The dynamic logic of policies and contingent planning

Thomas Bolander, Thorsten Engesser, Andreas Herzig, Robert Mattmüller, Bernhard Nebel

The Dynamic Logic of Policies and Contingent Planning

Thomas Bolander[1], Thorsten Engesser[3], Andreas Herzig[2(✉)], Robert Mattmüller[3], and Bernhard Nebel[3]

[1] DTU Compute, Technical University of Denmark, Lyngby, Denmark
tobo@dtu.dk
[2] IRIT, CNRS, University of Toulouse, Toulouse, France
herzig@irit.fr
[3] Faculty of Engineering, University of Freiburg, Freiburg im Breisgau, Germany
{engesser,mattmuel,nebel}@cs.uni-freiburg.de

Abstract. In classical deterministic planning, solutions to planning tasks are simply sequences of actions, but that is not sufficient for contingent plans in non-deterministic environments. Contingent plans are often expressed through policies that map states to actions. An alternative is to specify contingent plans as programs, e.g. in the syntax of Propositional Dynamic Logic (PDL). PDL is a logic for reasoning about programs with sequential composition, test and non-deterministic choice. However, as we show in the paper, none of the existing PDL modalities directly captures the notion of a solution to a planning task under non-determinism. We add a new modality to star-free PDL correctly capturing this notion. We prove the appropriateness of the new modality by showing how to translate back and forth between policies and PDL programs under the new modality. More precisely, we show how a policy solution to a planning task gives rise to a program solution expressed via the new modality, and vice versa. We also provide an axiomatisation of our PDL extension through reduction axioms into standard star-free PDL.

1 Introduction

Several authors have investigated how Propositional Dynamic Logic PDL can account for conformant planning [2,5,11,12]. We here push this program further and investigate how contingent planning can be captured in PDL. We argue that the standard PDL operators $[\pi]$ and $\langle\pi\rangle$ of necessity and possibility are not well-suited to account for conditional plans and introduce a third modal operator $(\![\pi]\!)\gamma$, read "π is strong for γ". Such an operator was already proposed for conformant planning in some of the above papers. Just as these proposals, $(\![a]\!)\varphi$ will be equivalent to $\langle a\rangle\top \wedge [a]\varphi$ for atomic actions a. More generally, for sequences of atomic actions $a_1;\ldots;a_n$ we have

$$(\![a_1;\ldots;a_n]\!)\varphi \leftrightarrow (\langle a_1\rangle\top \wedge [a_1](\cdots(\langle a_n\rangle\top \wedge [a_n]\varphi)\cdots)).$$

https://doi.org/10.1007/978-3-030-19570-0_43

We here go beyond sequential compositions and integrate nondeterministic composition and test. We show that this accounts for contingent planning, in the sense that there is a policy solving a contingent planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ with initial states S, goal formula γ and set of actions Act if and only if there is a program π such that $\mathcal{M}_{\mathsf{Act}}, S \Vdash (\!|\pi|\!)\gamma$, where $\mathcal{M}_{\mathsf{Act}}$ is the PDL Kripke model that captures the semantics of the actions Act.

The paper is organised as follows. In the next section we briefly recall PDL and define planning tasks and their sequential solutions. In Sect. 3 we define policies and contingent planning. In Sect. 4 we extend PDL by the new operator $(\!|\cdot|\!)$. In Sect. 5 we associate to every program a policy and, the other way round, we associate to every policy a program in Sect. 6.

2 Background: PDL and Sequential Plans

Propositional Dynamic Logic (PDL) is a modal logic that can immediately capture at least some forms of planning. Let us detail this for the case of sequential plans under full observability. We start by a brief introduction of star-free PDL; the reader is referred to [8,9] for more details.

Let Prp denote a finite set of propositional variables and Act a finite set of actions. A *Kripke model* $\mathcal{M}_{\mathsf{Act}} = \langle W, \{R_a\}_{a \in \mathsf{Act}}, V \rangle$ then consists of a set W of states (alias possible worlds), each action $a \in \mathsf{Act}$ is modelled by a binary relation R_a on W, and $V : W \longrightarrow 2^{\mathsf{Prp}}$ is a valuation associating to every state the propositional variables that are true there. Given a state $s \in W$, the possible outcomes of executing a at s is the set of states $R_a(s) = \{t \in W \mid \langle s, t \rangle \in R_a\}$. When $R_a(s) \neq \emptyset$ we say that a *is applicable at* s.

A set of states $S \subseteq W$ is called *valuation determined* if for all distinct $s, t \in S$ we have $V(s) \neq V(t)$. So S is valuation determined if all states in S are distinguishable via their valuation. In automated planning, the set of states of a planning domain is often just taken to be a subset of 2^{Prp}, and hence the set of all states is trivially valuation determined. However, in PDL, models are rarely restricted to only allow one state per valuation, so we will not make that restriction here either. However, to ensure a match between PDL programs and policies, we need at least to make the following weaker assumption.

We will assume all Kripke models to be *locally valuation determined*: For all actions $a \in \mathsf{Act}$ and all states $s \in W$, $R_a(s)$ is valuation determined. This requirement ensures that distinct outcomes of nondeterministic actions are necessarily distinguishable via their valuations. This requirement is necessary to guarantee that every policy can be translated into a corresponding program. Policies are going to be defined as relations between states and actions. A policy could for instance contain $\langle s, a \rangle, \langle t_1, b \rangle, \langle t_2, c \rangle$, assigning action a to state s, action b to state t_1 and action c to state t_2. Suppose $R_a(s) = \{t_1, t_2\}$. Then the policy specifies to execute a in s, and depending on the outcome, do either b or c. If the two possible outcomes t_1 and t_2 of a are not distinguishable by their valuation, there might not exist a formula distinguishing them, and hence there can be no PDL program representing the policy.

Any model $\mathcal{M}_{\mathsf{Act}}$ can be unravelled to a bisimilar, and hence modally equivalent, *tree model* [6,9]. It follows that we can assume all our Kripke models to be *acyclic*. We recall that a PDL Kripke model is cyclic if there is a natural number $n \geq 1$ and sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_0 = s_n$ and for every $k \geq 1$, $\langle s_{k-1}, s_k \rangle \in R_{a_k}$ for some a_k. Unravelling a locally valuation determined model will of course give a model that is also locally valuation determined.

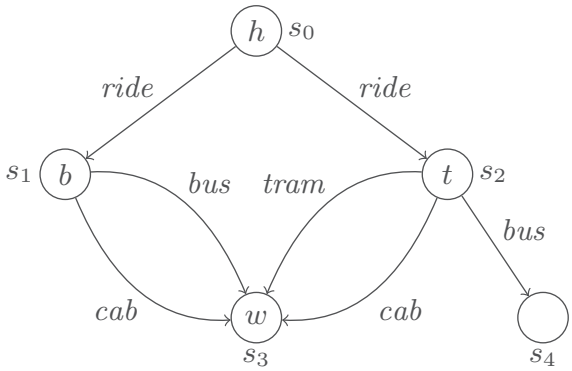

Fig. 1. A Kripke model.

Example 1. Consider the Kripke model given in Fig. 1. The story goes as follows: Initially, we are at home (h) and our goal is to get to work (w). We can get a lift by a friend who, depending on the traffic situation, will either drop us off at the train station (t) or the nearby bus station (b). We can then continue via multiple means of transportation, including the tram, the bus or a cab. At the bus station, taking the tram is not possible. Also, while we can take a bus from the train station, it will not get us to work.

Kripke models can interpret formulas and programs of PDL. We recall that the syntax of these is defined by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\pi\rangle\varphi \mid [\pi]\varphi$$
$$\pi ::= a \mid \pi;\pi \mid \pi \cup \pi \mid \varphi?$$

where a ranges over Act and p over the set of atomic propositions Prp. We use the standard abbreviations; in particular, the programs `skip` and `fail` respectively abbreviate $\top?$ and $\bot?$. Furthermore, we use the notation $\bigcup_{x \in X} \pi_x$ for finite nondeterministic compositions, understanding that the latter equals `fail` when X is empty. A *sequential program* is a PDL program of the form $a_1; \ldots; a_n$, for $n \geq 0$. By convention, when $n = 0$ we identify such a program with `skip`.

