{Wra}'_\# = \mathsf{Wra}'_1 \cup \mathsf{Wra}'_2$, and $\mathsf{Va}'_1 \setminus \mathsf{Wra}'_\# = \mathsf{Va}'_2 \setminus \mathsf{Wra}'_\#$, and $\mathsf{Va}'_\# = (\mathsf{Va}'_1 \cap \mathsf{Wra}'_1) \cup (\mathsf{Va}'_2 \cap \mathsf{Wra}'_2) \cup (\mathsf{Va}'_1 \cap \mathsf{Va}'_2)$.

Left to right. Suppose lhs. We define:

- $\mathsf{a}_\# = \langle \mathsf{Rda}_\#, \mathsf{Wra}_\#, \mathsf{Va}_\# \rangle = \langle \mathsf{Rd}_1 \cup \mathsf{Rd}_2, \mathsf{Wr}_1 \cup \mathsf{Wr}_2, \mathsf{V} \rangle$,
- $\mathsf{a}_1 = \langle \mathsf{Rda}_1, \mathsf{Wra}_1, \mathsf{Va}_1 \rangle = \mathsf{m}_1 = \langle \mathsf{Rd}_1, \mathsf{Wr}_1, \mathsf{V} \rangle$,
- $\mathsf{a}_2 = \langle \mathsf{Rda}_2, \mathsf{Wra}_2, \mathsf{Va}_2 \rangle = \mathsf{m}_2 = \langle \mathsf{Rd}_2, \mathsf{Wr}_2, \mathsf{V} \rangle$,
- $\mathsf{a}_3 = \langle \mathsf{Rda}_3, \mathsf{Wra}_3, \mathsf{Va}_3 \rangle = \mathsf{m}_3 = \langle \mathsf{Rd}_3, \mathsf{Wr}_3, \mathsf{V} \rangle$,
- $\mathsf{a}'_\# = \langle \mathsf{Rda}'_\#, \mathsf{Wra}'_\#, \mathsf{Va}'_\# \rangle = \langle \mathsf{Rd}'_1 \cup \mathsf{Rd}'_2, \mathsf{Wr}'_1 \cup \mathsf{Wr}'_2, (\mathsf{V}'_1 \cap \mathsf{Wr}_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2) \rangle$,
- $\mathsf{a}'_1 = \langle \mathsf{Rda}'_1, \mathsf{Wra}'_1, \mathsf{Va}'_1 \rangle = \mathsf{m}'_1 = (\mathsf{Rd}'_1, \mathsf{Wr}'_1, \mathsf{V}'_1 \rangle$,
- $\mathsf{a}'_2 = \langle \mathsf{Rda}'_2, \mathsf{Wra}'_2, \mathsf{Va}'_2 \rangle = \mathsf{m}'_2 = \langle \mathsf{Rd}'_2, \mathsf{Wr}'_2, \mathsf{V}'_2 \rangle$,
- $\mathsf{a}'_3 = \langle \mathsf{Rda}'_3, \mathsf{Wra}'_3, \mathsf{Va}'_3 \rangle = \mathsf{m}'_3 = \langle \mathsf{Rd}'_3, \mathsf{Wr}'_3, \mathsf{V}'_3 \rangle$.

We must show that these models satisfy all the properties from 19 through 36.

19 if and only if

29 $\mathsf{Wra}_{\#} \cap \mathsf{Rda}_3 = \emptyset$, $\mathsf{Wra}_3 \cap \mathsf{Rda}_{\#} = \emptyset$. It holds because:

- $\mathsf{Wra}_{\#} \cap \mathsf{Rda}_3 = (\mathsf{Wr}_1 \cup \mathsf{Wr}_2) \cap \mathsf{Rd}_3 = (\mathsf{Wr}_1 \cap \mathsf{Rd}_3) \cup (\mathsf{Wr}_2 \cap \mathsf{Rd}_3)$. By 15, we have $\mathsf{Wr}_2 \cap \mathsf{Rd}_3 = \emptyset$. By 16, we have $\mathsf{Rd}_3 \subseteq \mathsf{Rd}_*$. By 11, we have $\mathsf{Wr}_1 \cap \mathsf{Rd}_* = \emptyset$. So, $\mathsf{Wr}_1 \cap \mathsf{Rd}_3 = \emptyset$.
- $\mathsf{Wra}_3 \cap \mathsf{Rda}_{\#} = \mathsf{Wr}_3 \cap (\mathsf{Rd}_1 \cup \mathsf{Rd}_2) = (\mathsf{Wr}_3 \cap \mathsf{Rd}_1) \cup (\mathsf{Wr}_3 \cap \mathsf{Rd}_2)$. By 15, we have $\mathsf{Wr}_3 \cap \mathsf{Rd}_2 = \emptyset$. By 11, we have $\mathsf{Wr}_* \cap \mathsf{Rd}_1 = \emptyset$. By 16, we have $\mathsf{Wr}_3 \subseteq \mathsf{Wr}_*$. So, $\mathsf{Wr}_3 \cap \mathsf{Rd}_1 = \emptyset$.

