Available online at www.sciencedirect.com

ScienceDirect

Journal of Applied Logic 6 (2008) 380–415

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Undoing the effects of action sequences ☆

Thomas Eiter [a,*], Esra Erdem [b,1], Wolfgang Faber [c]

[a] *Institute of Information Systems, Vienna University of Technology, Favoritenstraße 9-11, A-1040 Vienna, Austria*
[b] *Faculty of Engineering and Natural Sciences, Sabancı University, Orhanli, Tuzla, Istanbul 34956, Turkey*
[c] *Department of Mathematics, University of Calabria, Via P. Bucci, cubo 30b, 87036 Rende (CS), Italy*

Received 1 October 2006; accepted 2 May 2007

Available online 16 May 2007

Abstract

In this paper, we study the following basic problem: After having executed a sequence of actions, find a sequence of actions that brings the agent back to the state just before this execution. It emerges, for example, if an agent needs to find out which action sequences are undoable, and which ones are committed choices. A prototypical scenario is in the context of plan execution in a nondeterministic environment: Upon detecting that after executing some steps of the plan, an unintended state has been reached, backtracking to an earlier state by taking appropriate undo actions can be useful for recovery. In this paper, we consider the problem of undoing the effects of an action sequence by means of a reverse plan. Intuitively, by executing a reverse plan for an action sequence AS at the state S' reached after AS, the agent can always reach the state S she was at just before executing AS, possibly subject to conditions on the current state and S. Notably, this problem is different from a vanilla planning problem, since the state we have to get back to is in general unknown. We study this problem in a general logic-based action representation framework that can accommodate nondeterminism and concurrency. We formally define the notion of a reverse plan and determine the computational complexity of the existence and the recognition of such a plan. Guided by these results, we then present algorithms for constructing reverse plans. Unsurprisingly, the problem is intractable in general, and we present a knowledge compilation approach that constructs offline a reverse plan library for efficient (in some cases, linear time) online computation of action reversals. Our results for the generic framework can be adapted for expressive action languages such as $\mathcal{C}+$ or $\mathcal{K}$.

Keywords: Reasoning about actions; Undo actions; Reverse plans; Execution reversal; Computational complexity; Logic-based planning; Execution monitoring

1. Introduction

Reasoning about actions is an important area within knowledge representation and reasoning. Several logic-based languages for representing actions have been proposed (see e.g., [14,19,21,34]), and various reasoning problems about actions have been considered. The most prominent among them are temporal projection (inference about the

☆ Some of the results presented in this work appeared in [T. Eiter, E. Erdem, W. Faber, On reversing actions: Algorithms and complexity, in: M.M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, January 2007, pp. 336–341 [12]].

* Corresponding author.
E-mail addresses: eiter@kr.tuwien.ac.at (T. Eiter), esraerdem@sabanciuniv.edu (E. Erdem), faber@mat.unical.it (W. Faber).

1 Main part of the work carried out while visiting TU Wien.

1570-8683/$ – see front matter
doi:10.1016/j.jal.2007.05.002

state after a sequence of actions occurred), reasoning about the initial state after a sequence of actions occurred, and plan generation (generate a sequence of actions which takes the agent from an initial state to a goal state).

We study another reasoning problem about actions, namely the problem of undoing the effects of an execution of an action sequence, in a general logic-based framework. For example, after doing the action $go(home, office)$, the action $go(office, home)$ may reverse its effects and bring the agent back to her previous state. If this holds regardless of the state in which the agent was before doing $go(home, office)$ and afterwards, then $go(office, home)$ is called a *reverse action* for $go(home, office)$. If, more generally, a sequence of actions $R = B_1, \ldots, B_m$ is guaranteed to bring the agent back to the state before executing a sequence $AS = A_1, \ldots, B_n$, then R is called a *reverse plan* for AS. For example, $R = go(office, pub), go(pub, home)$ may be a reverse plan for $AS = go(home, bus\_stop), go(bus\_stop, office)$.

We note that the idea of undoing actions pervades several areas in computer science, like database systems [39], workflow and business process management [2,40], computer supported cooperative work [37], graphical editors [8], operating systems [4], and programming languages [3,26,41], but is also studied in related disciplines like cognitive science [32]. The paper [26] is recommended for an interesting historic survey of the idea of undoing in computer science.

Reverse actions and reverse plans are of interest in different scenarios, concerning either *a posteriori* or *a priori* reasoning about an action sequence, depending on whether the action sequence has already been executed or not. As for a priori reasoning, reverse plans are helpful to answer whether taking a certain sequence of actions is a committed choice, in the sense that its effects can not be reversed. For example, suppose there is a planned sequence of actions for a physical experiment (e.g., in a nuclear reactor), the outcome of which is uncertain. The existence of a reverse plan gives a guarantee that, however the experiment will evolve, one has the possibility to revert the system by taking its actions to the state before the experiment started.

The most prominent application of reverse plans in a posteriori reasoning is in error recovery. For this purpose, undo actions have been well-studied in the area of databases, where they are a standard method for error recovery [6]. In a more general context of plan execution and recovery, [23,24] use undo actions for execution of plans by mobile agents in a dynamic environment. However, the undo actions (one for each action) need to be specified (manually) by the user. It therefore is desirable to have tools which automatically generate undo actions, or more generally, reverse plans for backtracking. In fact, backtracking may be considered for various reasons, no only as a preliminary step to restart a plan (e.g., when the execution of the plan fails due to some undesired effects of an action in a nondeterministic environment). Another reason could be to switch from the current plan to one which is better (or safer) in the light of new information. When the current state and the state we want to backtrack to are known, then the problem amounts to a vanilla planning problem, which is intractable in general. However, the problem is different if the backtrack state is unknown.

The above considerations raise the following questions: given an action domain and an action A, does there exist a reverse action for A? More generally, given a sequence of actions AS, does there exist a reverse plan for AS? If so, how can a reverse action or plan be efficiently computed? From a computational point of view, can reverse actions or plans be fruitfully exploited for efficient backtracking in action execution? Motivated by these questions, we study action reversals and their computational aspects. The main contributions of this paper are as follows.

(1) We formally define the notions of a reverse action and a reverse plan for actions. Rather than to commit to a particular action language capable of modeling nondeterministic effects, such as the expressive action language $\mathcal{C}+$ [21] or $\mathcal{K}$ [15], we use here a generic transition-based framework for representing actions as in [31,38], using propositional logic as a specification language. In this framework, an action description can be represented by a transition diagram—a directed graph whose nodes correspond to states and whose edges correspond to action occurrences, which is described by propositional formulas. Besides nondeterminism, it also accommodates concurrent actions and dynamic worlds. Suppose that the execution of an action sequence AS at a state S leads to a state S'. Then, a *reverse plan* for AS is an action sequence R that always leads to the state S when executed at S'. When $|R| = 1$, it is called an *undo action* (or reverse action). There is salient different between this problem and a standard planning problem (which, in the terminology of planning, is a conformant or fail-safe planning problem [22,33]): while a planning problem always has a clearly defined goal (as a description which specifies acceptable states), there is no such goal for reverse plans, since the state which we have to revert to is, in general, unknown.

(2) We extend the above definitions to *conditional reversals*, considering also partial knowledge about the current state and the state before the execution. This is modeled by propositional formulas ϕ and ψ, such that the current state is a model of ϕ and respectively initial respectively ψ. By means of this, partial information about the states, such as sensory input, can be exploited to single out reverse plans which could not be adopted otherwise, since success might not be guaranteed.

(3) We thoroughly analyze the complexity of action reversals in the framework of [31,38], and characterize the complexity of recognizing and deciding existence of reverse actions and plans, both for plain as well as for conditional reversals. As we show, recognizing a reverse actions resp. a reverse plan of polynomial length for a sequence of actions is complete for the class coNP respectively Π_2^p from the polynomial hierarchy, and deciding the existence of such a plan (which intuitively can be nondeterministically guessed and checked) is Σ_2^p-complete respectively Σ_3^p-complete. Furthermore, we determine the boundary between Σ_2^p and Σ_3^p, in terms of the reverse plan length, which turns out to be surprising.
Deciding the existence of conformant plans is, and thus of the same degree of complexity as computing a reverse plan for a given action sequence.

(4) Due to their logic-based definitions, reverse plans for actions sequences can be readily computed, following Rintanen's approach [31], using a solver for Quantified Boolean Formulas (QBFs). We show that this can also be accomplished by translating the problem to a conformant planning problem and using a conformant planner.

(5) For efficient online computation, we present an algorithm which, given a sequence of actions $AS = A_0, \ldots, A_{i-1}$, an associated sequence of percepts $\Pi = \pi_0, \ldots, \pi_i$ about the states $S_0, \ldots, S_i$ in the execution, and a reverse plan library L, assembles a reverse plan for AS and Π from L. Each percept π_j is a formula which is satisfied by the respective state S_j, and informally represents information that we have about S_j; in the extremal cases, π_j has the single model S_j, which means we have full information, respectively π_j is a tautology, which means that we have no information. The algorithm is of low-order polynomial time for important classes of conditions, and, under reasonable assumptions met in practice, it runs in linear time in the worst case.

