Dynamic Logic of Parallel Propositional Assignments and its Applications to Planning

Andreas Herzig[1] , **Frédéric Maris**[2] and **Julien Vianey**[2]
[1]IRIT-CNRS
[2]IRIT-Univ. Toulouse
{herzig,maris,julien.vianey}@irit.fr

Abstract

We introduce a dynamic logic with parallel composition and two kinds of nondeterministic composition, exclusive and inclusive. We show PSPACE completeness of both the model checking and the satisfiability problem and apply our logic to sequential and parallel classical planning where actions have conditional effects.

1 Introduction

Many authors have investigated how planning problems and their solutions can be represented in Propositional Dynamic Logic PDL, including work on conformant and contingent planning and planning with epistemic actions [Spalazzi and Traverso, 2000; Andersen *et al.*, 2012; Li *et al.*, 2017; Bolander *et al.*, 2018; Cong *et al.*, 2018]. However and to the best of our knowledge, it has not been investigated yet how planning with concurrent actions can be captured in dynamic logic. Probably the reason for that is that there is no consensus on the semantics of parallel actions in the PDL community: there are proposals interpreting parallel composition as interleaving [Mayer and Stockmeyer, 1996; Benevides and Schechter, 2014], as intersection [Balbiani and Vakarelov, 2003], and as relations between states and sets of states [Peleg, 1987; Goldblatt, 1992]. There are also extensions with modalities from resource separation logics [Balbiani and Boudou, 2018; Benevides *et al.*, 2011; Veloso *et al.*, 2014].

We here take a different route and build on a simple version of dynamic logic where atomic programs are assignments of formulas to propositional variables. Dynamic Logic of Propositional Assignments DL-PA has numerous applications in knowledge representation [Herzig, 2014] in particular classical planning [Herzig *et al.*, 2014], and is considerably simpler than PDL. In particular, complexity is much lower: satisfiability checking is in PSPACE. We add an operator of parallel composition to DL-PA, as well as an operator of inclusive nondeterministic composition (as opposed to PDL's exclusive nondeterministic composition).

In order to solve planning tasks by parallel plans several notions of interference have been proposed [Knoblock, 1994; Dimopoulos *et al.*, 1997]. We choose the framework of independent parallel actions introduced in the planner GRAPHPLAN [Blum and Furst, 1997] where two actions interfere if one deletes a precondition or an effect of the other. This rather restrictive definition is used in most approaches to parallel classical planning. Non-interfering actions can be arranged in any sequential order with exactly the same outcome.

The paper is organised as follows. We extend DL-PA to DL-PPA in Section 2 and show in Section 3 that complexity stays in PSPACE. In Section 4 we show how sequential and parallel planning as well as their bounded versions can be polynomially translated to DL-PPA. Section 5 concludes.

2 DL-PA and DL-PPA

Our Dynamic Logic of Parallel Propositional Assignments DL-PPA extends Dynamic Logic of Propositional Assignments DL-PA by two program operators: an operator of parallel composition $\sqcap$ and a new operator of nondeterministic composition $\sqcup$. The distinction between $\sqcup$ and the standard operator of nondeterministic composition $\cup$ is similar to that between inclusive and exclusive disjunction: the interpretation of $\pi_1 \cup \pi_2$ is "do either π_1 or π_2", while that of $\pi_1 \sqcup \pi_2$ is "do either π_1, or π_2, *or both*". We call the former *exclusive* nondeterministic choice and the latter *inclusive* nondeterministic choice. The interpretation of parallel composition $\pi_1 \sqcap \pi_2$ is that both programs are executed on local copies of the variables, and then the resulting state is obtained by merging these local states: $\pi_1 \sqcap \pi_2$ fails if two assigned variables get assigned different truth values by π_1 and π_2; otherwise new truth values override old ones.

2.1 Language

The language of DL-PPA is built from a countably infinite set of propositional variables $\mathbb{P}$. Programs π and formulas φ are defined by the grammar

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

where p ranges over $\mathbb{P}$. The formula $\langle\pi\rangle\varphi$ reads "there is a possible execution of π such that φ is true afterwards". The program $p{\leftarrow}\varphi$ assigns the truth value of φ to p; for example, $p{\leftarrow}\neg p$ swaps the truth value of p. $\varphi?$ tests that φ is true (failing when φ is false). $\pi_1;\pi_2$ executes π_1 and π_2 in sequence. $\pi_1 \cup \pi_2$ nondeterministically chooses between executing either π_1 or π_2; $\pi_1 \sqcup \pi_2$ nondeterministically chooses between executing either π_1, or π_2, or both. $\pi_1 \sqcap \pi_2$ is the parallel composition of π_1 and π_2. The set of all formulas is $Fml_{\text{DL-PPA}}$.

The language of DL-PA is the fragment of DL-PPA without $\sqcup$ and $\sqcap$.

The set of propositional variables occurring in a formula φ is noted $\mathbb{P}_\varphi$; similarly, the set of variables occurring in a program π is noted $\mathbb{P}_\pi$. For example, $\mathbb{P}_{p \leftarrow q \vee r} = \{p, q, r\}$.

The *length* of formulas and programs is the number of symbols required to write them down, excluding parentheses and considering that the length of propositional variables is 1. We note them $\ell(\varphi)$ and $\ell(\pi)$.

We use standard abbreviations such as $\top$ and $\varphi \to \psi$, plus n-times iterations of programs, recursively defined by $\pi^{\leq 0} = \top?$ and $\pi^{\leq n+1} = \top? \cup (\pi; \pi^{\leq n})$. For $P = \{p_1, \ldots, p_n\}$, $;_{p_i \in P} p_i \leftarrow \varphi_i$ is a shorthand for $p_1 \leftarrow \varphi_1; \cdots; p_n \leftarrow \varphi_n$. We identify it with $\top?$ if $P = \emptyset$. We have to be careful here because sequential composition ';' is not commutative; e.g., the interpretation of $p \leftarrow q; q \leftarrow p$ will differ from that of $q \leftarrow p; p \leftarrow q$. We will make sure each time that the order does not matter.

2.2 Semantics of DL-PPA

Semantics is in terms of valuations, alias states, which are subsets of $\mathbb{P}$. So the set of all valuations is $2^{\mathbb{P}}$. We use $V, V', W, U, \ldots$ for valuations.