30 $\mathsf{Rda}_{\#} \cup \mathsf{Rda}_3 = (\mathsf{Rd}_1 \cup \mathsf{Rd}_2) \cup \mathsf{Rd}_3 = \mathsf{Rd}_1 \cup \mathsf{Rd}_* = \mathsf{Rd}$ (definition and 16 and 12). $\mathsf{Wra}_{\#} \cup \mathsf{Wra}_3 = (\mathsf{Wr}_1 \cup \mathsf{Wr}_2) \cup \mathsf{Wr}_3 = \mathsf{Wr}_1 \cup \mathsf{Wr}_* = \mathsf{Wr}$ (definition and 16 and 12). $\mathsf{V} = \mathsf{Va}_{\#} = \mathsf{Va}_3$ (definition).

31 $\mathsf{Wra}'_{\#} \cap \mathsf{Rda}'_3 = \mathsf{Wra}'_3 \cap \mathsf{Rda}'_{\#} = \emptyset$. It holds because:

- $\mathsf{Wra}'_{\#} \cap \mathsf{Rda}'_3 = (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2) \cap \mathsf{Rd}'_3$ by definition. It is equal to $(\mathsf{Wr}_1 \cap \mathsf{Rd}_3) \cup (\mathsf{Wr}_2 \cap \mathsf{Rd}_3)$, by 3, 8, 10. We have $\mathsf{Wr}_1 \cap \mathsf{Rd}_* = \emptyset$ (11), and $\mathsf{Rd}_3 \subseteq \mathsf{Rd}_*$ (16). So $\mathsf{Wr}_1 \cap \mathsf{Rd}_3 = \emptyset$. We have $\mathsf{Wr}_2 \cap \mathsf{Rd}_3 = \emptyset$ (15).
- $\mathsf{Wra}'_3 \cap \mathsf{Rda}'_{\#} = \mathsf{Wra}'_3 \cap (\mathsf{Rd}'_1 \cup \mathsf{Rd}'_2)$ by definition. It is equal to $(\mathsf{Wra}_3 \cap \mathsf{Rd}_1) \cup (\mathsf{Wra}_3 \cap \mathsf{Rd}_2)$, by 10, 3, 8. We have $\mathsf{Wr}_3 \subseteq \mathsf{Wr}_*$ (16) and $\mathsf{Wr}_* \cap \mathsf{Rd}_1 = \emptyset$ (11). So $\mathsf{Wra}_3 \cap \mathsf{Rd}_1 = \emptyset$. We have $\mathsf{Wra}_3 \cap \mathsf{Rd}_2 = \emptyset$ (15).