Our results shed light on the complexity of action reversals, and can be easily customized to particular action languages like $\mathcal{C}+$ [21] or $\mathcal{K}$ [14]. Furthermore, the transformations and algorithms may be adapted to use systems like CCALC [21,29], CPLAN [16,20], and DLV$^{\mathcal{K}}$ [14] for reasoning about actions; note that CPLAN and DLV$^{\mathcal{K}}$ support conformant planning.

In the context of execution monitoring of a single agent, reverse plans can sometimes be used for recovering from discrepancies, as seen in the second example above. We consider this method of recovery, as a component of a richer recovery framework, in which a particular recovery method is selected ad hoc from a suite of such methods. This method is appealing when a short reverse plan can be found, and when the remainder of the plan is comparatively long (e.g., the first action A of a long plan leads to a state where the rest of the plan is not executable, and A has an undo action in the library). Note that other logic-based execution monitoring frameworks [9,17,18,35,36] do not consider undo actions (or reverse plans) for recovering from discrepancies. In this sense, our algorithms for reverse plan assembly suggest action reversal as a complementary method for efficient backtracking (if no reverse plan exists, choose some other method).

In the following, we first present the action representation framework which we consider (Section 2). Then, we precisely describe the reverse of an action and an action sequence, and extend these definitions to handle various cases (Section 3). In the next section, we briefly discuss how reversals can be used to recover from discrepancies in execution monitoring (Section 4). After that, we turn to computational issues. We first analyze the computational complexity of execution reversal (Section 5), followed by a discussion of how to compute reversals offline (Section 6). Efficient online assembly of reversal plans for plan recovery is considered in (Section 7). We conclude with a discussion of the related work (Section 8) and future work (Section 9). In order not to disrupt the flow of reading, the proofs of all theorems have been moved to the appendix.

2. Action representation framework

We consider the action representation and planning framework described in [38], which is similar to that of [31].

We begin with a set $\mathcal{A}$ of action symbols and a disjoint set $\mathcal{F}$ of fluent symbols. Let $state(\mathcal{F})$ be a formula in which the only nonlogical symbols are elements of $\mathcal{F}$. This formula encodes the set of states that correspond to its models.

Let $act(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ be a formula whose only nonlogical symbols are elements of $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$, where $\mathcal{F}'$ is obtained from $\mathcal{F}$ by priming each element of $\mathcal{F}$. Then the formula

$$state(\mathcal{F}) \wedge act(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge state(\mathcal{F}') \tag{1}$$

encodes the set of transitions that corresponds to its models. That is,

(i) the start state corresponds to an interpretation of (the symbols in) $\mathcal{F}$,[2]
(ii) the set of actions executed corresponds to an interpretation of $\mathcal{A}$, and
(iii) the end state corresponds to an interpretation of $\mathcal{F}'$.

Formula (1) is abbreviated as $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$.

Example 2.1. (See [21].) Putting a puppy into water makes the puppy wet, and drying a puppy with a towel makes it dry. With the fluents

$$\mathcal{F} = \{inWater, wet\},$$

and the actions

$$\mathcal{A} = \{putIntoWater, dryWithTowel\},$$

the states can be described by the formula

$$state(\mathcal{F}) = inWater \supset wet.$$

Since there are three interpretations of $\mathcal{F}$ satisfying $state(\mathcal{F})$

$$\{inWater, wet\}, \{\neg inWater, wet\}, \{\neg inWater, \neg wet\}$$

there are three states: $\{inWater, wet\}, \{wet\}, \{\}$.

The action occurrences can be defined as follows:

$$\begin{aligned} act(\mathcal{F}, \mathcal{A}, \mathcal{F}') = {} & (inWater' \equiv inWater \vee putIntoWater) \ \wedge \\ & (wet' \equiv (wet \wedge \neg dryWithTowel) \vee putIntoWater) \ \wedge \\ & \big(dryWithTowel \supset (\neg inWater \wedge \neg putIntoWater)\big). \end{aligned}$$

The last line of the formula above expresses that *dryWithTowel* is executable when *inWater* is false and it is not executable concurrently with *putIntoWater*.

For instance, the interpretation

$$\{\neg inWater, wet, dryWithTowel, \neg putIntoWater, \neg inWater', \neg wet'\}$$

satisfies $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$, therefore it describes a transition:

$$\big\langle \{wet\}, \{dryWithTowel\}, \{\} \big\rangle. \tag{2}$$

Note that the interpretation of $\mathcal{A}$ above describes the occurrence of the single action *dryWithTowel*.

The meaning of a domain description can be represented by a transition diagram—a directed graph whose nodes correspond to states and whose edges correspond to action occurrences. In a transition diagram, a "trajectory" of length n is obtained by finding a model of the formula

$$tr_n(\mathcal{F}, \mathcal{A}) = \bigwedge_{t=0}^{n-1} tr(\mathcal{F}_t, \mathcal{A}_t, \mathcal{F}_{t+1})$$

[2] In the rest of the paper, we sometimes say "interpretation of $\mathcal{S}$" to mean an interpretation of the symbols in a set $\mathcal{S}$.

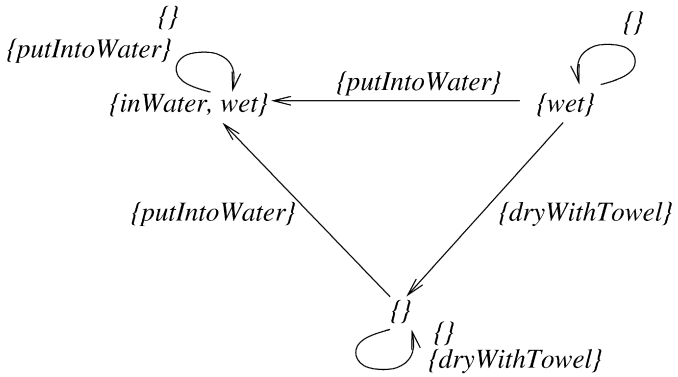


Fig. 1. The transition diagram for the action description of Example 2.1.

where each $\mathcal{F}_i$ (resp., each $\mathcal{A}_i$) is the set of fluents (resp., actions) obtained from $\mathcal{F}$ (resp., $\mathcal{A}$) by adding time stamp i to each fluent symbol (resp., each action symbol). The trajectory is the alternating sequence of states and action occurrences that correspond to the interpretation of fluents and actions respectively. The trajectory is of the form

$$S_0, A_0, S_1, \ldots, S_{n-1}, A_{n-1}, S_n$$

where each S_i is the state that corresponds to the interpretation of $\mathcal{F}_i$, and each A_i represents the action occurrences that correspond to the interpretation of $\mathcal{A}_i$.

Example 2.2. The transition diagram for the action description of Example 2.1 is presented in Fig. 1. In this diagram, paths of length 1 describe transitions, and paths of length n describe trajectories. For instance, the path

$$\{wet\}, \{\}$$

describes transition (2), and the path

$$\{wet\}, \{\}, \{inWater, wet\}$$

describes the trajectory

$$\{wet\}, \{dryWithTowel\}, \{\}, \{putIntoWater\}, \{inWater, wet\}$$

where a wet puppy is first dried with a towel and then put into the water. This trajectory is obtained from the following interpretation of $\mathcal{F}_0 \cup \mathcal{A}_0 \cup \mathcal{F}_1 \cup \mathcal{A}_1 \cup \mathcal{F}_2$

$$\{\neg inWater_0, wet_0, \neg putIntoWater_0, dryWithTowel_0,$$
$$\neg inWater_1, \neg wet_1, putIntoWater_1, \neg dryWithTowel_1,$$
$$inWater_2, wet_2\}$$

that satisfies the conjunction $tr(\mathcal{F}_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, \mathcal{A}_1, \mathcal{F}_2)$. Note that in this example a single action occurred at any time. In general, no, one or many actions may occur at a time.

We can talk about more specific states, transitions, or trajectories by applying some "substitutions" to the formulas describing them. Consider, for instance, Example 2.2. To find states reachable from the state $S = \{wet\}$, we need to find the transitions of the form $\langle S, A, S' \rangle$. These transitions can be described by substituting S for $\mathcal{F}$ in $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$, that is, by the formula $tr(S, \mathcal{A}, \mathcal{F}')$ obtained from $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ by replacing every atom $p \in \mathcal{F}$ with $\top$ if $p \in S$, and with $\bot$ otherwise.