Formulas and programs are interpreted by mutual recursion. In DL-PA, the interpretation of a formula φ was a set of valuations $\|\varphi\|_{\text{DL-PA}} \subseteq 2^{\mathbb{P}}$ and the interpretation of a program π is a binary relation on valuations $\|\pi\|_{\text{DL-PA}} \subseteq 2^{\mathbb{P}} \times 2^{\mathbb{P}}$. We recall the main clauses:

$$
\begin{aligned}
\|p\|_{\text{DL-PA}} &= \{V \ : \ p \in V\} \\
\|\langle\pi\rangle\varphi\|_{\text{DL-PA}} &= \{V \ : \ \text{there is } V' \text{ such that } (V, V') \in \|\pi\|_{\text{DL-PA}} \\
&\qquad\qquad \text{and } V' \in \|\varphi\|_{\text{DL-PA}}\} \\
\|p \leftarrow \varphi\|_{\text{DL-PA}} &= \{(V, V \cup \{p\}) \ : \ V \in \|\varphi\|_{\text{DL-PA}}\} \cup \\
&\qquad \{(V, V \setminus \{p\}) \ : \ V \notin \|\varphi\|_{\text{DL-PA}}\} \\
\|\varphi?\|_{\text{DL-PA}} &= \{(V, V) \ : \ V \in \|\varphi\|_{\text{DL-PA}}\} \\
\|\pi; \pi'\|_{\text{DL-PA}} &= \|\pi\|_{\text{DL-PA}} \circ \|\pi'\|_{\text{DL-PA}} \\
\|\pi \cup \pi'\|_{\text{DL-PA}} &= \|\pi\|_{\text{DL-PA}} \cup \|\pi'\|_{\text{DL-PA}} \\
\|\pi^*\|_{\text{DL-PA}} &= (\|\pi\|_{\text{DL-PA}})^* \ = \ \bigcup_{k \in \mathbb{N}_0} (\|\pi\|_{\text{DL-PA}})^k
\end{aligned}
$$

In contrast, in DL-PPA the interpretation of π is a ternary relation on the set of valuations $\|\pi\| \subseteq 2^{\mathbb{P}} \times 2^{\mathbb{P}} \times 2^{\mathbb{P}}$. When $(V, U, W) \in \|\pi\|$ then there is an execution of π from state V to state U assigning the variables in W. The definition is again by mutual recursion and the main clauses are given in Figure **??**; the others are standard.

Let us comment on the clauses that are new w.r.t. DL-PA.

The semantics of the assignment $p \leftarrow \varphi$ is: p is made true if φ is true and is made false if φ is false, and in both cases the set of assigned variables is the singleton $\{p\}$.

The semantics of parallel composition $\pi_1 \sqcap \pi_2$ is that each subprogram π_i is executed locally; then it is checked whether the modifications (in terms of assigned variables) are compatible: this is the case when all variables that are assigned by both subprograms (namely the variables in $W_1 \cap W_2$) get assigned the same truth value. If this is not the case then $\pi_1 \sqcap \pi_2$ fails; otherwise the resulting valuation U is computed by putting together

- the unchanged part of V, i.e., $V \setminus (W_1 \cup W_2)$;
- the updates of π_1, i.e., $U_1 \cap W_1$;
- the updates of π_2, i.e., $U_2 \cap W_2$.

Moreover, the set of variables W assigned by $\pi_1 \sqcap \pi_2$ is the union of those assigned by its subprograms.

The semantics of inclusive nondeterministic composition $\pi_1 \sqcup \pi_2$ is, as announced, the exclusive nondeterministic composition of the three programs π_1, π_2 and $\pi_1 \sqcap \pi_2$.

Here are some examples of interpretations of programs:

$$
\begin{aligned}
\|p \leftarrow \bot\| &= \{(V, V \setminus \{p\}, \{p\}) \ : \ V \subseteq \mathbb{P}\} \\
\|\top? \sqcap p \leftarrow \bot\| &= \|p \leftarrow \bot\| \\
\|p \leftarrow \top \sqcap p \leftarrow \bot\| &= \emptyset \\
\|p \leftarrow \top \sqcap q \leftarrow \bot\| &= \{(V, (V \setminus \{q\}) \cup \{p\}, \{p, q\}) \ : \ V \subseteq \mathbb{P}\} \\
\|p \leftarrow p \sqcap p \leftarrow \bot\| &= \{(V, V, \{p\}) \ : \ V \subseteq \mathbb{P} \text{ and } p \notin V\}
\end{aligned}
$$

Proposition 1. *Let π be a* DL-PPA *program and let $(V, U, W) \in \|\pi\|$. Then:*

- *$W \subseteq \mathbb{P}_\pi$;*
- *$V \setminus U \subseteq W$ and $U \setminus V \subseteq W$;*
- *If $\mathbb{P}_\pi \subseteq P$ then $V \subseteq P$ implies $U \subseteq P$.*

The first item can be restricted a bit further: W is a subset of the variables occurring on the left-hand side of assignments of π.

The next result says that the variables not occurring in a formula or a program do not matter in their interpretation.

Proposition 2. *Let φ be a* DL-PPA *formula and π a* DL-PPA *program. Let P be a set of variables none of which occurs in φ or π, i.e., P is such that $P \cap \mathbb{P}_\varphi = P \cap \mathbb{P}_\pi = \emptyset$. Then*

- *$V \cup P \in \|\varphi\|$ iff $V \setminus P \in \|\varphi\|$;*
- *$(V \cup P, U \cup P, W) \in \|\pi\|$ iff $(V \setminus P, U \setminus P, W) \in \|\pi\|$.*

A formula φ is *satisfiable* if and only if $\|\varphi\| \neq \emptyset$. Thanks to Propositions 1 and 2, when checking validity or satisfiability of a formula φ it suffices to check triples (V, U, W) whose elements are all subsets of $\mathbb{P}_\varphi$.