Example 2. Intuitively, in Example 1, the programs $\pi_1 = \mathit{ride}; (\mathit{tram} \cup \mathit{cab})$ and $\pi_2 = \mathit{ride}; ((b?; \mathit{bus}) \cup (t?; \mathit{tram}))$ successfully get us to work. Note that in π_1, after the application of *ride*, it becomes clear from the applicability of the actions whether the action *cab* has to be taken or whether we can choose either of *tram* or *cab*. In contrast, the program π_2 relies on tests to ensure that we make a good choice and, in particular, don't take the bus from the train station.

The interpretation of formulas and programs is defined by mutual recursion as follows:

$$\begin{aligned} R_{\pi_1;\pi_2} &= R_{\pi_1} \circ R_{\pi_2} \\ R_{\pi_1 \cup \pi_2} &= R_{\pi_1} \cup R_{\pi_2} \\ R_{\varphi?} &= \{\langle s, s\rangle \mid \mathcal{M}_{\mathsf{Act}}, s \Vdash \varphi\} \\ \mathcal{M}_{\mathsf{Act}}, s \Vdash p &\text{ iff } p \in V(s) \\ \mathcal{M}_{\mathsf{Act}}, s \Vdash \langle\pi\rangle\varphi &\text{ iff } \mathcal{M}_{\mathsf{Act}}, t \Vdash \varphi \text{ for some } t \in R_\pi(s) \\ \mathcal{M}_{\mathsf{Act}}, s \Vdash [\pi]\varphi &\text{ iff } \mathcal{M}_{\mathsf{Act}}, t \Vdash \varphi \text{ for every } t \in R_\pi(s) \end{aligned}$$

The truth conditions for the boolean connectives are the standard ones. From the provided abbrevations and semantics we get $R_{\mathtt{skip}} = \mathsf{id}_W = \{\langle s, s\rangle \mid s \in W\}$, $R_{\mathtt{fail}} = \emptyset$, and $R_{a_1;\ldots;a_n} = R_{a_1} \circ \ldots \circ R_{a_n}$, with the standard convention that it equals id_W when $n = 0$ (which justifies our identification of the empty sequential program with `skip`).

Before proceeding we generalise some semantic definitions from states to sets of states $S \subseteq W$. First, *a is applicable at S* if a is applicable at every $s \in S$. Second, $R_a(S) = \bigcup_{s \in S} R_a(s)$. Third, $\mathcal{M}_{\mathsf{Act}}, S \Vdash \varphi$ iff $\mathcal{M}_{\mathsf{Act}}, s \Vdash \varphi$ for every $s \in S$. So $\mathcal{M}_{\mathsf{Act}}, \emptyset \Vdash \varphi$ for all φ, and we can rewrite the last truth condition of the semantics more compactly as: $\mathcal{M}_{\mathsf{Act}}, s \Vdash [\pi]\varphi$ iff $\mathcal{M}_{\mathsf{Act}}, R_\pi(s) \Vdash \varphi$.

A *planning task* is given by a triple $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ with a valuation determined set of initial states $S \subseteq W$, goal formula γ of PDL and set of actions Act whose semantics is given by the Kripke model $\mathcal{M}_{\mathsf{Act}}$. Traditionally, planning tasks have a single initial state, but generalising to arbitrary (valuation determined) sets makes the technicalities in the following cleaner. A sequential program $a_1; \ldots; a_n$, alias a sequential plan, is a *solution* of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ if and only if

- $n = 0$ implies $\mathcal{M}_{\mathsf{Act}}, S \Vdash \gamma$;
- $n > 0$ implies a_1 is applicable at S and $a_2; \ldots; a_n$ is a solution of $\langle R_{a_1}(S), \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.

Sequential solutions of planning tasks can immediately be characterised in PDL when all actions are deterministic, i.e., when every $R_a(s)$ is either empty or a singleton: then the sequential plan $a_1; \ldots; a_n$ is a solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ iff $\mathcal{M}_{\mathsf{Act}}, S \Vdash \langle a_1; \ldots; a_n\rangle\gamma$. Things are less straightforward when actions can be nondeterministic. We follow [1,2,5,12] and introduce a third modal operator $(\![a_1; \ldots; a_n]\!)$ (noted $[\![a_1; \ldots; a_n]\!]$ in [1] and $((a_1; \ldots; a_n))$ in [2,5]) whose semantics is defined recursively by:

$$\begin{aligned} \mathcal{M}_{\mathsf{Act}}, s \Vdash (\![\mathtt{skip}]\!)\varphi \quad &\text{iff} \quad \mathcal{M}_{\mathsf{Act}}, s \Vdash \varphi \\ \mathcal{M}_{\mathsf{Act}}, s \Vdash (\![a; \pi]\!)\varphi \quad &\text{iff} \quad a \text{ is applicable at } s \text{ and } \mathcal{M}_{\mathsf{Act}}, R_a(s) \Vdash (\![\pi]\!)\varphi \end{aligned}$$

So $(\![a_1; \ldots; a_n]\!)\varphi$ is equivalent to $\langle a_1\rangle\top \wedge [a_1](\cdots(\langle a_n\rangle\top \wedge [a_n]\varphi)\cdots)$. It was established by several authors, e.g. [2,5,12], that $a_1; \ldots; a_n$ is a solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ iff $\mathcal{M}_{\mathsf{Act}}, S \Vdash (\![a_1; \ldots; a_n]\!)\gamma$.

In the rest of the paper we are going to extend the argument of this new modal operator to arbitrary star-free PDL programs. We are going to show that these programs capture policies, which means that our framework accounts for contingent planning.

3 Policies and Strong Solutions to Planning Tasks

Policies relate states to actions and provide the solution concept for contingent planning. We here introduce a slight generalisation of strong policies as defined in [4]. Note that we have assumed our models $\mathcal{M}_{\mathsf{Act}}$ to be acyclic, and hence our policies will also automatically be acyclic, consistent with the notion of a strong policy in [4].

Given a model $\mathcal{M}_{\mathsf{Act}}$, a *policy* (called state-action table in [4]) is a relation $\Lambda \subseteq W \times (\mathsf{Act} \cup \{\mathsf{stop}\})$. It is *defined* at a set of states $S \subseteq W$ if for every $s \in S$ there is an $x \in \mathsf{Act} \cup \{\mathsf{stop}\}$ such that $\langle s, x\rangle \in \Lambda$. It is *strongly executable* if for every $s \in W$ and $a \in \mathsf{Act}$, $\langle s, a\rangle \in \Lambda$ implies a is applicable at s and Λ is defined at $R_a(s)$. Note that our policies can be nondeterministic, since Λ is any relation between states and actions (instead of a partial function from states to actions). So we can for instance have $\langle s, a\rangle, \langle s, b\rangle \in \Lambda$, which means that the policy specifies two actions a and b in s that nature will choose nondeterministically between.

The special symbol stop is a 'license to stop': the elements of the set

$$\mathsf{Stop}(\Lambda) = \{t \in W \mid \langle t, \mathsf{stop}\rangle \in \Lambda\}$$

are the checkpoints of Λ where we are going to evaluate whether the goal is fulfilled. Such an entity is not a standard ingredient of polices, but makes sense in a nondeterministic setting: we can have policies such as $\Lambda = \{\langle s, \mathsf{stop}\rangle, \langle s, a\rangle\}$ which at s specifies a nondeterministic choice by nature between stoping to act (terminating policy execution) and performing action a. If the model $\mathcal{M}_{\mathsf{Act}}$ is such that some goal γ is true at s and at every outcome state $R_a(s)$, then we are entitled to say that Λ guarantees γ.

Given a finite policy Λ, the *depth* of Λ from a set of states $S \subseteq W$ is recursively defined by:

$$\mathsf{d}(\Lambda, S) = \begin{cases} 0 & \text{if } \Lambda \setminus (S \times \{\mathsf{stop}\}) = \emptyset \\ 1 + \mathsf{d}(\Lambda, \bigcup_{\langle s,a\rangle \in \Lambda \setminus (S \times \{\mathsf{stop}\})} R_a(s)) & \text{otherwise} \end{cases}$$

So when $\mathsf{d}(\Lambda, S) = 0$, the only elements of Λ are of the form $\langle s, \mathsf{stop}\rangle$. The function $\mathsf{d}(\Lambda, S)$ is well-defined because Λ is finite and $\mathcal{M}_{\mathsf{Act}}$ is acyclic.