Similarly, trajectories S_0, A_0, S_1, A_1, S_2 of length 2 with the initial state $S_0 = \{\}$ and the action occurrence $A_1 = \{dryWithTowel\}$ at time stamp 1 can be expressed by $tr(S_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, A_1, \mathcal{F}_2)$. Here, $tr(S_0, \mathcal{A}_0, \mathcal{F}_1)$ is the formula obtained from $tr(\mathcal{F}_0, \mathcal{A}_0, \mathcal{F}_1)$ by replacing every atom $p_0 \in \mathcal{F}_0$ with $\top$ if $p \in S_0$, and with $\bot$ otherwise. Similarly, $tr(\mathcal{F}_1, A_1, \mathcal{F}_1)$ is the formula obtained from $tr(\mathcal{F}_1, \mathcal{A}_1, \mathcal{F}_2)$ by replacing every atom $p_1 \in \mathcal{A}_1$ with $\top$ if

$p \in A_1$, and with $\bot$ otherwise. The conjunction $tr(S_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, A_1, \mathcal{F}_2)$ can be replaced by $tr_2(S_0, A_1)$. Similarly, the conjunction $tr(S_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, A_1, \mathcal{F}_2) \wedge tr(\mathcal{F}_2, A_2, S_3)$ is denoted by $tr_3(S_0 \cup S_3, A_1 \cup A_2)$.

Example 2.3. Consider a version of the blocks world in which a block B can be thrown from location $L1$ to another location $L2$. Unfortunately, throwing is not accurate and so the block may end up on any location, and not necessarily on $L2$.

With the fluents $on(B, L)$ ("block B is on location L"), and the actions $throw(B, L, L1)$ ("throw block B from location L to location $L1$"), the states, i.e., $state(\mathcal{F})$, can be defined by the conjunction of the following formulas:[3]

- every block should be on some location:

$$\bigvee_{L} on(B, L);$$

- if a block is on some location then it is not anywhere else:

$$on(B, L) \supset \bigwedge_{L \neq L1} \neg on(B, L1); \tag{3}$$

- a block cannot have more than one block on itself:

$$on(B1, B) \supset \bigwedge_{B1 \neq B2} \neg on(B2, B);$$

- every block is "supported" by the table (here $supported(B)$ is an auxiliary propositional variable defined in terms of $on(B, L)$):[4]

$$supported(B).$$

Formula $act(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ can be defined by the conjunction of the following formulas:

- the preconditions of $throw(B, L, L1)$:

$$throw(B, L, L1) \supset \left(on(B, L) \wedge \bigwedge_{B1} \neg on(B1, B) Q \right); \tag{4}$$

- the effects of $throw(B, L, L1)$, and inertia:

$$on(B, L1)' \supset \left(on(B, L1) \vee \bigvee_{L, L2} throw(B, L, L2) \right); \tag{5}$$

- no-concurrency:[5]

$$throw(B, L, L1) \supset \bigwedge_{B1 \neq B, L2, L3} \neg throw(B1, L2, L3) \wedge \bigwedge_{L2 \neq L1} \neg throw(B, L, L2). \tag{6}$$

In the following sections, an expression of the form $\mathcal{F} \equiv \mathcal{F}'$ denotes the conjunction $\bigwedge_{f \in F} f \equiv f'$.

[3] In the following, $B, B1, B2$ range over a finite set of block constants, and $L, L1, L2, L3$ range over the set of location constants that consists of the set of block constants and the constant *table*.

[4] For instance, for a domain with three blocks, we can define $supported(B)$ by the disjunction $on(B, table) \vee \bigvee_{B \neq B1} (on(B, B1) \wedge on(B1, table)) \vee \bigvee_{B \neq B1, B1 \neq B2, B \neq B2} (on(B, B1) \wedge on(B1, B2) \wedge on(B2, table))$.

[5] Note that, any concurrent action involving $throw(B, L, L1)$ and $throw(B, L2, L3)$ is not executable due to the state constraint (3).

3. Action execution reversals

When an agent has executed a sequence of actions $\langle A_0, \ldots, A_i \rangle$, it sometimes may be desirable for her that the effects of the whole or part of the action sequence be undone, such that the agent is back in the state S_j, $j < i$, which she had reached after taking the actions $A_0, \ldots, A_{j-1}$.

A prototypical application scenario for this is execution monitoring. Consider an agent that wants to execute the action sequence $\langle A_0, \ldots, A_{n-1} \rangle$ in a nondeterministic environment, as seen in the examples of Section 1. Suppose that she reaches a state S_j after executing the actions $A_0, \ldots, A_{j-1}$. She discovers that there is something wrong at S_j and wants to go back to $i < j$ so that she can retry. One way to go back would be to try and reach the earlier state S_i by solving a planning problem with the initial state S_j and the goal state S_i, and then to execute the computed plan of some given length. This, however, can only work if S_i is known to the agent, that is if she has logged the states. Another way to go back to i is to undo the sequence $\langle A_{i+1}, \ldots, A_{j-1} \rangle$ of actions at S_j. This has the advantage that S_i does not have to be known. This application scenario is discussed in detail in Section 4.

Note, however, that execution reversal is not bound to execution monitoring, and that an agent might for other reasons decide to backtrack, as discussed in the introduction.

An action can be undone by executing one of its "reverse actions" or by executing a "reverse plan". In general, an action sequence can be undone by executing one of its "reverse plan". In the following, we will make these three sorts of reversals precise.

3.1. Reverse actions and plans

We define a reverse of an action below relative to a given action description represented by a transition diagram.

Definition 3.1. An action A' is a *reverse action* for A, if, for all $\mathcal{F}$ and $\mathcal{F}'$, the formula

$$revAct(\mathcal{F}, \mathcal{F}'; A, A') = tr(\mathcal{F}, A, \mathcal{F}') \supset \big(tr(\mathcal{F}', A', \mathcal{F}) \wedge \forall \mathcal{F}'' \big(tr(\mathcal{F}', A', \mathcal{F}'') \supset \mathcal{F} \equiv \mathcal{F}''\big)\big) \tag{7}$$

is a tautology (i.e., $\forall \mathcal{F} \forall \mathcal{F}' revAct(\mathcal{F}, \mathcal{F}'; A, A')$ evaluates to true).

The formula above expresses the following condition about actions A and A'. Take any two states S, S' (described by the interpretations of fluents in $\mathcal{F}$ and $\mathcal{F}'$ respectively) such that executing A at S leads to S'. Then executing A' at state S' always leads to S. (See Fig. 2.)

Example 3.1. In the setting of Example 6.1, $carry(B, L)$ is a reverse action for $throw(B, L, L1)$. Indeed, consider any two states S, S' such that, for some block B, and for some locations $L, L1$, executing $throw(B, L, L1)$ at state S leads to state S'. Due to the preconditions of $throw(B, L, L1)$, i.e., (4), we know that S is a state at which block B is on location L and block B is clear. Due to the nondeterministic effect of $throw(B, L, L1)$ and due to inertia, i.e., (14), we know that S' is a state at which block B is on some location $L2$, block B is clear, and other blocks are in the same locations as they are in S. Due to the preconditions of $carry(B, L)$, i.e., (12), we can execute $carry(B, L)$ at state S'. Due to the deterministic effect of $carry(B, L)$ and due to inertia, i.e., (14), carrying block B onto location L leads to the state S'' at which B is on location L, and other blocks are in the same locations as they are in S. That is, $S'' = S$.

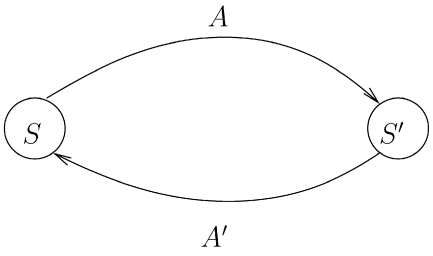


Fig. 2. Action A' is a reverse action for action A.

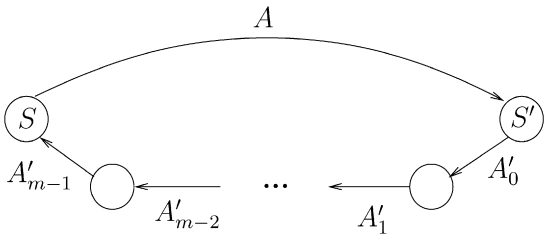


Fig. 3. Action sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ is a reverse plan for action A.

Many of the benchmarks used in some AIPS/ICAPS planning competitions[6] are from the transportation domain (logistics, blocks world, grid, etc.) described by STRIPS-style operators. Many of the actions in this domain are reversible in this sense (moving from x to y is the reverse of moving from y to x, putting down an object is the reverse of picking up an object, etc.). Consider such a deterministic action domain where each state is represented by a set of atoms, and each action is represented by a STRIPS-style operator consisting of three lists of atoms, a precondition list, an add list, and a delete list. The execution of an operator (P, A, D) at a state S such that $P \subseteq S$ results in the state S' is defined by $(S \setminus D) \cup A$. Then, in accordance with Definition 3.1, about two STRIPS-style operators $\alpha = (P, A, D)$ and $\alpha' = (P', A', D')$, we can say that α' is a reverse action for α if, for every two states S and S' such that $P \subseteq S$ and $S' = (S \setminus D) \cup A$, it holds that $P' \subseteq S'$ and $S = (S' \setminus D') \cup A'$.