When the variables of π_1 and π_2 are disjoint then they can be sequentialised:

Proposition 3. *Let π_1 and π_2 be two programs such that $\mathbb{P}_{\pi_1} \cap \mathbb{P}_{\pi_2} = \emptyset$. Then $\|\pi_1 \sqcap \pi_2\| = \|\pi_1; \pi_2\| = \|\pi_2; \pi_1\|$.*

Just as in DL-PA [Balbiani *et al.*, 2014], the Kleene star can be eliminated in DL-PPA: intuitively, when there exists a run of π^* going from V to U then it is possible to go from V to U in no more than $2^{|\mathbb{P}_\pi|}$ iterations of π. (Indeed, if there is a longer run, it necessarily goes through the same valuation twice and can be shortened.)

Proposition 4. *For all* DL-PPA *programs π, $\|\pi^*\| = \|\pi^{\leq 2^{|\mathbb{P}_\pi|}}\|$.*

2.3 Counters

In DL-PA one can implement k-bit counters (see e.g. [Balbiani *et al.*, 2013, Section II.C]), and this transfers to DL-PPA. The encoding of a binary number of length k requires a vector of $\lceil \log k \rceil$ propositional variables. We represent the state of the counter by a formula ct, which stands for the conjunction of these $\lceil \log k \rceil$ variables or negations thereof. Furthermore we use the following abbreviations:

$$
\begin{aligned}
\|p\| &= \{V \ : \ p \in V\} \\
\|\langle\pi\rangle\varphi\| &= \{V \ : \ \text{there are } U, W \text{ such that } (V, U, W) \in \|\pi\| \text{ and } U \in \|\varphi\|\} \\
\|p{\leftarrow}\varphi\| &= \{(V, V{\cup}\{p\}, \{p\}) \ : \ V \in \|\varphi\|\} \cup \{(V, V{\setminus}\{p\}, \{p\}) \ : \ V \notin \|\varphi\|\} \\
\|\varphi?\| &= \{(V, V, \emptyset) \ : \ V \in \|\varphi\|\} \\
\|\pi_1;\pi_2\| &= \{(V, U, W) \ : \ \text{there are } U_1, W_1, W_2 \text{ such that } (V, U_1, W_1) \in \|\pi_1\|, (U_1, U, W_2) \in \|\pi_2\| \text{ and } W = W_1 \cup W_2\} \\
\|\pi_1 \cup \pi_2\| &= \|\pi_1\| \cup \|\pi_2\| \\
\|\pi_1 \sqcup \pi_2\| &= \|\pi_1\| \cup \|\pi_2\| \cup \|\pi_1 \sqcap \pi_2\| \\
\|\pi_1 \sqcap \pi_2\| &= \left\{(V, U, W) \ : \ \begin{array}{l} \text{there are } U_1, W_1, U_2, W_2 \text{ such that } (V, U_1, W_1) \in \|\pi_1\|, (V, U_2, W_2) \in \|\pi_2\|, \\ W_1 \cap W_2 \cap U_1 = W_1 \cap W_2 \cap U_2, U = (V{\setminus}W) \cup (U_1{\cap}W_1) \cup (U_2{\cap}W_2), \text{ and } W = W_1 \cup W_2 \end{array}\right\} \\
\|\pi^*\| &= \bigcup_{k \in \mathbb{N}_0} \|\pi\|^k
\end{aligned}
$$

Figure 1: Interpretation of DL-PPA programs

- $ct{\leftarrow}0$ is a program that resets ct: all variables of the counter are set to false;
- For every integer i, $ct{=}i$ is a formula that is true if and only if the value encoded by ct is i;
- $ct{+}{+}$ increments the value of ct.

We suppose that variables used in a counter do not occur elsewhere, so that they do not interfere with other programs.

Counters allow us to formulate in a more compact way our abbreviation $\pi^{\leq k}$ whose expansion has length exponential in k: using a $\lceil \log k \rceil$-bit counter we can identify $\pi^{\leq k}$ with

$$ct{\leftarrow}0; (\neg ct{=}k\,?; \pi; ct{+}{+})^*$$

whose length is linear in $\ell(\pi) + \log k$. Indeed, although these two programs do not have the same interpretation, they behave the same as far as the variables of π are concerned.

Proposition 5. *For valuations $V, U, W \subseteq \mathbb{P}_\pi$, we have $(V, U, W) \in \|\pi^{\leq k}\|$ if and only if*

$$(V, U{\cup}U', W{\cup}W') \in \|ct{\leftarrow}0; (\neg ct{=}k\,?; \pi; ct{+}{+})^*\|$$

for some subsets U' and W' of the set of counter variables.

3 Reduction from DL-PPA to DL-PA

We are now going to give a polynomial reduction of formulas and programs of DL-PPA to DL-PA. Our translation eliminates $\sqcap$ and $\sqcup$ thanks to the introduction of (several kinds of) fresh copies of propositional variables.

3.1 Copying Variables

First, the copy δ_p of p stores that p has been assigned; it will allow us to simulate the third component W of the interpretation of programs that keeps trace of assigned variables.

Second, when translating parallel composition $\pi_1 \sqcap \pi_2$ we sequentialise π_1 and π_2, and in order to safely do so we let each of them work on local copies of its variables p, respectively noted $(p)^1$ and $(p)^2$. Similarly, we also use this technique when translating inclusive nondeterministic composition $\pi_1 \sqcup \pi_2$. (Observe that we cannot simply take over the interpretation of $\pi_1 \sqcup \pi_2$ as $\pi_1 \cup \pi_2 \cup (\pi_1 \sqcap \pi_2)$ because this can generate exponential growth.)

It is convenient to extend copying from variables to sets of variables and to programs. We associate to each set of variables $P \subseteq \mathbb{P}$ the copies $(P)^1 = \{(p)^1 \ : \ p \in P\}$ and $(P)^2 = \{(p)^2 \ : \ p \in P\}$; and we map programs π to $(\pi)^1$ by replacing all variables p of π by $(p)^1$; similarly, we map π to $(\pi)^2$. For example, $(p{\leftarrow}q \wedge r)^2 = (p)^2{\leftarrow}(q)^2 \wedge (r)^2$.

Proposition 6. *Let $V, U, W \subseteq \mathbb{P}_\pi$ and let i be either 1 or 2. Then:*

- $(V, U, W) \in \|\pi\|$ *iff* $((V)^i, (U)^i, (W)^i) \in \|(\pi)^i\|$;
- $V \in \|\varphi\|$ *iff* $(V)^i \in \|(\varphi)^i\|$.