Example 3. Consider the following policies for the model from Example 1:

$$\Lambda_1 = \{\langle s_0, \mathit{ride}\rangle, \langle s_1, \mathit{cab}\rangle, \langle s_2, \mathit{tram}\rangle, \langle s_2, \mathit{cab}\rangle, \langle s_3, \mathsf{stop}\rangle\}$$
$$\Lambda_2 = \{\langle s_0, \mathit{ride}\rangle, \langle s_1, \mathit{bus}\rangle, \langle s_2, \mathit{tram}\rangle, \langle s_3, \mathsf{stop}\rangle\}$$

Both policies are acyclic, finite, strongly executable, and have a depth of 2 from $\{s_0\}$. More importantly, we can see that following these policies will always lead us to the goal w. Also, Λ_1 and Λ_2 intuitively correspond to the programs π_1 and π_2 from Example 2. Note that Λ_1 is a nondeterministic policy, which allows us to take either the tram or the cab from s_2.

In the following, we will define what it means for a policy to solve a planning task. Later, we will also define the correspondence between policies and programs.

Definition 1. *A policy Λ is a* strong solution *of a planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ iff all of the following hold:*

1. *Λ is finite and strongly executable;*
2. *Λ is defined at S;*
3. *$\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda) \Vdash \gamma$.*

Lemma 1. *The policy $S \times \{\mathsf{stop}\}$ is a strong solution of a planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ iff $\mathcal{M}_{\mathsf{Act}}, S \Vdash \gamma$.*

Proof. The policy $S \times \{\mathsf{stop}\}$ is strongly executable, and is defined at S. It is also finite, since the set S of initial states of any planning task is valuation determined, and hence finite (there is only a finite set of propositional variables and hence only a finite set of possible valuations).

Lemma 2. *If Λ is a strong solution of a planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ then Λ is a strong solution of $\langle R_a(s), \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ for every $s \in S$ and $\langle s, a \rangle \in \Lambda$.*

Proof. Let Λ be a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ and $\langle s, a \rangle \in \Lambda$ for some $s \in S$. In order to establish that Λ is a strong solution of $\langle R_a(s), \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ it is enough to prove that Λ is defined at $R_a(s)$. This holds because Λ is strongly executable.

Lemma 3. *Suppose Λ_1 and Λ_2 are both strong solutions of the planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$. Then $\Lambda_1 \cup \Lambda_2$ is also a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$.*

Proof. Condition 1 of Definition 1: We need to prove finiteness and strong executability. $\Lambda_1 \cup \Lambda_2$ is clearly finite, as both Λ_1 and Λ_2 are. For strong executability, note that if $\langle s, a \rangle \in \Lambda_1 \cup \Lambda_2$ then $\langle s, a \rangle \in \Lambda_i$ for $i = 1$ or $i = 2$. Strong executability of Λ_i implies that $R_a(s)$ is non-empty and Λ_i is defined at $R_a(s)$. Therefore $\Lambda_1 \cup \Lambda_2$ is defined at $R_a(s)$, too. Condition 2: If Λ_1 is defined at S, then any extension of Λ_1 is also defined at S, including $\Lambda_1 \cup \Lambda_2$. Condition 3: If $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda_1) \Vdash \gamma$ and $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda_2) \Vdash \gamma$ then $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda_1) \cup \mathsf{Stop}(\Lambda_2) \Vdash \gamma$, and $\mathsf{Stop}(\Lambda_1) \cup \mathsf{Stop}(\Lambda_2) = \mathsf{Stop}(\Lambda_1 \cup \Lambda_2)$.

Lemma 4. *Suppose Λ_1 is a strong solution of $\langle S_1, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ and Λ_2 is a strong solution of $\langle S_2, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$. Then $\Lambda_1 \cup \Lambda_2$ is a strong solution of $\langle S_1 \cup S_2, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$.*

Proof. Condition 1: As in the proof of the previous lemma. Condition 2: $\Lambda_1 \cup \Lambda_2$ is clearly defined at $S_1 \cup S_2$. Condition 3: $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda_1 \cup \Lambda_2) \Vdash \gamma$ just as in the proof of the previous lemma.

The policy $\Lambda^{-\mathsf{stop}}$ is obtained from Λ by deleting all licenses to stop: $\Lambda^{-\mathsf{stop}} = \Lambda \setminus (W \times \{\mathsf{stop}\})$. This definition is useful to combine policies sequentially.

Lemma 5. *Suppose Λ_1 is finite, strongly executable and defined at S, and suppose Λ_2 is a strong solution of $\langle \mathsf{Stop}(\Lambda_1), \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$. Then $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$.*

Proof. Condition 1: $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is clearly finite. Let us show that it is also strongly executable. Suppose $\langle s, a\rangle \in \Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$. If $\langle s, a\rangle \in \Lambda_2$ then, as Λ_2 is strongly executable, a is applicable at s and Λ_2 is defined at $R_a(s)$, and so is $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$. Otherwise, if $\langle s, a\rangle \in \Lambda_1^{-\mathsf{stop}}$ then, as Λ_1 is strongly executable, a is applicable at s and Λ_1 is defined at $R_a(s)$. The latter means that for every $t \in R_a(s)$ there is an $x_t \in \mathsf{Act} \cup \{\mathsf{stop}\}$ such that $\langle t, x_t\rangle \in \Lambda_1$. We distinguish two cases: (1) when $x_t \in \mathsf{Act}$ then $\langle t, x_t\rangle \in \Lambda_1^{-\mathsf{stop}}$, and so $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is defined at t; (2) when $x_t = \mathsf{stop}$ then $t \in \mathsf{Stop}(\Lambda_1)$, and as Λ_2 is defined at $\mathsf{Stop}(\Lambda_1)$ we have that $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is defined at t. It follows that $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is defined at $R_a(s)$.

Condition 2: let us show that $\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$ is defined at S. Let $s \in S$ be chosen arbitrarily. As Λ_1 is defined at S, either there is an $a \in \mathsf{Act}$ such that $\langle s, a\rangle \in \Lambda_1$, and hence $\langle s, a\rangle \in \Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$, as required. Otherwise, $\langle s, \mathsf{stop}\rangle \in \Lambda_1$. This implies $s \in \mathsf{Stop}(\Lambda_1)$, and since Λ_2 applies to $\mathsf{Stop}(\Lambda_1)$, there must exist an $x \in \mathsf{Act} \cup \{\mathsf{stop}\}$ such that $\langle s, x\rangle \in \Lambda_2$, implying $\langle s, x\rangle \in \Lambda_1^{-\mathsf{stop}} \cup \Lambda_2$, as required.

Condition 3: As $\mathsf{Stop}(\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2) = \mathsf{Stop}(\Lambda_2)$ and Λ_2 is a strong solution of $\langle \mathsf{Stop}(\Lambda_1), \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$, we must have $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda_1^{-\mathsf{stop}} \cup \Lambda_2) \Vdash \gamma$.

The above lemmas basically show that union and sequential composition of two policies preserve strong solutions.

The sequential policy associated to a sequential program $a_1; \ldots; a_n$ and a set of states $S \subseteq W$ is

$$\mathtt{SPol}\big(a_1; \ldots; a_n, S\big) = \begin{cases} \emptyset & \text{if } n = 0 \\ (S \times \{a_1\}) \cup \mathtt{SPol}\big(a_2; \ldots; a_n, R_{a_1}(S)\big) & \text{otherwise} \end{cases}$$

It is then straightforward to prove that the sequential program $a_1; \ldots; a_n$ is a solution of the planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ if and only if the policy $\mathtt{SPol}\big(a_1; \ldots; a_n, S\big)$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. The other way round, it is clearly not the case that any policy can be mapped to a sequential program. For example, consider the policy Λ_2 from Example 3. This policy assigns the action *bus* to s_1 and *tram* to s_2. Since the action *ride* executed in s_0 can either result in s_1 or s_2, the policy specifies distinct actions depending on the outcome of *ride*. This policy can hence not be represented as a sequential program, as after the execution of *ride*, the choice of action to follow is conditional on the outcome of *ride*. This is also why the PDL program π_2 of Example 2 corresponding to Λ_2 includes tests on the outcome of *ride*: $\pi_2 = \mathit{ride}; ((b?; \mathit{bus}) \cup (t?; \mathit{tram}))$.