Definition 3.2. A *reverse plan for an action* A is a sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ $(m \geqslant 0)$ of actions such that, for all $\mathcal{F}$ and $\mathcal{F}'$, the following formula holds:

$$revPlan\big(\mathcal{F}, \mathcal{F}'; A, [A'_0, \ldots, A'_{m-1}]\big) = tr(\mathcal{F}, A, \mathcal{F}') \supset \forall \mathcal{F}_0, \ldots, \mathcal{F}_m \exists \mathcal{F}'_1, \ldots, \mathcal{F}'_m \Bigg(\mathcal{F}_0 \equiv \mathcal{F}' \supset \Bigg(\bigwedge_{t=0}^{m-1} \big(tr_t(\mathcal{F}, A') \supset tr(\mathcal{F}_t, A'_t, \mathcal{F}'_{t+1})\big) \wedge \big(tr_m(\mathcal{F}, A') \supset \mathcal{F}_m \equiv \mathcal{F}\big)\Bigg)\Bigg). \tag{8}$$

The formula above expresses the following condition about an action A and an action sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$. Take any two states S, S' (described by the interpretations of fluents in $\mathcal{F}$ and $\mathcal{F}'$ respectively) such that executing A at S leads to S'. Then the action sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ is executable at state S', and it always leads to S. (See Fig. 3.) The executability condition of $\langle A'_0, \ldots, A'_{m-1}\rangle$ is described above by the formula $\bigwedge_{t=0}^{m-1}(tr_t(\mathcal{F}, A') \supset tr(\mathcal{F}_t, A'_t, \mathcal{F}'_{t+1}))$. Note that $revPlan(\mathcal{F}, \mathcal{F}'; A, [A'_0])$ is equivalent to $revAct(\mathcal{F}, \mathcal{F}'; A, A'_0)$.

3.2. Multi-step reversals

We can further generalize the notion of reversing by considering plans, rather than actions, to be reversed. There are two motivations for this generalization: It might not always be possible to find reverse plans for single actions, but only for sequences of actions. Also, a reverse plan for an action sequence might be shorter than a reverse plan obtained by concatenating reverse plans for subsequences (as in Example 3.2).

Definition 3.3. A sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ $(m \geqslant 0)$ of actions is a *reverse plan for an action sequence* $\langle A_0, \ldots, A_{k-1}\rangle$ $(k > 0)$, if, for all $\mathcal{F}$ and $\mathcal{F}'$, the following formula is true:

$$multiRev\big(\mathcal{F}, \mathcal{F}'; [A_0, \ldots, A_{k-1}], [A'_0, \ldots, A'_{m-1}]\big) = \exists \mathcal{F}_0, \ldots, \mathcal{F}_k \big(\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A) \wedge \mathcal{F}' \equiv \mathcal{F}_k\big) \supset \forall \mathcal{F}'_0, \ldots, \mathcal{F}'_m \exists \mathcal{F}''_1, \ldots, \mathcal{F}''_m (\mathcal{F}'_0 \equiv \mathcal{F}' \supset \bigwedge_{t=0}^{m-1} \big(tr_t(\mathcal{F}', A') \supset tr(\mathcal{F}'_t, A'_t, \mathcal{F}''_{t+1})\big) \wedge \big(tr_m(\mathcal{F}', A') \supset \mathcal{F}'_m \equiv \mathcal{F}\big). \tag{9}$$

[6] See http://www.cs.colostate.edu/meps/repository/aips98.html.

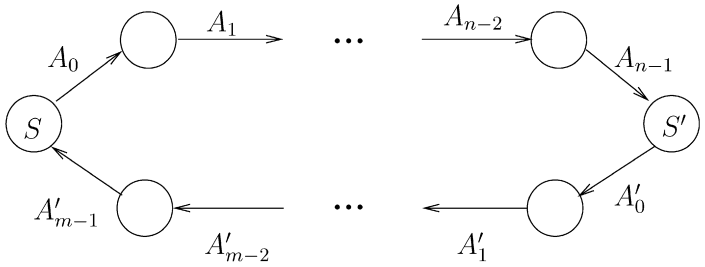


Fig. 4. Action sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ is a reverse plan for action sequence $\langle A_0, \ldots, A_{n-1}\rangle$.

The formula above is very similar to $revPlan(\mathcal{F}, \mathcal{F}'; A, [A_0, \ldots, A_{m-1}])$. The only difference is that, in the premise of the formula, a trajectory is considered instead of a single transition. (See Fig. 4.) Note that $multiRev(\mathcal{F}, \mathcal{F}'; [A_0], [A'_0, \ldots, A'_m])$ is equivalent to $revPlan(\mathcal{F}, \mathcal{F}'; A_0, [A'_0, \ldots, A'_m])$.

Example 3.2. In the setting of Example 6.1, a reverse plan for the action sequence

$$\big\langle throw(B1, L1, L2), carry(B1, L3), throw(B1, L3, L4), carry(B1, L5)\big\rangle$$

is $\langle carry(B1, L1)\rangle$. Indeed, executing the action sequence above at a state S changes the location of block $B1$ from location $L1$ to location $L5$, without changing the locations of other blocks. Carrying block $B1$ to location $L1$ at this new state brings the blocks world back to its state S.

3.3. Conditional reversals

In the above, a reverse plan is defined for an action sequence at any state reachable by that sequence. However, at some such states, an action sequence may not admit any reverse plan. That is, an action sequence may have a reverse plan under some conditions, that do not necessarily hold at every reachable state. Furthermore, if some information about the state which we want to reach by reversing actions is available, e.g., values of some fluents obtained by sensing, then the applicability of a reverse plan might be possible depending on this information. To make execution reversals applicable in such situations, we extend the concept of a reverse plan to a "conditional reverse plan" that takes state information into account.

Definition 3.4. A sequence $\langle A'_0, \ldots, A'_{m-1}\rangle$ $(m \geqslant 0)$ of actions is a *$\phi; \psi$-reverse plan for an action sequence* $\langle A_0, \ldots, A_{k-1}\rangle$ $(k > 0)$ if, for any $\mathcal{F}$ and $\mathcal{F}'$, the formula

$$\psi(\mathcal{F}) \wedge \phi(\mathcal{F}') \supset multiRev\big(\mathcal{F}, \mathcal{F}'; [A_0, \ldots, A_{k-1}], [A'_0, \ldots, A'_{m-1}]\big) \tag{10}$$

is true. Here $\phi(\mathcal{F}')$ and $\psi(\mathcal{F})$ are formulas with all nonlogical symbols in $\mathcal{F}'$ and $\mathcal{F}$, respectively.

For the case where $\psi(\mathcal{F}) \equiv \top$, we simply write ϕ-reverse plan for $\phi; \psi$-reverse plan.

Example 3.3. Consider a variant of Example 6.1 with another deterministic action: $carry2(B, L, L1)$ (i.e., the action of carrying a block B from a location L to another location $L1$). Note that $\langle carry2(B, L4, L1)\rangle$ is not a reverse plan for the action sequence

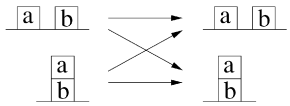
$$\big\langle throw(B, L1, L2), carry(B, L3), throw(B, L3, L4)\big\rangle,$$

as B need not be on $L4$ after the execution of this action sequence, due to the nondeterministic effects of $throw(B, L3, L4)$. However, it is a ϕ-reverse plan, with $\phi(\mathcal{F}')$ being $on(B, L4)'$.

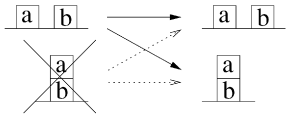
The example above shows that ϕ-reverse plans relax the notion of a reverse plan. The next example shows that action sequences that do not have reverse plans but only ϕ-reverse plans for uninteresting cases (e.g., ϕ is not satisfied at any state where the given action sequence is executable), may possess $\phi; \psi$-reverse plans.

Example 3.4. Consider a variant of the action domain of Example 6.1, with the nondeterministic action *throw1*(B, L) of throwing a block B to a location L. The difference between *throw* of Example 6.1 and *throw1* is that *throw1* does not specify where the block is to be picked up.

Assume that there are two blocks a and b, and the block b is mounted on the table (i.e., at every state $on(b, Table)$ holds). Then the following transitions can occur when executing *throw1*(a, b):



We can see that no ϕ-reverse plan exists for $\langle throw1(a, b)\rangle$, if $\phi \equiv on(b, Table)$: there is no action sequence AS that, when executed just after *throw1*(a, b) at a state S, always results in S. (Otherwise, i.e., $\phi \equiv \neg on(b, Table)$, any plan is trivially ϕ-reverse for $\langle throw1(a, b)\rangle$; but this is not an interesting case, since there is no state where b is not on the table.) On the other hand, with $\phi \equiv \top$ and $\psi \equiv on(a, table)$,



$\langle carry(a, table)\rangle$ is a ϕ; ψ-reverse plan for $\langle throw1(a, b)\rangle$.

A question which comes up naturally is whether it is possible to formulate conditions, which are necessary or sufficient for the existence of a reverse action for a given action. In the following, we briefly discuss two conditions, of which one is necessary, while the other one is sufficient.