3.2 The Translation

We are going to show that $\sqcap$ and $\sqcup$ can be polynomially eliminated from DL-PPA formulas. The resulting formula is in the language of DL-PA, which will allow us to transfer complexity results. The recursive definition of the translation of formulas and programs is given in Figure **??**. It is homomorphic for all the program operators and logical operators of PDL, so we only comment the non-trivial cases: assignments, parallel composition and inclusive nondeterministic composition.

The translation of $p{\leftarrow}\varphi$ stores that p has been assigned by means of a fresh variable δ_p.

The translation of $\pi_1 \sqcap \pi_2$ relies on the introduction of fresh copies $(p)^1$ and $(p)^2$ of propositional variables p: as announced in Section 3.1, the program $(\pi_1)^1$ is obtained by substituting each p in π_1 by its copy $(\pi_1)^1$; and similarly, $(\pi_2)^2$ is obtained by substituting each p in π_2 by $(\pi_2)^2$. The idea is to execute π_1 and π_2 on local copies of propositional variables and then check whether the two results can be merged: did the variables that were assigned by both subprograms modify their copies in the same way, i.e., is it the case that $(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)$ for every $p \in \mathbb{P}_{\pi_1} \cap \mathbb{P}_{\pi_2}$? If so then the values of the relevant copies are copied back to each p, where we the value of p is kept unchanged when none of the copies has been assigned. Moreover, the δ_p are updated in order to keep track of the modification of the variables p. Finally, the auxiliary variables $(p)^1$, $(p)^2$, $\delta_{(p)^1}$, $\delta_{(p)^2}$ are set to false.

In the translation of inclusive nondeterministic composition $\pi_1 \sqcup \pi_2$, the check of compatibility followed by copying

back is replaced by an exclusive nondeterministic composition of either copying back the values computed by π_1, or those computed by π_2, or checking compatibility and copying back the values computed by $\pi_1 \sqcap \pi_2$.

Example 1. *Consider the program* $p{\leftarrow}\top \sqcap p{\leftarrow}\bot$. *Its translation* $f\,(p{\leftarrow}\top \sqcap p{\leftarrow}\bot)$ *is:*

$$(p)^1{\leftarrow}p; (p)^2{\leftarrow}p; (p)^1{\leftarrow}\top; \delta_{(p)^1}{\leftarrow}\top; (p)^2{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\top;$$
$$(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)?;$$
$$p{\leftarrow}(p{\wedge}\neg\delta_{(p)^1}{\wedge}\neg\delta_{(p)^1}) \vee ((p)^1{\wedge}\delta_{(p)^1}) \vee ((p)^2{\wedge}\delta_{(p)^2});$$
$$\delta_p{\leftarrow}\delta_p \vee \delta_{(p)^1} \vee \delta_{(p)^2};$$
$$(p)^1{\leftarrow}\bot; \delta_{(p)^1}{\leftarrow}\bot; (p)^2{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\bot$$

Just as the original program, the translated program has an empty interpretation because after setting $\delta_{(p)^2}$ *and* $\delta_{(p)^1}$ *to true and* $(p)^1$ *to true and* $(p)^2$ *to false, the test* $(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)?$ *is going to fail.*

3.3 Correctness of the Translation

For a set $W \subseteq \mathbb{P}$, let $\delta_W = \{\delta_p \ : \ p \in W\}$.

Proposition 7. *For every formula* φ, $\|\varphi\| = \|f\,(\varphi)\|_{\mathsf{DL\text{-}PA}}$.

Proof. Let $\mathbb{P}^0$ be a finite set of variables such that $\mathbb{P}_\varphi \subseteq \mathbb{P}^0$ and and such that for every $p \in \mathbb{P}_\varphi$, none of the copies $(p)^1$, $(p)^2$, δ_p is in $\mathbb{P}^0$. We prove by simultaneous induction:

1. $V \in \|\psi\|$ iff $V{\cup}\delta_P \in \|f\,(\psi)\|_{\mathsf{DL\text{-}PA}}$, for every $V \subseteq \mathbb{P}^0$ and every $P \subseteq \mathbb{P}$;
2. $(V, U, W) \in \|\pi\|$ iff $(V{\cup}\delta_P, U{\cup}\delta_W{\cup}\delta_P) \in \|f\,(\pi)\|_{\mathsf{DL\text{-}PA}}$, for every $V \subseteq \mathbb{P}^0$ and every $U, W, P \subseteq \mathbb{P}$.

For the case of a formula of the form $\langle\pi\rangle\varphi$ we use Propositions 1 and 2. For the cases program operators we use Proposition 1. For parallel composition and for inclusive nondeterministic composition we moreover use Proposition 6.

Then the result follows by Proposition 2. □

Proposition 8. *The length* $\ell(f\,(\varphi))$ *of the translation of a* DL-PPA *formula* φ *is polynomial in the length* $\ell(\varphi)$ *of* φ.

3.4 Complexity Results

Lower bounds for DL-PPA reasoning tasks follow from those for its fragment DL-PA. In order to establish upper bounds we appeal to the translation.

Theorem 1. *The* DL-PPA *model checking problem of deciding, given* V *and* φ, *whether* $V \in \|\varphi\|$ *is PSPACE-complete.*

Proof. The lower bound is due to PSPACE hardness of DL-PA model checking [Herzig *et al.*, 2011]. The upper bound is obtained by Proposition 7 and Proposition 8 thanks to PSPACE membership of DL-PA [Balbiani *et al.*, 2014]. □

Theorem 2. *The* DL-PPA *satisfiability checking problem is PSPACE-complete.*

Proof. The lower bound is due to PSPACE hardness of DL-PA satisfiability checking [Herzig *et al.*, 2011]. The upper bound follows from PSPACE membership of DL-PPA model checking because satisfiability checking can be polynomially reduced to model checking: a DL-PPA formula φ is satisfiable if and only if $\emptyset \in \|\langle;_{p\in\mathbb{P}_\varphi}(p{\leftarrow}\top \cup p{\leftarrow}\bot)\rangle\varphi\|$. □

4 DL-PPA Applied to Automated Planning

In this section, we formally define actions and sequential and parallel planning tasks within our framework DL-PPA. Then we show that DL-PPA model checking can be used to test the solvability of such tasks.