4 Extending PDL by the Modal Operator $(\!|\pi|\!)$

As announced, we extend the language of PDL by a third modal operator on programs $(\!|\pi|\!)$. The interpretation of $(\!|\pi|\!)$ is defined inductively:

$$\begin{array}{ll}
\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|a|\!)\varphi & \text{iff } a \text{ is applicable at } s \text{ and } \mathcal{M}_{\mathsf{Act}}, R_a(s) \Vdash \varphi \\
\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_1; \pi_2|\!)\varphi & \text{iff } \mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_1|\!)(\!|\pi_2|\!)\varphi \\
\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_1 \cup \pi_2|\!)\varphi & \text{iff } \mathcal{M}_{\mathsf{Act}}, s \Vdash ((\!|\pi_1|\!)\top \vee (\!|\pi_2|\!)\top) \wedge \\
 & \qquad ((\!|\pi_1|\!)\top \rightarrow (\!|\pi_1|\!)\varphi) \wedge \\
 & \qquad ((\!|\pi_2|\!)\top \rightarrow (\!|\pi_2|\!)\varphi) \\
\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\psi?|\!)\varphi & \text{iff } \mathcal{M}_{\mathsf{Act}}, s \Vdash \psi \text{ and } \mathcal{M}_{\mathsf{Act}}, s \Vdash \varphi
\end{array}$$

Satisfiability and validity are defined in the standard way. For example, the equivalence $(\![\pi]\!)\varphi \leftrightarrow (\![\pi;\varphi?]\!)\top$ is easily seen to be valid. Furthermore, it can be checked that the equivalences $(\![a]\!)\varphi \leftrightarrow \langle a\rangle\top \wedge [a]\varphi$ and $(\![a]\!)\top \leftrightarrow \langle a\rangle\top$ are valid. These equivalences however do not generalise to arbitrary programs.

For the model from Example 1, one can easily verify that $\mathcal{M}_{\mathsf{Act}}, s_0 \Vdash (\![\pi_1]\!)w$ and $\mathcal{M}_{\mathsf{Act}}, s_0 \Vdash (\![\pi_2]\!)w$.

Remark 1. Programs that are equivalent in PDL are no longer necessarily equivalent under the new modality. To witness, consider the programs *ride* and $(\mathit{ride};b?) \cup (\mathit{ride};\neg b?)$, which have the same interpretation in PDL. In our running example, *ride* is applicable at s_0 and nondeterministically produces either b or $\neg b$. Thus $\mathcal{M}_{\mathsf{Act}}, s_0 \Vdash (\![\mathit{ride}]\!)\top$, while $\mathcal{M}_{\mathsf{Act}}, s_0 \nVdash (\![(\mathit{ride};b?) \cup (\mathit{ride};\neg b?)]\!)\top$ because $\mathcal{M}_{\mathsf{Act}}, s_0 \nVdash (\![\mathit{ride};b?]\!)\top$ and $\mathcal{M}_{\mathsf{Act}}, s_0 \nVdash (\![\mathit{ride};\neg b?]\!)\top$.

Remark 2. Our semantics has two kinds of nondeterminism. The nondeterminism of atomic programs is *demonic*: it is the environment who chooses for example the outcome of the nondeterministic *ride* action. The nondeterminism of the choice-operator $\cup$ has an *angelic* component: while all applicable actions have to be successful, it is not required that all actions are applicable. Let us illustrate this by a couple of examples.

The choice operator necessarily has to be given a semantics that has such an angelic flavour if we want to account for nondeterministic policies. In our running example, the policy $\Lambda_3 = \{\langle s_2, \mathit{tram}\rangle, \langle s_2, \mathit{cab}\rangle, \langle s_3, \mathsf{stop}\rangle\}$ is a strong solution of the planning problem of getting from the train station to work. So $\mathcal{M}_{\mathsf{Act}}, s_2 \Vdash (\![\mathit{tram}]\!)w \wedge (\![\mathit{cab}]\!)w$. The only reasonable description of Λ as a PDL program is $\mathit{tram} \cup \mathit{cab}$, and indeed, $\mathcal{M}_{\mathsf{Act}}, s \Vdash (\![\mathit{tram} \cup \mathit{cab}]\!)w$. Now let us contrast this with states s_1 and s_2 where $\mathcal{M}_{\mathsf{Act}}, s_1 \Vdash b \wedge (\![\mathit{bus}]\!)w$ and $\mathcal{M}_{\mathsf{Act}}, s_2 \Vdash \neg b \wedge (\![\mathit{tram}]\!)w$. The policy $\Lambda_4 = \{\langle s_1, \mathit{bus}\rangle, \langle s_2, \mathit{tram}\rangle, \langle s_3, \mathsf{stop}\rangle\}$ is a strong solution of the planning problem $\langle \{s_1, s_2\}, w, \mathcal{M}_{\mathsf{Act}}\rangle$. The only reasonable description of Λ_4 seems to be the PDL counterpart of the conditional program "`if` b `then` *bus* `else` *tram*", namely $(b?;\mathit{bus}) \cup (\neg b?;\mathit{tram})$; and indeed, we have $\mathcal{M}_{\mathsf{Act}}, \{s_1, s_2\} \Vdash (\![(b?;\mathit{bus}) \cup (\neg b?;\mathit{tram})]\!)w$.

Note that we cannot have a purely angelic semantics where $\cup$ is interpreted as disjunction, that is, where $(\![a_1 \cup a_2]\!)\varphi \leftrightarrow ((\![a_1]\!)\varphi \vee (\![a_2]\!)\varphi)$. To see this, first note that we have $\mathcal{M}_{\mathsf{Act}}, s_2 \Vdash (\![\mathit{tram}]\!)w \wedge (\![\mathit{bus}]\!)\top \wedge (\![\mathit{bus}]\!)\neg w$. A purely angelic semantics would hence give us $\mathcal{M}_{\mathsf{Act}}, s_2 \Vdash (\![\mathit{tram} \cup \mathit{bus}]\!)w$, something we would not like to assert: the agent does not have a free choice between *tram* and *bus* to guarantee w (if taking the bus, the agent will not end up at the workplace). Contrast with the fact that $\mathcal{M}_{\mathsf{Act}}, s_1 \Vdash (\![\mathit{bus}]\!)w \wedge \neg(\![\mathit{tram}]\!)\top$. According to our semantics, we actually get $\mathcal{M}_{\mathsf{Act}}, s_1 \Vdash (\![\mathit{bus} \cup \mathit{tram}]\!)w$. This is OK, since *tram* is not even applicable at s_1, so the only possible execution of $\mathit{bus} \cup \mathit{tram}$ in s_1 is to execute *bus*. One can think of $\cup$ as giving a demonic nondeterministic choice where nature chooses which action will be executed, but only among the applicable ones. From the perspective of the acting agent, we can think of it as the agent choosing an arbitrary action, but again only among the applicable ones.

An axiomatisation of the validities of our language is obtained by adding the following to the axiomatisation of PDL:

$$(\![a]\!)\varphi \leftrightarrow \langle a\rangle\top \wedge [a]\varphi \qquad \text{(Atom)}$$
$$(\![\pi_1;\pi_2]\!)\varphi \leftrightarrow (\![\pi_1]\!)(\![\pi_2]\!)\varphi \qquad \text{(Seq)}$$
$$(\![\pi_1 \cup \pi_2]\!)\varphi \leftrightarrow ((\![\pi_1]\!)\top \vee (\![\pi_2]\!)\top) \wedge ((\![\pi_1]\!)\top \rightarrow (\![\pi_1]\!)\varphi) \wedge ((\![\pi_2]\!)\top \rightarrow (\![\pi_2]\!)\varphi) \qquad \text{(NDet)}$$
$$(\![\psi?]\!)\varphi \leftrightarrow \psi \wedge \varphi \qquad \text{(Test)}$$

The first four axioms are reduction axioms for program operators. They can be used from the left to the right to eliminate complex programs, applying the rule of replacement of equivalents (that can be derived without the rule of equivalents for $(\![\pi]\!)$) to subformulas that are not in the scope of any $(\![\pi]\!)$. This results in formulas where all programs are of the form a.