Let us first focus on the necessary condition. Imagine the following situation: The action A, which is to be reversed, results in the same state S when executed in two different states S' and S'', i.e., $tr(S', A, S)$ and $tr(S'', A, S)$ both hold. It is then impossible to find a reverse plan $\langle A'_0, \ldots, A'_{m-1}\rangle$ for A. If it would, then if some $S_0, \ldots, S_m$ exist such that $tr_m(S, A')$, then both $S_m = S'$ and $S_m = S''$ must hold, which is impossible, as we assumed that $S' \neq S''$. This necessary condition can be stated more generally as follows:

Proposition 1. *If a ϕ; ψ-reverse plan for an action sequence $A = \langle A_0, \ldots, A_{n-1}\rangle$ exists, then, for every two sequences $S = S_0, \ldots, S_n$ and $S' = S'_0, \ldots, S'_n$ of states such that $tr_n(S, A)$, $tr_n(S', A)$, $\psi(S_0)$, and $\psi(S'_0)$ hold, it holds that $S_n \neq S'_n$.*

For instance, in Example 3.4, for the states $\{on(a, b), on(b, Table)\}$ and $\{on(a, Table), on(b, Table)\}$, the execution of the action *throw1*(a, b) can lead to the same state. Hence, due to Proposition 1, no ϕ-reverse plan exists for $\langle throw1(a, b)\rangle$, if $\phi \equiv on(b, Table)$.

We can also find a sufficient condition, motivated by the following property of functions: A function f is *involutory* iff $f(f(x)) = x$ for each x in the domain of f. We say that an action sequence $A_0, \ldots, A_{m-1}$ is (ψ-)*involutory*, if, for every state S (satisfying ψ), the following hold:

- for every sequence $S = S_0, \ldots, S_m$ of states such that $tr_m(S, A)$ holds, there exist a sequence $S_m = S'_0, \ldots, S'_m = S$ of states such that $tr_m(S', A)$ holds;
- for every two sequences $S = S_0, \ldots, S_m$ and $S_m = S'_0, \ldots, S'_m$ of states such that $tr_m(S, A) \wedge tr_m(S', A)$ holds, it holds that $S'_m = S$.

Therefore, an action is involutory, if executing the action twice in any state, where the action is executable, always results in the starting state. An example of an involutory action is a *toggle* action: If a simple light switch is toggled twice, it will always be in the same state as before. Then a sufficient condition can be stated as follows:

Proposition 2. *A ψ-involutory action sequence is always $\top; \psi$-reversible, and a $\top; \psi$-reverse plan is precisely the action sequence itself.*

4. An application: Execution monitoring

As already pointed out in the previous section, a prototypical application of execution reversal is execution monitoring of plans. We briefly discuss and illustrate in this section how reverse plans can be fruitfully used in this context. To this, let us first define the notion of a plan in the action representation framework defined in Section 2.

In a planning problem, an initial state (which is not necessarily unique) is described by a formula $init_{rev}(\mathcal{F})$ such that $init_{rev}(\mathcal{F}) \models state(\mathcal{F})$, and the goal is described by a formula $goal_{rev}(\mathcal{F})$; in both $init_{rev}(\mathcal{F})$ and $goal_{rev}(\mathcal{F})$, the only nonlogical symbols are elements of $\mathcal{F}$.

A *plan of length n* for the planning problem is obtained from any model of the formula

$$init_{rev}(\mathcal{F}_0) \wedge tr_n(\mathcal{F}, \mathcal{A}) \wedge goal_{rev}(\mathcal{F}_n)$$

as the sequence

$$\langle A_0, \ldots, A_{n-1} \rangle$$

of action occurrences A_i that correspond to the interpretation of $\mathcal{A}_i$. We will use the notation $|P|$ to denote the length of a plan P. Note that in general this does not correspond to the numbers of actions in P.

Example 4.1. Consider, in the action domain of Example 2.3, the blocks a, b, c, d, and a planning problem $\mathcal{P}$ with the initial state and the goal state as follows:

A plan of length 3 for the planning problem $\mathcal{P}$ is[7]

$$P = \big\langle throw(a, c, d), throw(b, table, c), throw(a, d, b) \big\rangle.$$

When an agent is situated in a nondeterministic environment, execution of a plan may need to be monitored to ensure that the plan does not fail to achieve the goal. For example, imagine a shopping agent that accidentally picks the wrong, but more expensive, milk from a shelf. Later, at the cashier, she might realize that she does not have enough money to pay, and the remainder of her shopping plan is obsolete.

Execution monitoring may help to reveal that things go wrong and to recover from any detected execution failure. To this end, the agent may determine a discrepancy between the actual and the expected state of the world. Such a discrepancy usually implies a failure (at least in the agent's belief), for which a recovery should be sought. A diagnosis of the discrepancy can be useful to find a reasonable plan recovery. In the previous scenario, by monitoring, the agent might discover earlier the wrong milk in the shopping cart and conclude that she did not pick the right one. She then can return the expensive milk, grab the right one instead, and continue with the execution of the rest of her shopping plan.

Note that, alternatively, the agent may ensure that the execution of the plan succeeds, by constructing a "conditional plan" which takes into account all possible contingencies. The agent finds out which subplan to execute by information obtained through "sensing actions" in the plan. The shopping agent may include a sensing action in her plan to check the brand and the price of the milk after grabbing the milk. If it is the right one, she can continue with the rest of her intended plan, otherwise switch to some other plan. However, no conditional plan may exist in general, or constructing and storing it might be prohibitively expensive, since conditional plans can have exponential size. Note

[7] We sometimes drop curly brackets from singleton action occurrences in a plan. For instance, in P, singleton action occurrences of the form $\{throw(B, L, L1)\}$ are written as $throw(B, L, L1)$.

that for plans of polynomially-bounded length, conditional planning is PSPACE-complete [38, Theorem 4] whereas classical planning is NP-complete [38, Theorem 1]. Also most of the logic-based systems (e.g., DLV$^{\mathcal{K}}$ or CPLAN) can only compute linear plans. In execution monitoring, the agent usually does not include sensing actions in the plan, but instead monitors the execution of a reasonable classic plan for possible points of failures, and deals with them when they cause a failure or a discrepancy.

We consider a framework for monitoring the execution of a plan relative to a set of intended trajectories. According to this framework, the monitoring agent (possibly different from the agent executing the plan), from time to time,

(1) checks whether there is a discrepancy between the current state and the corresponding states of the given trajectories relative to the plan;
(2) if no discrepancy is detected then continues with the execution of the plan; otherwise, may try to find a diagnosis of discrepancies by examining the given trajectories against evolutions of the current state;
(3) if a diagnosis is found then may recover from the discrepancies by backtracking to the diagnosed point of failure and executing the plan from that point on; otherwise, finds another plan (via other recovery methods) from the current state to reach a goal state.

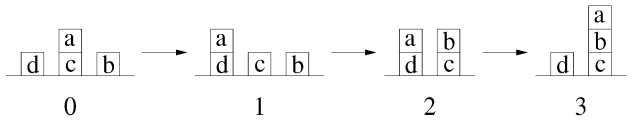
Steps 1–3 of the framework above are presented in Fig. 5 as well.

Here is an example describing the framework above.

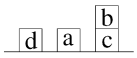
Example 4.2. Consider the blocks world described in Examples 2.3 and 6.1. Suppose that an agent is executing the plan P, i.e.,[8]

$$\langle \mathit{throw}(a,c,d), \mathit{throw}(b,\mathit{table},c), \mathit{throw}(a,d,b)\rangle,$$

and that she is given the following intended trajectory T_P:

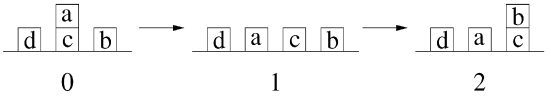


Suppose also that, while the agent is executing the plan P, at time stamp 2, the following state S_2



is observed. Since this observed state is different from the expected state at time stamp 2 according to T_P, a discrepancy can be detected.

When such a discrepancy is detected, the agent can look for an explanation, i.e., a point of failure. For that, the agent finds out how S_2 may have evolved from the initial state while executing P. In this case, there is one evolution E_P of the state S_2 reached at time stamp 2 from the initial state S_0:



The point of failure for the detected discrepancy above is the state S_0 at time stamp 0 because the evolution E_P of S_2 "deviates" from the trajectory T_P at time stamp 0 in state S_0. That is, the states of T_P and E_P are identical at time stamp 0, but they differ at time stamp 1. The reason is that, in E_P, block a ended up on the table rather than on block d when executing $\mathit{throw}(a,c,d)$.

This point of failure can be used as a checkpoint if the agent executes this plan often. On the other hand, since such a discrepancy may prevent the execution of the plan later on, it can be used to recover from the discrepancy above. In

[8] One might wonder why the agent executes the plan P above instead of $\langle \mathit{carry}(a,d), \mathit{carry}(b,c), \mathit{carry}(a,b)\rangle$. One reason can be that, in this domain, carrying a block is more expensive (e.g., more time-consuming) than throwing a block.