4.1 Sequential Planning with Conditional Effects

A planning task is a triple $(Act, V_0, Goal)$ where:

- Act is a set of *conditional actions*;
- $V_0 \subseteq \mathbb{P}$ is a valuation (the *initial state*);
- $Goal \in Fml_{\mathsf{DL\text{-}PPA}}$ is the *goal*.

A conditional action is a pair $\mathsf{a} = (pre(\mathsf{a}), eff(\mathsf{a}))$ where:

- $pre(\mathsf{a}) \in Fml_{\mathsf{DL\text{-}PPA}}$ is the *precondition* of a;
- $eff(\mathsf{a}) \in Fml_{\mathsf{DL\text{-}PPA}} \times 2^{\mathbb{P}} \times 2^{\mathbb{P}}$ is a set of triples ce of the form $(cnd(ce), ceff^+(ce), ceff^-(ce))$, the *conditional effects* of a, where $cnd(ce)$ is a DL-PPA formula (the condition) and $ceff^+(ce)$ and $ceff^-(ce)$ are sets of variables that are respectively added and deleted by a if condition ce is true.

We say that a state V is *reachable by a sequential plan* from a state V_0 via a set of conditional actions Act if there is a *sequential plan*, that is, a sequence of actions $\langle \mathsf{a}_1, \ldots, \mathsf{a}_m\rangle$ from Act, and a sequence of states $\langle V_0, \ldots, V_m\rangle$ with $m \geq 0$ such that $V = V_m$ and $\tau_{\mathsf{a}_k}(V_{k-1}) = V_k$ for every k such that $1 \leq k \leq m$. A planning task is *solvable by a sequential plan* if there is at least one state $V \in \|Goal\|$ such that V is reachable by a sequential plan from V_0 via Act; otherwise it is unsolvable by a sequential plan.

A conditional action a determines a partial function τ_{a} from $2^{\mathbb{P}}$ to $2^{\mathbb{P}}$ as follows. First, $\tau_{\mathsf{a}}(V)$ is defined (and therefore action a is executable in state V) if and only if

- $V \in \|pre(\mathsf{a})\|$ and
- $ceff^+(ce_1) \cap ceff^-(ce_2) = \emptyset$ for all $ce_1, ce_2 \in eff(\mathsf{a})$ such that $V \in \|cnd(ce_1) \wedge cnd(ce_2)\|$.

Second, when τ_{a} is defined in V then:

$$\tau_{\mathsf{a}}(V) = \left(V \setminus \bigcup_{\substack{ce\in eff(\mathsf{a})\\ V\in\|cnd(ce)\|}} ceff^-(ce)\right) \cup \bigcup_{\substack{ce\in eff(\mathsf{a})\\ V\in\|cnd(ce)\|}} ceff^+(ce)$$

We associate to action a the DL-PPA program $\mathsf{exeAct}(\mathsf{a})$:

$$\mathsf{exeAct}(\mathsf{a}) = pre(\mathsf{a})? \sqcap \bigsqcap_{ce\in eff(\mathsf{a})} \left(\begin{array}{l}\neg cnd(ce)?\\ \cup \left(\begin{array}{l} cnd(ce)?\\ \sqcap \bigsqcap_{p\in ceff^+(ce)} p{\leftarrow}\top\\ \sqcap \bigsqcap_{q\in ceff^-(ce)} q{\leftarrow}\bot\end{array}\right)\end{array}\right)$$

The program $\mathsf{exeAct}(\mathsf{a})$ behaves like a:

Lemma 1. *For every action* $\mathsf{a} \in Act$:

1. τ_{a} *is defined in* V *iff there are* U, W *such that* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{a})\|$;
2. *If* τ_{a} *is defined in* V *then* $\tau_{\mathsf{a}}(V) = U$ *iff* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{a})\|$ *for some* W.

$$
\begin{aligned}
f(p) &= p & f(p{\leftarrow}\varphi) &= p{\leftarrow}f(\varphi);\delta_p{\leftarrow}\top\\
f(\bot) &= \bot & f(\varphi?) &= f(\varphi)?\\
f(\neg\varphi) &= \neg f(\varphi) & f(\pi_1;\pi_2) &= f(\pi_1);f(\pi_2)\\
f(\varphi_1\vee\varphi_2) &= f(\varphi_1)\vee f(\varphi_2) & f(\pi_1\cup\pi_2) &= f(\pi_1)\cup f(\pi_2)\\
f(\langle\pi\rangle\varphi) &= \langle f(\pi)\rangle f(\varphi) & f(\pi^*) &= (f(\pi))^*
\end{aligned}
$$

$$
\begin{aligned}
f(\pi_1\sqcap\pi_2) = {} & \Big(;_{p\in\mathbb{P}_{\pi_1}}(p)^1{\leftarrow}p\Big);\Big(;_{p\in\mathbb{P}_{\pi_2}}(p)^2{\leftarrow}p\Big);f\Big((\pi_1)^1\Big);f\Big((\pi_2)^2\Big);\\
& \bigwedge_{p\in\mathbb{P}_{\pi_1}\cap\mathbb{P}_{\pi_2}}((\delta_{(p)^1}\wedge\delta_{(p)^2})\rightarrow((p)^1\leftrightarrow(p)^2))?;\\
& ;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}}\Big(p{\leftarrow}(p\wedge\neg\delta_{(p)^1}\wedge\neg\delta_{(p)^1})\vee((p)^1\wedge\delta_{(p)^1})\vee((p)^2\wedge\delta_{(p)^2})\ ;\ \delta_p{\leftarrow}\delta_p\vee\delta_{(p)^1}\vee\delta_{(p)^2}\Big);\\
& \Big(;_{p\in\mathbb{P}_{\pi_1}}(p)^1{\leftarrow}\bot;\delta_{(p)^1}{\leftarrow}\bot\Big);\Big(;_{p\in\mathbb{P}_{\pi_2}}(p)^2{\leftarrow}\bot;\delta_{(p)^2}{\leftarrow}\bot\Big)
\end{aligned}
$$