Soundness of our axioms can be proved straightforwardly, given that the axioms closely match the truth conditions for $(\![\pi]\!)$. Completeness of the axiomatisation can be proved by eliminating all $(\![\pi]\!)$ from formulas via the above reduction axioms. If all $(\![\pi]\!)$ are eliminated from a formula, what remains is a formula in the language of standard PDL whose satisfiability can be checked by solvers for PDL [7,10]. It follows that satisfiability and validity in our augmented PDL are both decidable.

We note that

$$(\![\pi_1 \cup \pi_2]\!)\varphi \leftrightarrow ((\![\pi_1]\!)\varphi \wedge (\![\pi_2]\!)\varphi) \vee ((\![\pi_1]\!)\varphi \wedge \neg(\![\pi_2]\!)\top) \vee (\neg(\![\pi_1]\!)\top \wedge (\![\pi_2]\!)\varphi)$$

is a propositionally equivalent formulation of the axiom for nondeterministic composition. In the rest of the section we provide some properties of our logic.

The modal operator $(\![\pi]\!)$ has almost all the properties of a normal modal operator: it satisfies the rule of monotony and the axiom of conjunction, and therefore also the K-axiom [3]. However, the rule of necessitation does not preserve validity.

Proposition 1. *The following rule of monotony for* $(\![\pi]\!)$ *is derivable:*

$$\text{if } \varphi \rightarrow \varphi' \text{ then } (\![\pi]\!)\varphi \rightarrow (\![\pi]\!)\varphi' \qquad (\mathrm{RM}_{(\![.]\!)})$$

Proof. By induction on the form of π. We only give the case of nondeterministic composition. Suppose $\varphi \rightarrow \varphi'$. $(\![\pi_1 \cup \pi_2]\!)\varphi$ is logically equivalent to

$$((\![\pi_1]\!)\top \vee (\![\pi_2]\!)\top) \wedge ((\![\pi_1]\!)\top \rightarrow (\![\pi_1]\!)\varphi) \wedge ((\![\pi_2]\!)\top \rightarrow (\![\pi_2]\!)\varphi).$$

Applying to the latter the induction hypothesis that $\varphi \rightarrow \varphi'$ implies $(\![\pi_1]\!)\varphi \rightarrow (\![\pi_1]\!)\varphi'$ and $(\![\pi_2]\!)\varphi \rightarrow (\![\pi_2]\!)\varphi'$ (twice), we obtain

$$(\![\pi_1 \cup \pi_2]\!)\varphi \rightarrow ((\![\pi_1]\!)\top \vee (\![\pi_2]\!)\top) \wedge ((\![\pi_1]\!)\top \rightarrow (\![\pi_1]\!)\varphi') \wedge ((\![\pi_2]\!)\top \rightarrow (\![\pi_2]\!)\varphi'),$$

which is equivalent to $(\![\pi_1 \cup \pi_2]\!)\varphi \rightarrow (\![\pi_1 \cup \pi_2]\!)\varphi'$.

Given that we have rules of equivalence for the other PDL connectives, it follows from the above that the rule of replacement of equivalents is derivable. Note that the theorem $(\![\pi]\!)\varphi \to (\![\pi]\!)\top$ directly follows from $\mathrm{RM}_{(\![.]\!)}$.

Proposition 2. *The following axiom of conjunction is a theorem:*

$$((\![\pi]\!)\varphi \wedge (\![\pi]\!)\varphi') \to (\![\pi]\!)(\varphi \wedge \varphi') \qquad (\mathrm{C}_{(\![.]\!)})$$

Proof. The proof is by induction on the form of the program. The base cases use axioms Atom and Test and PDL theorems. The induction step uses the above rule of monotony $\mathrm{RM}_{(\![.]\!)}$. For sequential composition we have:

1. $((\![\pi_1]\!)(\![\pi_2]\!)\varphi \wedge (\![\pi_1]\!)(\![\pi_2]\!)\varphi') \to (\![\pi_1]\!)((\![\pi_2]\!)\varphi \wedge (\![\pi_2]\!)\varphi')$ (by IH)
2. $((\![\pi_2]\!)\varphi \wedge (\![\pi_2]\!)\varphi') \to (\![\pi_2]\!)(\varphi \wedge \varphi')$ (by IH)
3. $(\![\pi_1]\!)((\![\pi_2]\!)\varphi \wedge (\![\pi_2]\!)\varphi') \to (\![\pi_1]\!)(\![\pi_2]\!)(\varphi \wedge \varphi')$ (from 2 by $\mathrm{RM}_{(\![.]\!)}$)
4. $((\![\pi_1]\!)(\![\pi_2]\!)\varphi \wedge (\![\pi_1]\!)(\![\pi_2]\!)\varphi') \to (\![\pi_1]\!)(\![\pi_2]\!)(\varphi \wedge \varphi')$ (from 1 and 3)
5. $((\![\pi_1;\pi_2]\!)\varphi \wedge (\![\pi_1;\pi_2]\!)\varphi') \to (\![\pi_1;\pi_2]\!)(\varphi \wedge \varphi')$ (from 4 by Axiom Seq)

The case of nondeterministic composition is similar but a bit lengthy due to the size of Axiom NDet.

The equivalence $((\![\pi]\!)\varphi \wedge (\![\pi]\!)\varphi') \leftrightarrow (\![\pi]\!)(\varphi \wedge \varphi')$ can be proved from Axiom $\mathrm{C}_{(\![.]\!)}$ and the derived inference rule $\mathrm{RM}_{(\![.]\!)}$.

Altogether, it looks like $(\![\cdot]\!)$ is a normal modal operator. However, the rule of necessitation 'from φ infer $(\![\pi]\!)\varphi$' fails to preserve validity. Indeed, $(\![\pi]\!)\top$ fails to be valid. To see this, consider a model where R_a is empty: then $\langle a\rangle\top$ is false at any state s, and therefore $(\![a]\!)\top$ is false everywhere, too. This is as it should be: if $\langle a\rangle\top$ was valid then any action a would be applicable. Worse, validity of $\langle\pi\rangle\top$ for any program π would mean that e.g. the 'fail' program $\bot?$ would be applicable.

The following theorems can be proved by induction on the form of programs, except item (7).

Proposition 3. *The following are theorems.*

$$(\![\pi]\!)\bot \leftrightarrow \bot \tag{1}$$
$$(\![\pi]\!)\varphi \to \langle\pi\rangle\varphi \tag{2}$$
$$[\pi]\varphi \wedge (\![\pi]\!)\top \to (\![\pi]\!)\varphi \tag{3}$$
$$(\![\pi \cup \bot?]\!)\varphi \leftrightarrow (\![\pi]\!)\varphi \tag{4}$$
$$(\![\pi_1 \cup \pi_2]\!)\top \leftrightarrow ((\![\pi_1]\!)\top \vee (\![\pi_2]\!)\top) \tag{5}$$
$$(\![(\pi_1;\pi_2)\cup(\pi_1;\pi_2')]\!)\varphi \to (\![\pi_1;(\pi_2 \cup \pi_2')]\!)\varphi \tag{6}$$
$$(\![\psi_1? \cup \psi_2?]\!)\varphi \leftrightarrow (\psi_1 \vee \psi_2) \wedge \varphi \tag{7}$$

None of the implications in Proposition 3 can be extended into equivalences. To see this for item (2), it suffices to consider a nondeterministic atomic action a with two possible outcomes p and $\neg p$: then $\langle a\rangle p$ holds but $(\![a]\!)p$ does not. For item (3), this follows from the falsifiability of $(\![\pi]\!)\varphi \to [\pi]\varphi$. To see this, consider the

program $\pi = p? \cup (a; \neg p?)$ and a model $\mathcal{M}_{\mathsf{Act}}$ with a state s where p is true and where a nondeterministically produces outcome p or $\neg p$. Then $\mathcal{M}_{\mathsf{Act}}, s \Vdash \langle\!\langle \pi \rangle\!\rangle p$, in particular because $\mathcal{M}_{\mathsf{Act}}, s \Vdash \neg \langle\!\langle a; \neg p? \rangle\!\rangle \top$. On the other hand, $\mathcal{M}_{\mathsf{Act}}, s \nVdash [\pi] p$ because $\mathcal{M}_{\mathsf{Act}}, s \nVdash [a; \neg p?] p$. For item (6), this can be seen from from the example that we have given in Remark 1.