32

- Instrumental claims:
  - (claim 1) $\mathsf{Wr}'_* = \mathsf{Wr}_*$, by 16, 7, 9 and 18.
  - (claim 2) $\mathsf{Wr} = \mathsf{Wr}'$, by 12, 3, claim 1, 14.
  - (claim 3) $\mathsf{Wr}_1 \subseteq \mathsf{Wr}'$, by claim 2, and 12.
  - (claim 4.1) $\mathsf{Wr}_2 \subseteq \mathsf{Wr}'$, by claim 2, 16, and 12.
  - (claim 4.2) $\mathsf{Wr}_3 \subseteq \mathsf{Wr}'$, by claim 2, 16, and 12.
  - (claim 5) $\mathsf{V}'_1 \setminus \mathsf{Wr}' = \mathsf{V}_1 \setminus \mathsf{Wr}'$, by 3, 12, and claim 2.
  - (claim 6) $\mathsf{V}'_2 \setminus \mathsf{Wr}' = \mathsf{V}_2 \setminus \mathsf{Wr}'$, by 8, 16, and 12.
  - (claim 7) $\mathsf{V}'_3 \setminus \mathsf{Wr}' = \mathsf{V}_3 \setminus \mathsf{Wr}'$, by 10, 16, and 12.
  - (claim 8) $\mathsf{Wr}_1 \cap \mathsf{Wr}_2 = \emptyset$, by 11, 12, the 'write-set included in read-set' model constraint, and 16.
- $\mathsf{Rd}' = \mathsf{Rda}'_{\#} \cup \mathsf{Rda}'_3$ and $\mathsf{Wr}' = \mathsf{Wra}'_{\#} \cup \mathsf{Wra}'_3$ hold by definition, 16 and 12.
- $\mathsf{Va}'_{\#} \setminus \mathsf{Wr}' = \mathsf{Va}'_3 \setminus \mathsf{Wr}'$ holds because:
  - $\mathsf{Va}'_{\#} \setminus \mathsf{Wr}' = (\mathsf{V}'_1 \cup \mathsf{V}'_2) \setminus \mathsf{Wr}'$ by definitions, claim 3, and claim 4.1. By claim 5 and claim 6, it is equal to $(\mathsf{V}_1 \cup \mathsf{V}_2) \setminus \mathsf{Wr}'$, which by 12 and 16 is $\mathsf{V} \setminus \mathsf{Wr}'$.
  - Moreover, $\mathsf{Va}'_3 \setminus \mathsf{Wr}' = \mathsf{V}'_3 \setminus \mathsf{Wr}'$ by definition. It is equal to $\mathsf{V}_3 \setminus \mathsf{Wr}'$ by claim 7, and to $\mathsf{V} \setminus \mathsf{Wr}'$ by 12 and 16.
- $\mathsf{V}' = (\mathsf{Va}'_{\#} \cap \mathsf{Wra}'_{\#}) \cup (\mathsf{Va}'_3 \cap \mathsf{Wra}'_3) \cup (\mathsf{Va}'_{\#} \cap \mathsf{Va}'_3)$ holds because:
  - $\mathsf{Va}'_{\#} \cap \mathsf{Wra}'_{\#} = ((\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2)) \cap (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2)$ by definition, 3, and 8. We get $(\mathsf{V}'_1 \cap \mathsf{Wr}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_1 \cap \mathsf{Wr}'_1 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{Wr}'_2)$. With elementary set theory simplifications and claim 8, we obtain $(\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2)$.
  - $\mathsf{Va}'_3 \cap \mathsf{Wra}'_3 = \mathsf{V}'_3 \cap \mathsf{Wr}'_3$.
  - $\mathsf{Va}'_{\#} \cap \mathsf{Va}'_3((\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2)) \cap \mathsf{V}'_3$ by definition, 3, and 8. We get $(\mathsf{V}'_1 \cap \mathsf{Wr}'_1 \cap \mathsf{V}'_3) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2 \cap \mathsf{V}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{V}'_3)$.

- The rhs quantity $(\mathsf{Va}'_{\#} \cap \mathsf{Wra}'_{\#}) \cup (\mathsf{Va}'_3 \cap \mathsf{Wra}'_3) \cup (\mathsf{Va}'_{\#} \cap \mathsf{Va}'_3)$ is then equal to $(\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{V}'_3)$.
- Moreover, by 14, we have $\mathsf{V}' = (\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_* \cap \mathsf{Wr}'_*) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_*)$.
- By 18, $\mathsf{V}'_* = (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2)$.
- By claim 1, $\mathsf{Wr}'_* = \mathsf{Wr}_*$, which by 16 is $\mathsf{Wr}_2 \cup \mathsf{Wr}_3$ which by 8 and 10 is $\mathsf{Wr}'_2 \cup \mathsf{Wr}'_3$. So $\mathsf{Wr}'_2 \subseteq \mathsf{Wr}'_*$ and $\mathsf{Wr}'_3 \subseteq \mathsf{Wr}'_*$.
- So $(\mathsf{V}'_* \cap \mathsf{Wr}'_*) = (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{Wr}'_*)$, with $\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{Wr}'_* = \mathsf{V}'_1 \cap \mathsf{V}'_2 \cap (\mathsf{Wr}'_2 \cup \mathsf{Wr}'_3)$ by 18, which is $(\mathsf{V}'_2 \cap \mathsf{V}'_3 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_2 \cap \mathsf{V}'_3 \cap \mathsf{Wr}'_3)$. So $(\mathsf{V}'_* \cap \mathsf{Wr}'_*) = (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3)$.
- Also $(\mathsf{V}'_1 \cap \mathsf{V}'_*) = (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{V}'_3)$, with the first two disjuncts included in $\mathsf{V}'_* \cap \mathsf{Wr}'_*$.
- The lhs quantity V' is then equal to $(\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_3 \cap \mathsf{Wr}'_3) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2 \cap \mathsf{V}'_3)$.