$$
\begin{aligned}
f(\pi_1\sqcup\pi_2) = {} & \Big(;_{p\in\mathbb{P}_{\pi_1}}(p)^1{\leftarrow}p\Big);\Big(;_{p\in\mathbb{P}_{\pi_2}}(p)^2{\leftarrow}p\Big);f\Big((\pi_1)^1\Big);f\Big((\pi_2)^2\Big);\\
& \Big(\Big(;_{p\in\mathbb{P}_{\pi_1}}p{\leftarrow}(p)^1;\delta_p{\leftarrow}\delta_p\vee\delta_{(p)^1}\Big)\cup\\
& \Big(;_{p\in\mathbb{P}_{\pi_2}}p{\leftarrow}(p)^2;\delta_p{\leftarrow}\delta_p\vee\delta_{(p)^2}\Big)\cup\\
& \Big(\bigwedge_{p\in\mathbb{P}_{\pi_1}\cap\mathbb{P}_{\pi_2}}((\delta_{(p)^1}\wedge\delta_{(p)^2})\rightarrow((p)^1\leftrightarrow(p)^2))?;\\
& (;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}}p{\leftarrow}(p\wedge\neg\delta_{(p)^1}\wedge\neg\delta_{(p)^1})\vee((p)^1\wedge\delta_{(p)^1})\vee((p)^2\wedge\delta_{(p)^2})\ ;\ \delta_p{\leftarrow}\delta_p\vee\delta_{(p)^1}\vee\delta_{(p)^2})\Big)\Big);\\
& \Big(;_{p\in\mathbb{P}_{\pi_1}}(p)^1{\leftarrow}\bot;\delta_{(p)^1}{\leftarrow}\bot\Big);\Big(;_{p\in\mathbb{P}_{\pi_2}}(p)^2{\leftarrow}\bot;\delta_{(p)^2}{\leftarrow}\bot\Big)
\end{aligned}
$$

Figure 2: Translation from DL-PPA to DL-PA

Proof. Let us take an arbitrary state V.

When $\tau_{\mathsf{a}}(V)$ is not defined then there are two cases. First, if $V\notin\|pre(\mathsf{a})\|$ then the program fails because $\|pre(\mathsf{a})?\|=\emptyset$. Second, if there are $ce_1,ce_2\in eff(\mathsf{a})$ and $p\in\mathbb{P}$ such that $V\in\|cnd(ce_1)\wedge cnd(ce_2)\|$ and $ceff^{+}(ce_1)\cap ceff^{-}(ce_2)$ contains p then the program fails because $\|p{\leftarrow}\bot\sqcap p{\leftarrow}\top\|=\emptyset$.

When $\tau_{\mathsf{a}}(V)$ is defined then $V\in\|pre(\mathsf{a})\|$, so $(V,V,\emptyset)\in\|pre(\mathsf{a})?\|$. Moreover, for each $ce\in eff(\mathsf{a})$ such that $V\in\|cnd(ce)\|$, the programs $\bigsqcap_{p\in ceff^{+}(ce)}p{\leftarrow}\top$ and $\bigsqcap_{q\in ceff^{-}(ce)}q{\leftarrow}\bot$ are executed in parallel and all the assignments are consistent (no $p{\leftarrow}\bot$ and $p{\leftarrow}\top$ is executed in parallel). Therefore the parallel composition of all these programs leads, by definition, to the state $\tau_{\mathsf{a}}(V)=U$ with $(V,U,W)\in\|\mathsf{exeAct}(\mathsf{a})\|$, where U is the set of variables assigned to $\top$ and W the set of all assigned variables in the program $\mathsf{exeAct}(\mathsf{a})$. □

4.2 Parallel Planning with Conditional Effects

In order to define solvability of a planning task by a parallel plan, we must determine the conditions of parallel executability of a set of conditional actions $A=\{\mathsf{a}_1,\ldots,\mathsf{a}_m\}$ in a state V. We consider that two actions are independent and can be executed in parallel if and only if these actions have no contradictory effects and no cross interactions.

Two actions a and a' that are executable in V have *contradictory effects* in state V if and only if there are conditional effects $ce\in eff(\mathsf{a})$ and $ce'\in eff(\mathsf{a}')$ such that:

- $V\in\|cnd(ce)\|$ and $V\in\|cnd(ce')\|$, and
- $ceff^{+}(ce)\cap ceff^{-}(ce')\neq\emptyset$.

Action a has *cross interaction* with another action $\mathsf{a}'\neq\mathsf{a}$ in state V if and only if there is a conditional effect $ce'\in eff(\mathsf{a}')$ such that:

- $V\in\|cnd(ce')\|$, and
- $V\notin\|\langle(\bigsqcap_{p\in ceff^{+}(ce')}p{\leftarrow}\top)\sqcap(\bigsqcap_{p\in ceff^{-}(ce')}p{\leftarrow}\bot)\rangle pre(\mathsf{a})\|$.

We say that a and a' have *cross interactions in state V* if and only if a has cross interaction with a' in the state V or a' has cross interaction with a in the state V.

In words, in any parallel plan, no effect of an action can be destroyed by an effect of another action executed in parallel, and no precondition of an action can be destroyed by an effect of another action executed in parallel. For example, in parallel classical planning, this makes that it is not possible to pick and drop a same object in parallel (contradictory effect) and that it is not possible for two agents to pick up the same object in parallel (cross interaction).

A set of conditional actions $\mathsf{A}=\{\mathsf{a}_1,\ldots,\mathsf{a}_m\}$ determines a partial function τ_{A} from $2^{\mathbb{P}}$ to $2^{\mathbb{P}}$: $\tau_{\mathsf{A}}(V)$ is defined ($\mathsf{a}_1,\ldots,\mathsf{a}_m$ are executable in parallel in state V) if and only if:

1. for all $\mathsf{a}_i\in\mathsf{A}$, τ_{a_i} is defined in V, and
2. for all $\mathsf{a}_i,\mathsf{a}_j\in\mathsf{A}$ such that $i\neq j$:
   - a_i and a_j have no contradictory effects, and
   - a_i and a_j have no cross interactions.