5 From Programs to Policies

In this section we associate a policy to a given program. Recall that models $\mathcal{M}_{\mathsf{Act}}$ are assumed to be acyclic (without loss of generality). We recursively associate to every program π and set of states S a policy $\mathsf{Pol}(\pi, S)$ as follows:

- If $\mathcal{M}_{\mathsf{Act}}, S \nVdash \langle\!\langle \pi \rangle\!\rangle \top$ then $\mathsf{Pol}(\pi, S) = \emptyset$;
- If $\mathcal{M}_{\mathsf{Act}}, S \Vdash \langle\!\langle \pi \rangle\!\rangle \top$ then, depending on the form of π:

$$
\begin{aligned}
\mathsf{Pol}(\psi?, S) &= S \times \{\mathsf{stop}\} \\
\mathsf{Pol}(a, S) &= \big(S \times \{a\}\big) \cup \big(R_a(S) \times \{\mathsf{stop}\}\big) \\
\mathsf{Pol}(\pi_1; \pi_2, S) &= \big(\mathsf{Pol}(\pi_1, S)\big)^{-\mathsf{stop}} \cup \mathsf{Pol}(\pi_2, \mathsf{Stop}(\mathsf{Pol}(\pi_1, S))) \\
\mathsf{Pol}(\pi_1 \cup \pi_2, S) &= \bigcup_{s \in S} \big(\mathsf{Pol}(\pi_1, \{s\}) \cup \mathsf{Pol}(\pi_2, \{s\})\big)
\end{aligned}
$$

Example 4. Suppose $\mathcal{M}_{\mathsf{Act}}$ is such that $R_{a_1} = \{\langle s_1, t_1 \rangle\}$ and $R_{a_2} = \{\langle s_2, t_2 \rangle\}$. Then $\mathsf{Pol}(a_1 \cup a_2, \{s_1, s_2\}) = \{\langle s_1, a_1 \rangle, \langle s_2, a_2 \rangle, \langle t_1, \mathsf{stop} \rangle, \langle t_2, \mathsf{stop} \rangle\}$. This justifies the case of nondeterministic composition: $\mathsf{Pol}(a_1 \cup a_2, \{s_1, s_2\})$ would be empty had we defined $\mathsf{Pol}(\pi_1 \cup \pi_2, S)$ as $\mathsf{Pol}(\pi_2, S) \cup \mathsf{Pol}(\pi_2, S)$ (plus $\mathsf{Pol}(a, S)$ as empty if a is inapplicable at some $s \in S$).

Example 5. Suppose $\mathcal{M}_{\mathsf{Act}}$ is the model from our running example (Fig. 1). For the program $h?$ we have $\mathsf{Pol}(h?, \{s_0\}) = \{\langle s_0, \mathsf{stop} \rangle\}$. Consider the program $\mathit{ride}; b?$. Then $\mathcal{M}_{\mathsf{Act}}, s_0 \nVdash \langle\!\langle \mathit{ride}; b? \rangle\!\rangle \top$, and therefore $\mathsf{Pol}(\mathit{ride}; b?, \{s_0\}) = \emptyset$. Consider the program $\pi = h? \cup (\mathit{ride}; b?)$. Then $\mathcal{M}_{\mathsf{Act}}, s_0 \Vdash \langle\!\langle h? \rangle\!\rangle \top$ and $\mathcal{M}_{\mathsf{Act}}, s_0 \nVdash \langle\!\langle \mathit{ride}; b? \rangle\!\rangle \top$, and therefore $\mathsf{Pol}(\pi, \{s_0\}) = \{\langle s_0, \mathsf{stop} \rangle\}$.

While $R_\pi(S)$ contains $\mathsf{Stop}(\mathsf{Pol}(\pi, S))$ for every set of states S (the proof is by induction on the form of π), the converse fails to hold. This can be seen from the above example: s_1 is not in $\mathsf{Stop}(\mathsf{Pol}(\pi, S))$, although it is in $R_{\mathit{ride}}(s)$.

Example 6. For the model from our running example and the program consisting of the single action ride, we obtain the following policy:

$$
\begin{aligned}
\mathsf{Pol}(\mathit{ride}, \{s_0\}) &= (\{s_0\} \times \{\mathit{ride}\}) \cup (\{s_1, s_2\} \times \{\mathsf{stop}\}) \\
&= \{\langle s_0, \mathit{ride} \rangle, \langle s_1, \mathsf{stop} \rangle, \langle s_2, \mathsf{stop} \rangle\}
\end{aligned}
$$

For the program $\pi_1 = \mathit{ride}; (\mathit{tram} \cup \mathit{cab})$ we then obtain the following policy:

$$\begin{aligned}
&\mathsf{Pol}(\pi_1, \{s_0\}) \\
&= \mathsf{Pol}(\mathit{ride}, \{s_0\})^{-\mathsf{stop}} \cup \mathsf{Pol}(\mathit{tram} \cup \mathit{cab}, \mathsf{Stop}(\mathsf{Pol}(\mathit{ride}, \{s_0\}))) \\
&= \{\langle s_0, \mathit{ride}\rangle\} \cup \mathsf{Pol}(\mathit{tram} \cup \mathit{cab}, \{s_1, s_2\}) \\
&= \{\langle s_0, \mathit{ride}\rangle\} \cup \bigcup \{\mathsf{Pol}(\mathit{tram}, \{s_1\}) \cup \mathsf{Pol}(\mathit{cab}, \{s_1\}), \\
&\qquad\qquad \mathsf{Pol}(\mathit{tram}, \{s_2\}) \cup \mathsf{Pol}(\mathit{cab}, \{s_2\})\} \\
&= \{\langle s_0, \mathit{ride}\rangle\} \cup \emptyset \cup \{\langle s_1, \mathit{cab}\rangle, \langle s_3, \mathsf{stop}\rangle\} \\
&\qquad\qquad \cup \{\langle s_2, \mathit{tram}\rangle, \langle s_3, \mathsf{stop}\rangle\} \cup \{\langle s_2, \mathit{cab}\rangle, \langle s_3, \mathsf{stop}\rangle\} \\
&= \{\langle s_0, \mathit{ride}\rangle, \langle s_1, \mathit{cab}\rangle, \langle s_2, \mathit{tram}\rangle, \langle s_2, \mathit{cab}\rangle, \langle s_3, \mathsf{stop}\rangle\} = \Lambda_1
\end{aligned}$$

This verifies that indeed Λ_1 is the policy corresponding to π_1 as claimed in Example 3.

Lemma 6. *Suppose S is finite. Then $\mathsf{Pol}(\pi, S)$ finite, and for every $a \in \mathsf{Act}$, $\langle s, a\rangle \in \mathsf{Pol}(\pi, S)$ implies that a is applicable at s.*

Proof. We can prove by induction on the form of π that for every $a \in \mathsf{Act}$, if $\langle s, a\rangle \in \mathsf{Pol}(\pi, S)$ then a is applicable at s. Now note that since models are assumed to be locally valuation determined, for every action a and state s, $R_a(s)$ must be finite. Finiteness of $\mathsf{Pol}(\pi, S)$ is then due to finiteness of S and to finiteness of every $R_a(s)$; the proof is by induction on the form of π.

The hypotheses of Lemma 6 are not enough to guarantee strong executability of $\mathsf{Pol}(\pi, S)$. Consider our model from Fig. 1 which clearly satisfies the hypotheses of Lemma 6. Here, $\mathsf{Pol}(\mathit{ride}; \mathit{tram}, \{s_0\})$ is empty because $\mathcal{M}_{\mathsf{Act}}, s_0 \not\Vdash (\![\mathit{ride}; \mathit{tram}]\!)\top$. The next result says that $\mathsf{Pol}(\pi, S)$ is strongly executable under the condition that $(\![\pi]\!)\gamma$ is true at S (for any γ, so in particular when γ is $\top$). It moreover says that then $\mathsf{Pol}(\pi, S)$ is defined at S and $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\mathsf{Pol}(\pi, S)) \Vdash \gamma$.