20 if and only if

  24 if and only if

    33 $\mathsf{Wra}_1 \cap \mathsf{Rda}_2 = \emptyset$, $\mathsf{Wra}_2 \cap \mathsf{Rda}_1 = \emptyset$. It holds, because:
    - $\mathsf{Wra}_1 \cap \mathsf{Rda}_2 = \mathsf{Wr}_1 \cap \mathsf{Rd}_2$ by definition. From 16, $\mathsf{Rd}_2 \subseteq \mathsf{Rd}_*$. From 11, $\mathsf{Wr}_1 \cap \mathsf{Rd}_* = \emptyset$. So $\mathsf{Wr}_1 \cap \mathsf{Rd}_2 = \emptyset$.
    - $\mathsf{Wra}_2 \cap \mathsf{Rda}_1 = \mathsf{Wr}_2 \cap \mathsf{Rd}_1$ by definition. From 16, $\mathsf{Wr}_2 \subseteq \mathsf{Wr}_*$. From 11, $\mathsf{Wr}_* \cap \mathsf{Rd}_1 = \emptyset$. So $\mathsf{Wr}_2 \cap \mathsf{Rd}_1 = \emptyset$.

    34 By definition.

    35 By definition, 33, 3, and 8.

    36 $\mathsf{Rda}'_{\#} = \mathsf{Rda}'_1 \cup \mathsf{Rda}'_2$, $\mathsf{Wra}'_{\#} = \mathsf{Wra}'_1 \cup \mathsf{Wra}'_2$, and $\mathsf{Va}'_{\#} = (\mathsf{Va}'_1 \cap \mathsf{Wra}'_1) \cup (\mathsf{Va}'_2 \cap \mathsf{Wra}'_2) \cup (\mathsf{Va}'_1 \cap \mathsf{Va}'_2)$ by definition. Also, $\mathsf{Va}'_1 \setminus \mathsf{Wra}'_{\#} = \mathsf{Va}'_2 \setminus \mathsf{Wra}'_{\#}$ holds because:
    - $\mathsf{Va}'_1 \setminus \mathsf{Wra}'_{\#} = \mathsf{V}'_1 \setminus (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2)$, which by 3 and 8 is equal to $\mathsf{V}_1 \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$, which by 12 is equal to $\mathsf{V} \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$.
    - Similarly, $\mathsf{Va}'_2 \setminus \mathsf{Wra}'_{\#}$ is equal to $\mathsf{V} \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$ by 8, 3, 16 and 12.

  25 By definition, and 2.

  26 By definition, 3, and 12.

  27 By definition, and 7.

  28 By definition, 8, 16, and 12.

21 $\mathsf{Rda}_{\#} = \mathsf{Rda}'_{\#}$ and $\mathsf{Wra}_{\#} = \mathsf{Wra}'_{\#}$ hold by definition, 3, and 8. Also by definition, 3, and 8, $\mathsf{Va}_{\#} \setminus \mathsf{Wra}_{\#} = \mathsf{Va}'_{\#} \setminus \mathsf{Wra}'_{\#}$ is equivalent to $\mathsf{V} \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2) = ((\mathsf{V}'_1 \cap \mathsf{Wr}'_1) \cup (\mathsf{V}'_2 \cap \mathsf{Wr}'_2) \cup (\mathsf{V}'_1 \cap \mathsf{V}'_2)) \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$. The right-hand-side simplifies into $(\mathsf{V}'_1 \cap \mathsf{V}'_2) \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$. Moreover, we have $\mathsf{V}'_1 \setminus \mathsf{Wr}'_1 = \mathsf{V} \setminus \mathsf{Wr}'_1$ (by 3 and 12) and $\mathsf{V}'_2 \setminus \mathsf{Wr}'_2 = \mathsf{V} \setminus \mathsf{Wr}'_2$ (by 8, 16, and 12). So we have $\mathsf{V}'_1 \setminus (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2) = \mathsf{V} \setminus (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2)$ and $\mathsf{V}'_2 \setminus (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2) = \mathsf{V} \setminus (\mathsf{Wr}'_1 \cup \mathsf{Wr}'_2)$. Hence, $(\mathsf{V}'_1 \cap \mathsf{V}'_2) \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$ simplifies into the left-hand-side $\mathsf{V} \setminus (\mathsf{Wr}_1 \cup \mathsf{Wr}_2)$.