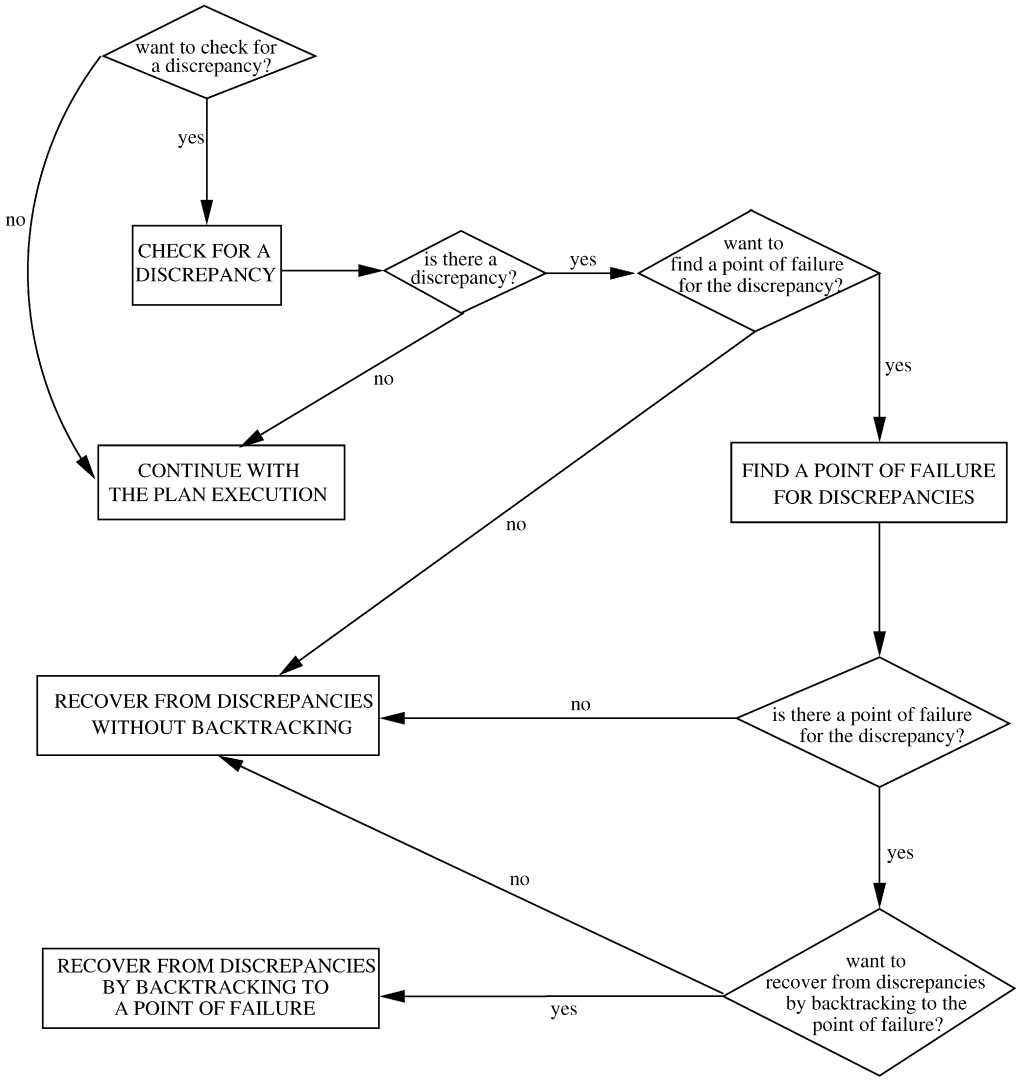


Fig. 5. Steps 1–3 of the monitoring framework described in Section 4.

particular, the agent can backtrack to the state S_0 from the state S_2 (e.g., undo the plan execution until the diagnosed point of failure S_0 by carrying block b onto the table and then by carrying block a onto block c) and then execute P again.

As seen in Fig. 5, in a more general framework, at Step 2, the agent may check whether the detected discrepancies are "relevant" to the successful execution of the rest of the plan, like in [9,35,36], considering some other possible trajectories as well. Also, the agent does not have to find a diagnosis of discrepancies every time there is a detected discrepancy. When to look for a diagnosis can be handled by a decision support model.

On the other hand, Step 3 of the framework can be refined further. Depending on the diagnosis of discrepancies, the length of the plan executed so far, the length of the remaining plan, and possibly some other criteria, the agent, with the help of a decision support model, can pick a plan recovery method among many, including replanning, backtracking, and patch planning.

Extensions and refinements of the framework described above are possible. More details about points of failure can be found in [13]. In the following, we will concentrate on how backtracking to a diagnosed point of failure can be

done, like in Example 4.2, and study this problem in the formal framework described in Section 2. In particular, we will study backtracking by plan reversals avoiding (re)planning.

5. Complexity results

We study the complexity of the following problems related to the computation of execution reversals with respect to a given action domain description:

(P1) for two given action sequences AS and R, and given formulas ϕ and ψ composed of fluent symbols, recognizing whether R is a $\phi;\psi$-reverse plan for AS;
(P2) for a given action sequence AS, deciding whether there exist an action sequence R of a polynomially bounded length, and formulas ϕ and ψ composed of fluent symbols, such that R is a $\phi;\psi$-reverse plan for AS, and that $\phi(S')$ holds for some state S' reached by AS from some state S such that $\psi(S)$ holds;
(P3) for a given action sequence AS, and given formulas ϕ and ψ composed of fluent symbols, deciding whether there exists an action sequence R of a polynomially bounded length such that R is a $\phi;\psi$-reverse plan for AS.

The polynomial hierarchy. For our discussion of the computational complexities of these problems, recall the following sequence of classes from the polynomial hierarchy: First, $\Sigma_0^p = \Pi_0^p = \mathrm{P}$; and for all $k \geqslant 1$, $\Sigma_k^p = \mathrm{NP}^{\Sigma_{k-1}^p}$ and $\Pi_k^p = \mathrm{coNP}^{\Sigma_{k-1}^p}$. Each complexity class at each level k ($k \geqslant 1$) of the hierarchy, includes all complexity classes at lower levels.

Several problems on planning and reasoning about actions which are complete for these classes for $k \leqslant 3$, can be found in [5,15,31,38]. In the first level of this hierarchy, $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \mathrm{coNP}$. Then, Σ_2^p (resp. Σ_3^p) corresponds to NP when an NP oracle (resp. a Σ_2^p oracle) for solving problems is available at no cost. Σ_2^p- and Σ_3^p-complete problems are thus rather difficult to solve. Π_2^p and Π_3^p are the complementary classes for Σ_2^p and Σ_3^p, respectively.

In the following we will also consider the complexity class D^P, which consists of the problems that can be solved in polynomial-time given the answer to one NP problem and one coNP problem.

For further background on complexity, we refer the reader to [25,30].

5.1. Summary of results

The complexity results for problems (P1)–(P3) are summarized in Table 1. According to these results, checking whether an action sequence is a $\phi;\psi$-reverse plan for another action sequence (i.e., (P1)) is easier than finding a $\phi;\psi$-reverse plan for an action sequence (i.e., (P2) and (P3)). Finding a $\phi;\psi$-reverse plan, where ϕ and ψ are given is harder than finding a $\phi;\psi$-reverse plan for arbitrary ϕ and ψ for $|R| = 2$, but is of the same complexity in all other cases. These problems get more difficult when the length of R increases: Problems (P1) and (P3) get more difficult when $|R| \geqslant 2$, while problem (P2) gets more difficult when $|R| > 2$.

Intuitively, the Σ_3^p-completeness of (P2) and (P3) is due to the following intermingled sources of complexity: (i) the exponential number of action sequences R of a polynomially-bounded length and in case of (P2), an exponential number of formulas ϕ and ψ which need to be considered, (ii) the test that for all states S and S' such that $\phi(S')$ and $\psi(S)$ hold and S' is reached from S after execution of AS, every execution of R which starts in S' ends in S, and (iii) the test that each partial execution of R starting in some state S' as in (ii) can be continued with the next action, i.e., the execution is not "stuck". In the case where $|R| = 1$, source (iii) vanishes, and similarly in the case where $|R| = 2$ for (P2). This will be considered more in detail below.

Table 1
Complexities of the decision problems (P1)–(P3), in terms of completeness

Problem	$\lvert R\rvert = 1$	$\lvert R\rvert = 2$	$\lvert R\rvert > 2$
(P1)	coNP	Π_2^p	Π_2^p
(P2)	Σ_2^p	Σ_2^p	Σ_3^p
(P3)	Σ_2^p	Σ_3^p	Σ_3^p

In problems (P1) and (P3), we do not check that the formulas ϕ and ψ are actually satisfied at some states S' and S, respectively, such that S' is reached from S be execution of AS (if no such states exist, the problem is trivially solved). Checking this condition changes the complexity of (P1) when $|R| = 1$ from coNP to D^P; it does not change the complexity of (P3).