Proposition 4. *Let $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ be a planning task and suppose $\mathcal{M}_{\mathsf{Act}}, S \Vdash (\![\pi]\!)\gamma$. Then $\mathsf{Pol}(\pi, S)$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.*

Proof. Since $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ is a planning task, S is valuation determined, and hence finite. Thus, by Lemma 6, $\mathsf{Pol}(\pi, S)$ is finite, and $\langle s, a\rangle \in \mathsf{Pol}(\pi, S)$ implies that a is applicable at s for every $a \in \mathsf{Act}$. To show that $\mathsf{Pol}(\pi, S)$ is strongly executable it remains to show that $\langle s, a\rangle \in \mathsf{Pol}(\pi, S)$ implies that $\mathsf{Pol}(\pi, S)$ is defined at $R_a(s)$. Furthermore, we have to show that $\mathsf{Pol}(\pi, S)$ is defined at S and that $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\mathsf{Pol}(\pi, S)) \Vdash \gamma$. We proceed by induction on the form of π.

$\mathcal{M}_{\mathsf{Act}}, S \Vdash (\![\psi?]\!)\gamma$ implies $\mathcal{M}_{\mathsf{Act}}, S \Vdash \psi$ and $\mathcal{M}_{\mathsf{Act}}, S \vDash \gamma$ by the truth condition for test. By Lemma 1, $\mathsf{Pol}(\psi?, S) = S \times \{\mathsf{stop}\}$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.

$\mathcal{M}_{\mathsf{Act}}, S \Vdash (\![a]\!)\gamma$ implies $\mathcal{M}_{\mathsf{Act}}, R_a(S) \Vdash \gamma$. The policy $\mathsf{Pol}(a, S) = \big(S \times \{a\}\big) \cup \big(R_a(S) \times \{\mathsf{stop}\}\big)$ is defined at S and at $R_a(S)$ (so due to the latter it is strongly executable). Hence $\mathsf{Pol}(a, S)$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.

$\mathcal{M}_{\mathsf{Act}}, S \Vdash (\!|\pi_1;\pi_2|\!)\gamma$ implies $\mathcal{M}_{\mathsf{Act}}, S \Vdash (\!|\pi_1|\!)(\!|\pi_2|\!)\gamma$. By induction hypothesis $\mathsf{Pol}(\pi_1, S)$ is a strong solution of $\langle S, (\!|\pi_2|\!)\gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. So $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\mathsf{Pol}(\pi_1, S)) \Vdash (\!|\pi_2|\!)\gamma$. We apply the induction hypothesis again: $\mathsf{Pol}(\pi_2, \mathsf{Stop}(\mathsf{Pol}(\pi_1, S)))$ is a strong solution of the planning task $\langle \mathsf{Stop}(\mathsf{Pol}(\pi_1, S)), \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. Then

$$\mathsf{Pol}(\pi_1;\pi_2, S) = \big(\mathsf{Pol}(\pi_1, S)\big)^{-\mathsf{stop}} \cup \mathsf{Pol}(\pi_2, \mathsf{Stop}(\mathsf{Pol}(\pi_1, S)))$$

is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ thanks to Lemma 5.

$\mathcal{M}_{\mathsf{Act}}, S \Vdash (\!|\pi_1 \cup \pi_2|\!)\gamma$ implies that for every $s \in S$, one of the following holds:

1. $\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_1|\!)\gamma$ and $\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_2|\!)\gamma$;
2. $\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_1|\!)\gamma$ and $\mathcal{M}_{\mathsf{Act}}, s \nVdash (\!|\pi_2|\!)\top$;
3. $\mathcal{M}_{\mathsf{Act}}, s \nVdash (\!|\pi_1|\!)\top$ and $\mathcal{M}_{\mathsf{Act}}, s \Vdash (\!|\pi_2|\!)\gamma$.

Remember that by definition $\mathsf{Pol}(\pi_i, S)$ is empty if $\mathcal{M}_{\mathsf{Act}}, S \nVdash (\!|\pi_i|\!)\top$. Therefore by the induction hypothesis one of the following holds:

1. $\mathsf{Pol}(\pi_1, \{s\})$ and $\mathsf{Pol}(\pi_2, \{s\})$ are both strong solutions of $\langle \{s\}, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$;
2. $\mathsf{Pol}(\pi_1, \{s\})$ is a strong solution of $\langle \{s\}, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ and $\mathsf{Pol}(\pi_2, \{s\}) = \emptyset$;
3. $\mathsf{Pol}(\pi_1, \{s\}) = \emptyset$ and $\mathsf{Pol}(\pi_2, \{s\})$ is a strong solution of $\langle \{s\}, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.

In each of these three cases we have that $\mathsf{Pol}(\pi_1, \{s\}) \cup \mathsf{Pol}(\pi_2, \{s\})$ is a strong solution of $\langle \{s\}, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$, where in the first case we apply Lemma 3. Finally, by Lemma 4 we conclude that $\mathsf{Pol}(\pi_1 \cup \pi_2, S) = \bigcup_{s\in S} \big(\mathsf{Pol}(\pi_1, \{s\}) \cup \mathsf{Pol}(\pi_2, \{s\})\big)$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. This completes the proof.

We immediately get the following, given that by definition $\mathsf{Pol}(\pi, S)$ is empty when $\mathcal{M}_{\mathsf{Act}}, S \nVdash (\!|\pi|\!)\gamma$.

Corollary 1. *Let $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ be a planning task. Then $\mathcal{M}_{\mathsf{Act}}, S \Vdash (\!|\pi|\!)\gamma$ iff $\mathsf{Pol}(\pi, S)$ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$.*

6 From Policies to Programs

In this section we associate a program to a given policy. Given a model $\mathcal{M}_{\mathsf{Act}} = \langle W, \{R_a\}_{a\in\mathsf{Act}}, V\rangle$, the *characteristic formula* of a state $s \in W$ is

$$\chi_s = \Big(\bigwedge_{p\in V(s)} p\Big) \wedge \Big(\bigwedge_{p\notin V(s)} \neg p\Big)$$

Such formulas will be tested in the program associated to a policy Λ in order to correctly capture the actions of Λ that apply at s. The crucial property is that χ_s is only true in states that have the same valuation as s, as is immediately seen from the definition of the χ_s.

The abbreviation $\mathsf{skipifstop}_\Lambda(s) = \bigcup_{\langle s,\mathsf{stop}\rangle\in\Lambda} \mathtt{skip}$ will be convenient: when $\langle s, \mathsf{stop}\rangle \in \Lambda$ then it is equivalent to `skip`; otherwise (by our convention of Sect. 2) it equals `fail`. Now we are ready to associate to every finite policy Λ and finite set of states S a program $\pi_{\Lambda,S}$ as follows:

$$\pi_{\Lambda,S} = \bigcup_{s\in S}\left(\chi_s?;\left(\mathsf{skipifstop}_\Lambda(s) \cup \bigcup_{a\,|\,\langle s,a\rangle\in\Lambda} \big(a;\pi_{\Lambda,R_a(s)}\big)\right)\right)$$

The function $\pi_{\Lambda,S}$ is well-defined because Λ is finite and $\mathcal{M}_{\mathsf{Act}}$ is acyclic.