22 By definition, and 9.

23 By definition, 10, 16, and 12.

Right to left. Suppose rhs. We define:

- $\mathsf{m}_* = \langle \mathsf{Rd}_*, \mathsf{Wr}_*, \mathsf{V}_* \rangle = \langle \mathsf{Rda}_2 \cup \mathsf{Rda}_3, \mathsf{Wra}_2 \cup \mathsf{Wra}_3, \mathsf{V} \rangle$,
- $\mathsf{m}_1 = \langle \mathsf{Rd}_1, \mathsf{Wr}_1, \mathsf{V}_1 \rangle = \mathsf{a}_1 = \langle \mathsf{Rda}_1, \mathsf{Wra}_1, \mathsf{Va} \rangle$,

– $\mathsf{m}_2 = \langle \mathsf{Rd}_2, \mathsf{Wr}_2, \mathsf{V}_2 \rangle = \mathsf{a}_2 = \langle \mathsf{Rda}_2, \mathsf{Wra}_2, \mathsf{Va} \rangle$,
– $\mathsf{m}_3 = \langle \mathsf{Rd}_3, \mathsf{Wr}_3, \mathsf{V}_3 \rangle = \mathsf{a}_3 = \langle \mathsf{Rda}_3, \mathsf{Wra}_3, \mathsf{Va} \rangle$,
– $\mathsf{m}'_* = \langle \mathsf{Rd}'_*, \mathsf{Wr}'_*, \mathsf{V}'_* \rangle = \langle \mathsf{Rda}'_2 \cup \mathsf{Rda}'_3, \mathsf{Wra}'_2 \cup \mathsf{Wra}'_3, (\mathsf{Va}'_2 \cap \mathsf{Wra}_2) \cup (\mathsf{Va}'_3 \cap \mathsf{Wra}_3) \cup (\mathsf{Va}'_2 \cap \mathsf{Va}'_3) \rangle$,
– $\mathsf{m}'_1 = \langle \mathsf{Rd}'_1, \mathsf{Wr}'_1, \mathsf{V}'_1 \rangle = \mathsf{a}'_1 = \langle \mathsf{Rda}'_1, \mathsf{Wra}'_1, \mathsf{Va}'_1 \rangle$,
– $\mathsf{m}'_2 = \langle \mathsf{Rd}'_2, \mathsf{Wr}'_2, \mathsf{V}'_2 \rangle = \mathsf{a}'_2 = \langle \mathsf{Rda}'_2, \mathsf{Wra}'_2, \mathsf{Va}'_2 \rangle$,
– $\mathsf{m}'_3 = \langle \mathsf{Rd}'_3, \mathsf{Wr}'_3, \mathsf{V}'_3 \rangle = \mathsf{a}'_3 = \langle \mathsf{Rda}'_3, \mathsf{Wra}'_3, \mathsf{Va}'_3 \rangle$.

We must show that these models satisfy all the properties from 1 to 18. This is done routinely, analogously to the proof of the left-to-right direction above.

References

[Abr80] K.R. Abrahamson. *Decidability and expressivity of logics of processes*. PhD thesis, Department of Computer Science, University of Washington, Seattle, WA, 1980. Report 80-08-01.

[BB18] Philippe Balbiani and Joseph Boudou. Iteration-free PDL with storing, recovering and parallel composition: a complete axiomatization. *Journal of Logic and Computation*, 28(4):705–731, 2018.

[BdFV11] Mario R. F. Benevides, Renata P. de Freitas, and Jorge Petrucio Viana. Propositional dynamic logic with storing, recovering and parallel composition. *Electronic Notes in Theoretical Computer Science*, 269:95–107, 2011.

[BF97] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BHST14] Philippe Balbiani, Andreas Herzig, François Schwarzentruber, and Nicolas Troquard. DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. *CoRR*, abs/1411.7825, 2014.

[BHT13] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments: a well-behaved variant of PDL. In O. Kupferman, editor, *Logic in Computer Science (LICS)*. IEEE, 2013.

[BHT19] Joseph Boudou, Andreas Herzig, and Nicolas Troquard. Resource separation in dynamic logic of propositional assignments. In Luís Soares Barbosa and Alexandru Baltag, editors, *Dynamic Logic. New Trends and Applications - Second International Workshop, DaLí 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, volume 12005 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2019.