The complexity of problems can be lower under some conditions. For example, if the reverse plan is short (i.e., has a length smaller than some constant) and contains no parallel actions, and ϕ and ψ are formulas from a polynomial size set of formulas, then only a polynomial number of candidates for ϕ; ψ-reverse plans need to be checked for (P3). If the executability of actions can be determined in polynomial time then (P1) gets coNP-complete, and (P2) and (P3) get Σ_2^p-complete.

Also tractability can be gained in certain cases. For example, if ϕ and ψ are conjunctions of literals which have a single model and the description of transitions $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ is such that for given fluent values S (resp., S') and action occurrences A all fluent values S' (resp., S) such that $tr(S, A, S')$ holds can be determined in polynomial time, then finding a short ϕ; ψ-reverse plan without parallel actions for a short action sequence is feasible in polynomial time. Thus in particular, reversal of the current state in an action sequence execution is possible in polynomial time under these conditions.

Note that the described condition on $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ implies that there are only polynomially many possible next states that result when taking an action, and that a state can be reached by taking a particular action only from polynomially many predecessor states. Such a condition is meaningful if actions have "bounded" effects in the sense that they only change a few fluents in the state, which may be frequently the case.

We leave a detailed study of lower complexity fragments and tractability issues for future work.

5.2. Detailed discussion of results

In the following, we present a detailed discussion of the results in Table 1. For that, we need the following notation: $D(\mathcal{A}, \mathcal{F})$ denotes an action domain description, specified by a set $\mathcal{F}$ of fluents, a set $\mathcal{A}$ of actions, and formulas $state(\mathcal{F})$ and $act(\mathcal{F}, \mathcal{A}, \mathcal{F}')$. Recall that $\phi(\mathcal{X})$ and $\psi(\mathcal{X})$ denote propositional combinations of symbols in $\mathcal{X}$.

Let us study first the complexity of the recognition problem (P1). We start with the simpler problem of recognizing a ϕ; ψ-reverse action for an action:

Theorem 3. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, two actions A and A' in $2^{\mathcal{A}}$, formulas $\phi(\mathcal{F})$, and $\psi(\mathcal{F})$, deciding whether A' is a ϕ; ψ-reverse action for A is* coNP*-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $A' = A$.*

The computational complexity does not increase for the problem of recognizing a ϕ; ψ-reverse plan of length 1 for an action sequence:

Theorem 4. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, an action A in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, deciding whether $\langle A \rangle$ is a ϕ; ψ-reverse plan for AS is* coNP*-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $|AS| = 1$.*

For the problem of recognizing a ϕ; ψ-reverse plan R (of length greater than 1) for an action sequence AS, the complexity increases by one level in the polynomial hierarchy:

Theorem 5. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, two action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, deciding whether R is a ϕ; ψ-reverse plan for AS is Π_2^p-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $|R| \geqslant 2$ is fixed.*

Intuitively, this increase can be explained as follows. Consider any two states S and S' such that executing AS at state S leads to state S'. Executing R at state S' may lead to an intermediate state S'' at which the rest of R is not executable, i.e., no successor state of S'' can be reached by the next action in R. Deciding whether an action is not executable at some state amounts to solving a coNP-complete problem.

Now let us study the complexity of the decision problem (P2). We will say that a ϕ; ψ-reverse plan R for an action sequence AS is *effective* if $\phi(\mathcal{F})$ is satisfied at some state reachable via AS from some arbitrary state which satisfies $\psi(\mathcal{F})$.

We start with a simpler version of this problem where both AS and R are of length 1.

Theorem 6. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action A in $2^{\mathcal{A}}$, deciding whether there exist formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$, and an action A' in $2^{\mathcal{A}}$ such that A' is an effective ϕ; ψ-reverse action for A is Σ_2^p-complete.*

Surprisingly, for sequences of actions, finding formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$, and some ϕ; ψ-reverse plan of length smaller than or equal to 2 is not more difficult:

Theorem 7. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ and a sequence R of at most two actions in $2^{\mathcal{A}}$ such that R is an effective ϕ; ψ-reverse plan for AS is Σ_2^p-complete. The Σ_2^p-hardness holds even if $|AS| = 1$ and $\psi(\mathcal{F})$ is fixed to $\top$.*

Intuitively, the reason is that in general formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ can be considered which limit the applicability of the reversal plan R to a single state. Since the plan $R = \langle A_1, A_2 \rangle$ has length two, the NP-hard test whether at an intermediate state S'' that was reached from S' after executing A_1, the action A_2 is executable, i.e., some successor state S''' of S'' can be reached by A_2, can be eliminated: the target state of the reversal, S (from which S' was reached by taking A), must be reached by A_2.

For the general problem of deciding whether there exist formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$, and a ϕ; ψ-reverse plan R (of polynomial length) for an action sequence AS, the complexity increases by one level in the polynomial hierarchy:

Theorem 8. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ and a sequence R of polynomial number of actions in $2^{\mathcal{A}}$ such that R is an effective ϕ; ψ-reverse plan for AS is Σ_3^p-complete. The Σ_3^p-hardness holds even if $|AS| = 1$ and $|R|$ is fixed to 3.*

Finally, let us study the computational complexity of problem (P3), i.e., finding a ϕ; ψ-reverse plan R for an action sequences AS where AS, ϕ, and ψ are given. For R containing a single action A, the problem is of the same difficulty as (P2).

Theorem 9. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, deciding whether there exists a ϕ; ψ-reverse plan R for AS such that $|R| = 1$ is Σ_2^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $|AS| = 1$.*

However, already for reverse plans of length at least two, the problem is harder and at the next level of the polynomial hierarchy.

Theorem 10. *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, deciding whether there exists a ϕ; ψ-reverse plan R for AS is Σ_3^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $|R|$ is fixed to 2.*

6. Computation of reverse plans

We compute reverse plans in the spirit of knowledge compilation [7]: first we compute offline reverse plans for some action sequences, and then use this information online to construct a concrete reverse plan for a given action sequence. In the offline phase, the computed reverse plans for action sequences are collected in a library. This library may not contain all possible reverse plans for all action sequences (since exponentially many of them exist), but a polynomial number of reverse plans for short action sequences (typically, of a few steps, and the reverse plans are short themselves). From these short reverse plans, one might efficiently compose online reverse plans for longer action sequences. For example, a reverse plan $\langle B_2, B_1 \rangle$ for the action sequence $\langle A_1, A_2 \rangle$ can be composed of the two reverse

actions B_1 and B_2 that undo the effects of two actions A_1 and A_2, respectively. As we show in Section 7, such a construction of a reverse plan for an action sequence, from the reverse plan library, can be done efficiently.

We define a reverse plan library in terms of its constituents as follows.

Definition 6.1. A *reverse plan item* (*RPI*) is a tuple of the form (AS, R, ϕ, ψ) such that R is a ϕ; ψ-reverse plan for the (nonempty) action sequence AS, where $\phi = \phi(\mathcal{F})$ and $\psi = \psi(\mathcal{F})$. An RPI is *single-step*, if $|AS| = 1$, i.e., AS consists of a single action, and *unconditional*, if $\phi = \psi = true$.

Definition 6.2. A *reverse plan library* L is a (finite) set of RPIs; it is called *single-step* (resp., *unconditional*), if each RPI in it is single-step (resp., unconditional).

The compilation of a reverse plan library also seems to be useful when, in the context of execution monitoring, for some action domains, reverse plans for the same action sequences need to be computed repeatedly. Thus, storing reverse plans that are frequently needed in the library speeds up the recovery. A further aspect is criticality of situations, in which quick recovery is necessary (e.g., leaving from a dangerous state). We do not further explore the issue of which reverse plan items should eventually be stored in the library here, and focus on basic methods to compute RPIs.

As we have pointed out in Section 5.1, under certain conditions reversal of an action sequence from the current state is feasible in polynomial time. Nonetheless, the precompilation approach still makes sense in this case, since by keeping some of those reverse plans (e.g., the more frequently used ones) in the reverse plan library, online reversal can be speeded up on average considerably and facilitated within linear time (cf. Section 7).

There are various ways to compute RPIs to fill a reverse plan library. Thanks to the logical framework and definitions of reverse actions and plans, it is fairly straightforward to encode the problem of actually finding an RPI (AS, R, ϕ, ψ) by solving quantified Boolean formulas (QBFs), which has been proposed as a problem solving method in the planning domain earlier, e.g., [31].

Another possibility is to reduce the problem to solving conformant planning problems defined relative to a modification of D. This approach is somehow more general than working with QBFs, since any conformant planner for the action representation framework can be used, of which some might be based on QBFs.[9] While action reversal is similar in spirit to conformant planning (in both problems, a sequence of actions must work out under all circumstances), there is a subtle difference which makes expressing reverse plans in conformant planning not straightforward, independent of the underlying action representation framework. Indeed, while in conformant planning, the states which we want to obtain are clearly identified by a goal condition, in action reversal such a condition is lacking; in fact, we might know nothing about the (single) state to which we want reverse. This must be overcome in a reduction, and requires for $\phi;\psi$-reversal a thoughtful construction.