When τ_{A} is defined in V then:

$$\tau_{\mathsf{A}}(V) = \Big(V \setminus \bigcup_{\substack{\mathsf{a}\in\mathsf{A}, ce\in \mathit{eff}(\mathsf{a}),\\ V\in\|\mathit{cnd}(ce)\|}} \mathit{ceff}^{-}(ce)\Big) \cup \bigcup_{\substack{\mathsf{a}\in\mathsf{A}, ce\in \mathit{eff}(\mathsf{a}),\\ V\in\|\mathit{cnd}(ce)\|}} \mathit{ceff}^{+}(ce)$$

To every set of conditional actions we can associate a DL-PPA program that behaves exactly like the parallel execution of its elements. Given $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$, let $\mathsf{exeAct}(\mathsf{A})$ be the DL-PPA program

$$\textstyle\bigsqcap_{\mathsf{a}\in\mathsf{A}} \Big(\mathsf{exeAct}(\mathsf{a}) \sqcap \bigsqcap_{\mathsf{a}'\in\mathsf{A},\mathsf{a}\neq\mathsf{a}'} \langle \mathsf{exeAct}(\mathsf{a}')\rangle \mathit{pre}(\mathsf{a})?\Big)$$

Lemma 2. *For every finite set of actions* $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$*,*

1. τ_{A} *is defined in* V *iff there are* U, W *such that* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{A})\|$*;*
2. *If* τ_{A} *is defined in* V *then* $\tau_{\mathsf{A}}(V) = U$ *if and only if* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{A})\|$ *for some* W*.*

Proof. Let us take an arbitrary state V. By Lemma 1, for all $i \in \{1, \ldots, m\}$ we know that $\mathsf{exeAct}(\mathsf{a}_i)$ behaves like a_i, and then is executable if $\tau_{\mathsf{a}_i}(V)$ is defined. If two actions $\mathsf{a}, \mathsf{a}' \in \mathsf{A}$ have contradictory effects in state V, there are $ce \in \mathit{eff}(\mathsf{a})$ and $ce' \in \mathit{eff}(\mathsf{a}')$ and $p \in \mathbb{P}$ such that $V \in \|\mathit{cnd}(ce) \wedge \mathit{cnd}(ce')\|$ and $\mathit{ceff}^{+}(ce) \cap \mathit{ceff}^{-}(ce')$ contains p, then the program fails because of the execution of the parallel composition of $p{\leftarrow}\bot$ and $p{\leftarrow}\top$ in the program $\mathsf{exeAct}(\mathsf{a}) \sqcap \mathsf{exeAct}(\mathsf{a}')$. If action a has cross interaction with action a' in state V, there is a conditional effect $ce' \in \mathit{eff}(\mathsf{a}')$ such that $V \in \|\mathit{cnd}(ce')\|$ and $V \notin \|\langle(\bigsqcap_{p\in\mathit{ceff}^{+}(ce')} p{\leftarrow}\top) \sqcap (\bigsqcap_{p\in\mathit{ceff}^{-}(ce')} p{\leftarrow}\bot)\rangle \mathit{pre}(\mathsf{a})\|$, and then the execution of $\langle\mathsf{exeAct}(\mathsf{a}')\rangle\mathit{pre}(\mathsf{a})?$ fails. Finally, the parallel composition of all these programs check contradictory effects and cross interactions and leads to the state $\tau_{\mathsf{A}}(V)$ by definition because of the parallel execution of all $\mathsf{exeAct}(\mathsf{a}_i)$. □

We say that a state V is *reachable by a parallel plan* from a state V_0 via a set of conditional actions Act if there is a *parallel plan*, that is, a sequence $\langle \mathsf{A}_1, \ldots, \mathsf{A}_m\rangle$ of sets of actions $\mathsf{A}_i \subseteq Act$ and a sequence of states $\langle V_0, \ldots, V_m\rangle$ with $m \geq 0$ such that $V = V_m$ and $\tau_{\mathsf{A}_k}(V_{k-1}) = V_k$ for every k such that $1 \leq k \leq m$. A planning task $(Act, V_0, Goal)$ is *solvable by a parallel plan* if there is at least one state $V \in \|Goal\|$ such that V is reachable by a parallel plan from V_0 via Act; otherwise it is unsolvable by a parallel plan.

4.3 Solvability of Bounded Horizon Planning Tasks

Now that we have defined a parallel encoding of actions and the solvability of a planning task, we can capture the solvability of a planning task in DL-PPA with bounded horizon k, which is known to be PSPACE-complete [Bylander, 1994].

Theorem 3. *A planning task* $(Act, V_0, Goal)$ *is solvable by a sequential plan with no more than* k *actions if and only if:*

$$V_0 \in \Big\|\Big\langle \Big(\textstyle\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a})\Big)^{\leq k} \Big\rangle Goal\Big\|$$

Proof. Our formula reads "there exists an execution of $(\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a}))^{\leq k}$ after which *Goal* is true". We know by Lemma 1 that $\mathsf{exeAct}(\mathsf{a})$ behaves correctly and produces the same effects as action a. The program $(\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a}))^{\leq k}$ non-deterministically chooses an action a from Act and executes the corresponding program $\mathsf{exeAct}(\mathsf{a})$, then repeats this a number of times less or equal than k. This produces a sequence of at most k actions, i.e., a sequential plan bounded by k. Therefore the formula is satisfied in the initial state if and only if there exists a sequential plan of length bounded by k after which the goal is satisfied, i.e., if and only if the planning task is solvable with a sequence of at most k actions. □

Theorem 4. *A planning task* $(Act, V_0, Goal)$ *is solvable by a parallel plan with no more than* k *steps if and only if:*

$$V_0 \in \Big\|\Big\langle \Big(\big(\textstyle\bigsqcap_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\bot\big); \big(\bigsqcup_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\top\big); \pi_{\mathit{exe}}\Big)^{\leq k} \Big\rangle Goal\Big\|$$

where $\mathit{exe}_{\mathsf{a}} \notin \mathbb{P}_{\mathsf{exeAct}(\mathsf{a})}$ *for all* $\mathsf{a} \in Act$*, and*

$$\pi_{\mathit{exe}} = \bigsqcap_{\mathsf{a}\in Act} \left(\begin{array}{l}(\neg \mathit{exe}_{\mathsf{a}}? \cup\\ (\mathit{exe}_{\mathsf{a}}? \sqcap \mathsf{exeAct}(\mathsf{a})\\ \quad \sqcap \bigsqcap_{\substack{\mathsf{a}'\in Act\\ \mathsf{a}\neq\mathsf{a}'}} \big(\begin{array}{l}\neg \mathit{exe}_{\mathsf{a}'}?\\ \cup(\mathit{exe}_{\mathsf{a}'}? \sqcap \langle \mathsf{exeAct}(\mathsf{a}')\rangle \mathit{pre}(\mathsf{a})?)\end{array}\big)\end{array}\right)$$