[BO16] Stephen Brookes and Peter W. O'Hearn. Concurrent separation logic. *SIGLOG News*, 3(3):47–65, 2016.

[Bou16] Joseph Boudou. Complexity optimal decision procedure for a propositional dynamic logic with parallel composition. In Nicola Olivetti and Ashish Tiwari, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 9706 of *Lecture Notes in Computer Science*, pages 373–388. Springer, 2016.

[Bro04] Stephen D. Brookes. A semantics for concurrent separation logic. In Gardner and Yoshida [GY04], pages 16–34.

[BT14] Philippe Balbiani and Tinko Tinchev. Definability and computability for PRSPDL. In Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors, *Advances in Modal Logic 10*, pages 16–33. College Publications, 2014.

[BV03] Philippe Balbiani and Dimiter Vakarelov. PDL with intersection of programs: A complete axiomatization. *Journal of Applied Non-Classical Logics*, 13(3-4):231–276, 2003.

[CHM+16] Martin C. Cooper, Andreas Herzig, Faustine Maffre, Frédéric Maris, and Pierre Régnier. A simple account of multi-agent epistemic planning. In G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *European Conference on Artificial Intelligence (ECAI)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 193–201. IOS Press, 2016.

[CS15] Tristan Charrier and François Schwarzentruber. Arbitrary public announcement logic with mental programs. In G. Weiss, P. Yolum, R.H. Bordini, and E. Elkind, editors, *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1471–1479. ACM, 2015.

[CS17] Tristan Charrier and François Schwarzentruber. A succinct language for dynamic epistemic logic. In K. Larson, M. Winikoff, S. Das, and E.H. Durfee, editors, *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 123–131. ACM, 2017.

[DHS05] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In D. Hutter and M. Ullmann, editors, *Security in Pervasive Computing*, pages 193–209, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[FHR19] Guillaume Feuillade, Andreas Herzig, and Christos Rantsoudis. A Dynamic Logic Account of Active Integrity Constraints. *Fundamenta Informaticae*, 169(3):179–210, 2019.

[Gol92] Robert Goldblatt. Parallel action: Concurrent dynamic logic with independent modalities. *Studia Logica*, 51(3/4):551–578, 1992.

[GY04] Philippa Gardner and Nobuko Yoshida, editors. *CONCUR 2004 - Concurrency Theory, 15th Int. Conf., London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*. Springer, 2004.

[Her13] Andreas Herzig. A simple separation logic. In L. Libkin, U. Kohlenbach, and R.J.G.B. de Queiroz, editors, *Workshop on Logic, Language, Information and Computation (WoLLIC)*, volume 8071 of *Lecture Notes in Computer Science*, pages 168–178. Springer, 2013.

[Her14] 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, *Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2014.

[HMNDBW14] Andreas Herzig, Viviane Menezes, Leliane Nunes De Barros, and Renata Wassermann. On the revision of planning tasks. In T. Schaub, editor, *European Conference on Artificial Intelligence (ECAI)*, August 2014.

[HMV19] Andreas Herzig, Frédéric Maris, and Julien Vianey. Dynamic logic of parallel propositional assignments and its applications to planning. In S. Kraus, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5576–5582. ijcai.org, 2019.

[MS96] Alain J. Mayer and Larry J. Stockmeyer. The complexity of PDL with interleaving. *Theoretical Computer Science*, 161(1&2):109–122, 1996.

[NGH18] Arianna Novaro, Umberto Grandi, and Andreas Herzig. Judgment aggregation in dynamic logic of propositional assignments. *Journal of Logic and Computation*, 28(7):1471–1498, 2018.

[O'H04] Peter W. O'Hearn. Resources, concurrency and local reasoning. In Gardner and Yoshida [GY04], pages 49–67.

[Pel87] David Peleg. Concurrent dynamic logic. *Journal of the ACM*, 34(2):450–479, 1987.

[Rey02] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science (LICS)*, pages 55–74. IEEE Computer Society, 2002.

[SG16] Christoph Scheben and Simon Greiner. Information flow analysis. In W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, editors, *Deductive Software Verification - The KeY Book - From Theory to Practice*, volume 10001 of *Lecture Notes in Computer Science*, pages 453–471. Springer, 2016.

[vDvdHK07] Hans P. van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Kluwer Academic Publishers, 2007.