Proposition 5. *If Λ is a strong solution of the planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ then $\mathcal{M}_{\mathsf{Act}}, S \Vdash [\![\pi_{\Lambda,S}]\!]\gamma$.*

Proof. By induction on the depth $\mathsf{d}(\Lambda, S)$ of the policy Λ from S. If $\mathsf{d}(\Lambda, S) = 0$ then there can be no $\langle a, s\rangle \in \Lambda$ for any $s \in S$, and for every s the subprogram $\bigcup_{a|\langle s,a\rangle\in\Lambda}\left(a; \pi_{\Lambda,R_a(s)}\right)$ of $\mathsf{Pol}(\Lambda, S)$ is the `fail` program. Therefore $\pi_{\Lambda,S}$ is $\bigcup_{s\in S}\left(\chi_s?; \left(\mathsf{skipifstop}_\Lambda(s) \cup \mathtt{fail}\right)\right)$, which is equivalent to $\bigcup_{s\in S}\left(\chi_s?; \left(\mathsf{skipifstop}_\Lambda(s)\right)\right)$, by Proposition 3, item (4). Suppose Λ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. So γ is true at $\mathsf{Stop}(\Lambda) = S$ and Λ applies to S. The latter means that $\mathsf{skipifstop}_\Lambda(s)$ equals `skip` for every $s \in S$. Therefore the program $\pi_{\Lambda,S}$ equals $\bigcup_{s\in S}(\chi_s?; \mathtt{skip})$, which is equivalent to the program $\bigcup_{s\in S}\chi_s?$ (more precisely, we have $R_{\pi;\mathtt{skip}} = R_\pi \circ R_{\top?} = R_\pi$). By item (7) of Proposition 3, the formula $[\![\bigcup_{s\in S}\chi_s?]\!]\gamma$ is equivalent to $(\bigvee_{s\in S}\chi_s) \wedge \gamma$; and as $\mathcal{M}_{\mathsf{Act}}, S \Vdash \bigvee_{s\in S}\chi_s$, we have that $\mathcal{M}_{\mathsf{Act}}, S \Vdash [\![\bigcup_{s\in S}\chi_s?]\!]\gamma$. So we can conclude that $\mathcal{M}_{\mathsf{Act}}, S \Vdash [\![\pi_{\Lambda,S}]\!]\gamma$.

If $\mathsf{d}(\Lambda, S) \geq 1$ then suppose Λ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$. Choose $s \in S$ arbitrarily. We then need to prove $\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![\pi_{\Lambda,S}]\!]\gamma$. By Lemma 2, Λ is a strong solution of $\langle R_a(s), \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ for every $\langle s, a\rangle \in \Lambda$. Then by the induction hypothesis we have for every $\langle s, a\rangle \in \Lambda$

$$\mathcal{M}_{\mathsf{Act}}, R_a(s) \Vdash [\![\pi_{\Lambda,R_a(s)}]\!]\gamma,$$

i.e., as Λ applies to s, that for every a such that $\langle s, a\rangle \in \Lambda$:

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![a]\!][\![\pi_{\Lambda,R_a(s)}]\!]\gamma.$$

Therefore,

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash \bigwedge_{a|\langle s,a\rangle\in\Lambda}[\![a; \pi_{\Lambda,R_a(s)}]\!]\gamma,$$

which by the validity of Axiom NDet implies that

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![\bigcup_{a|\langle s,a\rangle\in\Lambda}\left(a; \pi_{\Lambda,R_a(s)}\right)]\!]\gamma.$$

Furthermore, as Λ is a strong solution of $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}}\rangle$ we have $\mathcal{M}_{\mathsf{Act}}, \mathsf{Stop}(\Lambda) \Vdash \gamma$ and hence,

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![\mathsf{skipifstop}_\Lambda(s)]\!]\gamma \vee \neg[\![\mathsf{skipifstop}_\Lambda(s)]\!]\top.$$

Due to the validity of Axiom NDet we can combine the last two lines and obtain

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![\mathsf{skipifstop}_\Lambda(s) \cup \left(\bigcup_{a|\langle s,a\rangle\in\Lambda}\left(a; \pi_{\Lambda,R_a(s)}\right)\right)]\!]\gamma.$$

Since $\mathcal{M}_{\mathsf{Act}}, s \Vdash \chi_s$, we have:

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash [\![\chi_s?]\!][\![\mathsf{skipifstop}_\Lambda(s) \cup \left(\bigcup_{a|\langle s,a\rangle\in\Lambda}\left(a; \pi_{\Lambda,R_a(s)}\right)\right)]\!]\gamma.$$

Using the validity of Axiom Seq, we then get:

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash \langle\!\langle \chi_s?; \big(\mathsf{skipifstop}_{\Lambda}(s) \cup \big(\bigcup_{a \mid \langle s,a \rangle \in \Lambda} \big(a; \pi_{\Lambda, R_a(s)}\big)\big)\big)\big)\rangle\!\rangle \gamma.$$

Since the set S of initial states of any planning task is assumed to be valuation determined, we must have $\mathcal{M}_{\mathsf{Act}}, s \Vdash \neg\chi_t$ for every $t \in S \setminus \{s\}$. This implies $\mathcal{M}_{\mathsf{Act}}, s \Vdash \neg\langle\!\langle \chi_t \rangle\!\rangle \top$, and hence:

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash \neg\langle\!\langle \chi_t?; \big(\mathsf{skipifstop}_{\Lambda}(t) \cup \big(\bigcup_{a \mid \langle t,a \rangle \in \Lambda} \big(a; \pi_{\Lambda, R_a(t)}\big)\big)\big)\big)\rangle\!\rangle \top, \text{ for } t \in S \setminus \{s\}.$$

Applying Axiom NDet to the last two lines, we now get

$$\mathcal{M}_{\mathsf{Act}}, s \Vdash \langle\!\langle \bigcup_{s \in S} \Big(\chi_s?; \big(\mathsf{skipifstop}_{\Lambda}(s) \cup \bigcup_{a \mid \langle s,a \rangle \in \Lambda} \big(a; \pi_{\Lambda, R_a(s)}\big)\big)\Big)\rangle\!\rangle \gamma.$$

In other words, $\mathcal{M}_{\mathsf{Act}}, s \Vdash \langle\!\langle \pi_{\Lambda,S} \rangle\!\rangle \gamma$ as required.

Putting Propositions 4 and 5 together now finally gives us the following.

Corollary 2. *A planning task $\langle S, \gamma, \mathcal{M}_{\mathsf{Act}} \rangle$ has a strong solution iff there exists a star-free PDL program π such that $\mathcal{M}_{\mathsf{Act}}, S \Vdash \langle\!\langle \pi \rangle\!\rangle \gamma$.*

This indicates that PDL with our new modality $\langle\!\langle \cdot \rangle\!\rangle$ provides an appropriate linguistic and semantic framework to reason about policies.

References

1. Andersen, M.B., Bolander, T., Jensen, M.H.: Conditional epistemic planning. In: del Cerro, L.F., Herzig, A., Mengin, J. (eds.) JELIA 2012. LNCS (LNAI), vol. 7519, pp. 94–106. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33353-8_8
2. Bolander, T., Engesser, T., Mattmüller, R., Nebel, B.: Better eager than lazy? How agent types impact the successfulness of implicit coordination. In: Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018). AAAI Press (2018)
3. Chellas, B.F.: Modal Logic: An Introduction. Cambridge University Press, Cambridge (1980)
4. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artif. Intell. **147**(1–2), 35–84 (2003)
5. Engesser, T., Bolander, T., Mattmüller, R., Nebel, B.: Cooperative epistemic multi-agent planning for implicit coordination. In: Ghosh, S., Ramanujam, R. (eds.) Proceedings of the Ninth Workshop on Methods for Modalities, M4M. EPTCS, vol. 243, pp. 75–90 (2017)
6. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications, Studies in Logic and the Foundations of Mathematics, vol. 148. Elsevier (2003)
7. Goré, R., Widmann, F.: An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 437–452. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_32

8. Harel, D.: Dynamic logic. In: Gabbay, D.M., Günthner, F. (eds.) Handbook of Philosophical Logic, vol. II, pp. 497–604. D. Reidel, Dordrecht (1984)
9. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
10. Hustadt, U., Schmidt, R.A.: A comparison of solvers for propositional dynamic logic. In: Schmidt, R.A., Schulz, S., Konev, B. (eds.) Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, 14 July 2010. EPiC Series in Computing, vol. 9, pp. 63–73. EasyChair (2010)
11. Li, Y.: Knowing What to Do: A Logical Approach to Planning and Knowing How. Ph.D. thesis, University of Groningen (2017)
12. Yu, Q., Li, Y., Wang, Y.: More for free: a dynamic epistemic framework for conformant planning over transition systems. J. Logic Comput. **27**, 2383–2410 (2017)