Proof. The program $\bigsqcap_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\bot$ initialises a special fresh variable $\mathit{exe}_{\mathsf{a}} \notin \mathbb{P}_{\mathsf{exeAct}(\mathsf{a})}$ to $\bot$, for each action $\mathsf{a} \in Act$. Then the inclusive nondeterministic composition $\bigsqcup_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\top$ chooses some non empty subset of actions $A \subseteq Act$ and executes the program $\bigsqcap_{\mathsf{a}\in A} \mathit{exe}_{\mathsf{a}}{\leftarrow}\top$. At this point, $\mathit{exe}_{\mathsf{a}} = \top$ iff $\mathsf{a} \in \mathsf{A}$, and the program π_{exe} is executed. It is easily seen that, for a given choosen set of actions A, π_{exe} behaves like the program $\mathsf{exeAct}(\mathsf{A})$. We know by Lemma 2 that this latter program behaves correctly and produces the same effect as the parallel execution of all actions in A. The sequence $\bigsqcap_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\bot; \bigsqcup_{\mathsf{a}\in Act} \mathit{exe}_{\mathsf{a}}{\leftarrow}\top; \pi_{\mathit{exe}}$ is then repeated a number of times less or equal than k. This produces a sequence of at most k parallel executions of action sets, i.e., a parallel plan bounded by k. Therefore the formula is satisfied in the initial state if and only if there exists a parallel plan of length bounded by k after which the goal is satisfied, i.e., if and only if the planning task is solvable with a sequence of at most k parallel steps. □

Both in Theorem 3 and in Theorem 4, the size of the model checking problems is polynomial in the size of the planning task plus $\log k$. This relies on our compact representation of bounded programs $\pi^{\leq k}$, cf. Proposition 5.

5 Conclusion

We have shown how to capture parallel classical planning in an extension of DL-PA by parallel and inclusive nondeterministic composition whose model checking and satisfiability checking problems are still in PSPACE. This allows in particular to decide, within the complexity boundaries, the existence of a plan given a finite horizon.

For the unbounded case we get similar characterisations if we replace $\leq k$ in Theorems 3 and 4 by unbounded iteration *.

A straightforward continuation of our work is parallel epistemic planning: We can take over the epistemic extension of DL-PA in terms of observational variables that was applied to sequential epistemic planning in [Cooper *et al.*, 2016].

References

[Andersen *et al.*, 2012] Mikkel Birkegaard Andersen, Thomas Bolander, and Martin Holm Jensen. Conditional epistemic planning. In *Logics in Artificial Intelligence - 13th European Conference, JELIA*, pages 94–106, 2012.

[Balbiani and Boudou, 2018] Philippe Balbiani and Joseph Boudou. Iteration-free PDL with storing, recovering and parallel composition: a complete axiomatization. *J. Log. Comput.*, 28(4):705–731, 2018.

[Balbiani and Vakarelov, 2003] 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.

[Balbiani *et al.*, 2013] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments: A well-behaved variant of PDL. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 143–152, 2013.

[Balbiani *et al.*, 2014] 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.

[Benevides and Schechter, 2014] Mario R. F. Benevides and L. Menasché Schechter. Propositional dynamic logics for communicating concurrent programs with ccs's parallel operator. *J. Log. Comput.*, 24(4):919–951, 2014.

[Benevides *et al.*, 2011] Mario R. F. Benevides, Renata P. de Freitas, and Jorge Petrucio Viana. Propositional dynamic logic with storing, recovering and parallel composition. *Electr. Notes Theor. Comput. Sci.*, 269:95–107, 2011.

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

[Bolander *et al.*, 2018] Thomas Bolander, Thorsten Engesser, Robert Mattmüller, and Bernhard Nebel. Better eager than lazy? how agent types impact the successfulness of implicit coordination. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018*, pages 445–453, 2018.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[Cong *et al.*, 2018] Sébastien Lê Cong, Sophie Pinchinat, and François Schwarzentruber. Small undecidable problems in epistemic planning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 4780–4786, 2018.

[Cooper *et al.*, 2016] Martin C. Cooper, Andreas Herzig, Faustine Maffre, Frédéric Maris, and Pierre Régnier. A simple account of multi-agent epistemic planning. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pages 193–201, 2016.

[Dimopoulos *et al.*, 1997] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, pages 169–181, 1997.

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

[Herzig *et al.*, 2011] Andreas Herzig, Emiliano Lorini, Frédéric Moisan, and Nicolas Troquard. A dynamic logic of normative systems. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 228–233, 2011.

[Herzig *et al.*, 2014] Andreas Herzig, Maria Viviane de Menezes, Leliane Nunes de Barros, and Renata Wassermann. On the revision of planning tasks. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, pages 435–440, 2014.

[Herzig, 2014] Andreas Herzig. Belief change operations: A short history of nearly everything, told in dynamic logic of propositional assignments. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

[Knoblock, 1994] Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 98–103, 1994.

[Li *et al.*, 2017] Yanjun Li, Quan Yu, and Yanjing Wang. More for free: a dynamic epistemic framework for conformant planning over transition systems. *J. Log. Comput.*, 27(8):2383–2410, 2017.

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

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

[Spalazzi and Traverso, 2000] Luca Spalazzi and Paolo Traverso. A dynamic logic for acting, sensing, and planning. *J. Log. Comput.*, 10(6):787–821, 2000.

[Veloso *et al.*, 2014] Paulo A. S. Veloso, Sheila R. M. Veloso, and Mario R. F. Benevides. PDL for structured data: a graph-calculus approach. *Logic Journal of the IGPL*, 22(5):737–757, 2014.