6.1. Basic transformation to conformant planning

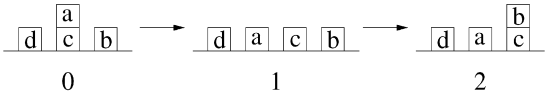
A *conformant plan* (cf. [38]) of length n for a planning problem $\mathcal{P}$ is a plan $P = \langle A_0, \ldots, A_{n-1} \rangle$ of length n for $\mathcal{P}$ such that the following formula holds:

$$\begin{aligned} &\forall \mathcal{F}_0, \ldots, \mathcal{F}_n \exists \mathcal{F}'_0, \ldots, \mathcal{F}'_n \; init(\mathcal{F}'_0) \\ &\quad \wedge \bigwedge_{t=0}^{n-1} \big(init(\mathcal{F}_0) \wedge tr_t(\mathcal{F}, A) \supset tr(\mathcal{F}_t, A_t, \mathcal{F}'_{t+1}) \big) \\ &\quad \wedge \big(init(\mathcal{F}_0) \wedge tr_n(\mathcal{F}, A) \supset goal(\mathcal{F}_n) \big). \end{aligned} \tag{11}$$

It expresses the following conditions on the plan P. The first conjunct expresses that there is at least one possible initial state. The second conjunct describes the executability of the plan. The last conjunct ensures that execution of the plan at an initial state leads to a goal state.

[9] Also reductions to conformant planners for other frameworks might be considered, but reasonably only if transforming the action description into the language of the conformant planner is not too expensive.

Example 6.1. Consider the blocks world domain, and the planning problem $\mathcal{P}$ described in Example 2.3. Plan $P = \langle throw(a, c, d), throw(b, table, c), throw(a, d, b)\rangle$ is not a conformant plan for $\mathcal{P}$ because it does not always lead to the goal state when executed at the initial state. For instance, the execution of the first two actions $throw(a, c, d), throw(b, table, c)$ may evolve like



where $throw(a, d, b)$ is not executable. As a matter of fact, no conformant plan made of *throw* actions exists for $\mathcal{P}$.

Now assume that another action is available, by which the agent can carry a block B safely to a location L (denoted by $carry(B, L)$), such that B will definitely reside on L after the execution of this action. With this new action, the definition of $act(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ is modified as follows. The conjunction (4) with the formula below describes the preconditions of actions:

$$carry(B, L) \supset \bigwedge_{B1} \neg on(B1, B), \tag{12}$$

and the conjunction of the following with (6) describes no-concurrency:

$$\left(carry(B, L) \supset \left(\bigwedge_{B1,L2,L3} \neg throw(B1, L2, L3) \ \wedge \bigwedge_{B1\neq B,L2} \neg carry(B1, L2)\right)\right). \tag{13}$$

The effects of actions, and inertia are defined, instead of (5), with the conjunction:

$$\big(carry(B, L) \supset on(B, L)'\big) \ \wedge \left(on(B, L1)' \supset \left(on(B, L1) \vee carry(B, L1) \vee \bigvee_{L,L2} throw(B, L, L2)\right)\right). \tag{14}$$

In the blocks world above, a conformant plan for the planning problem $\mathcal{P}$ is

$$P' = \big\langle carry(a, d), carry(b, c), carry(a, b)\big\rangle.$$

Indeed, plan P' is executable at the initial state, and, when executed, it leads to the goal state.

Let us now discuss how some unconditional RPIs for a given action sequence AS can be obtained by a simple reduction to conformant planning. To this end, we define a modified domain D_{rev} and planning problem P_{rev} as follows. We consider the fluents in $\mathcal{F}_{rev} = \mathcal{F} \cup \tilde{\mathcal{F}}$, where $\tilde{\mathcal{F}} = \{\tilde{f} \mid f \in \mathcal{F}\}$ consists of new fluent symbols. For D_{rev}, the states are defined by the interpretations of $\mathcal{F}_{rev}$ such that $state(\mathcal{F})$ holds. The action symbols are $\mathcal{A}_{rev} = \mathcal{A}$.

The transition function is defined by the formula $tr_{rev}(\mathcal{F}_{rev}\mathcal{A}_{rev}\mathcal{F}'_{rev})$ where fluent values in $\tilde{\mathcal{F}}$ are copied to $\tilde{\mathcal{F}}'$:

$$tr_{rev}(\mathcal{F}_{rev}\mathcal{A}_{rev}\mathcal{F}'_{rev}) = \big(tr(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge \tilde{\mathcal{F}} \equiv \tilde{\mathcal{F}}'\big). \tag{15}$$

For P_{rev}, the initial state is defined by the formula $init_{rev}(\mathcal{F}_{rev})$, which encodes all possible states over $\mathcal{F}$, and which additionally duplicates all fluents of $\mathcal{F}$ to $\tilde{\mathcal{F}}$:

$$init_{rev}(\mathcal{F}_{rev}) = \big(state(\mathcal{F}) \wedge \mathcal{F} \equiv \tilde{\mathcal{F}}\big). \tag{16}$$

The goal conditions are defined by the formula $goal_{rev}(\mathcal{F}_{rev})$ which makes sure that the fluent values for $\mathcal{F}$ (which have been changed by actions according to $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$) are equal to those in $\tilde{\mathcal{F}}$ (which, in turn, are equal to the initial state fluent values for $\mathcal{F}$):

$$goal_{rev}(\mathcal{F}_{rev}) = (\mathcal{F} \equiv \tilde{\mathcal{F}}). \tag{17}$$

Any conformant plan for P_{rev} of length $\geqslant 1$ represents reverse plans:

Theorem 11. *Let P_{rev} be the planning problem defined relative to the action description D_{rev}, and let $\langle A_0, \ldots, A_{n-1}\rangle$, $n > 0$, be a conformant plan for P_{rev}. Then for any $i \in \{1, \ldots, n\}$, $\langle A_i, \ldots, A_{n-1}\rangle$ is a reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D.*

Note that even if each conformant plan for P_{rev} gives rise to unconditional RPIs, many unconditional RPIs cannot be found in this manner. The reason is that each action in a conformant plan must be executable in any state reachable by executing the preceding actions. While this is a necessary condition for the "reverse" part of the plan, the "to-be-reversed" part will often not satisfy this condition.

In the following we will present an elaboration of this basic method, which allows for the computation of *all* $\phi;\psi$-reverse plans for a given action sequence and given formulas ϕ and ψ.

6.2. Advanced transformation to conformant planning

As noted above, using the transformation presented in the previous section it is only possible to find unconditional RPIs which satisfy a strict condition, basically the action sequence to be reversed should be a conformant plan (with a tautological goal). More precisely, for an action sequence $AS = \langle A_0, \ldots, A_{i-1}\rangle$ to be reversed, each action A_j must be executable in any state S_j which is reached by taking $A_0, \ldots, A_{j-1}$ starting at an arbitrary initial state, i.e., the execution must not get stuck at S_j under any circumstances.

This limitation can be overcome by a slight modification of the transformation, in which branching to a "don't care" state is possible at any intermediate state S_j. Being in such a "don't care" state, it is possible to execute whatever actions, arriving again in a "don't care" state. In this way, an action sequence AS will be executable under any circumstances. We will add a designated fluent symbol *normal*, which holds unless a "don't care" state has been entered.

However, for the reverse part of the plan, it should not be possible to make a transition to a "don't care" state. Indeed, if a state is reached by executing a part of a reverse plan, from which the reverse plan cannot be continued, the reverse plan is not usable, as it should, by definition, work under any circumstances.

To do this, we must be able to differentiate between the "forward" and the "reverse" phase. For the basic encoding, this has been done a posteriori, and has not been explicit in the encoding. Therefore we introduce a new action symbol *sep*, the purpose of which is indicating this transition. Furthermore, we introduce a new fluent symbol *sep_occ*, which indicates whether we are in the "forward" phase, in which going to a "don't care" state is allowed, or in the "reverse" phase, where going to "don't care" states is forbidden (unless one is already in such a state, coming from the "forward" phase).

These new symbols also provide a handle for testing a given condition ϕ on the states from which a reversal should be found: It suffices to assert ϕ in the state after which *sep* occurs. If ϕ does not hold in such a state, the reverse plan R for AS need not work. Since a conformant plan requires executability in any state, we again change to a "don't care" state in such a situation. Furthermore, we can handle a condition ψ on the state to which we want reverse by checking ψ on the initial state of the conformant planning problem. With this enhancement, we are in the position to compute also conditional RPIs (for given formulas ϕ and ψ, as in (P3)).

We refer to this modification of P_{rev} as P^c_{rev}, and describe it now in more detail. The new sets of fluent and action symbols are denoted by $\mathcal{F}^c_{rev}$ and $\mathcal{A}^c_{rev}$, respectively. The initial state formula is as in P_{rev}, augmented by the information that a "don't care" state has not been entered (*normal* holds) and that we are in the "forward" phase (*sep_occ* does not hold):

$$init^c_{rev}(\mathcal{F}^c_{rev}) = init_{rev}(\mathcal{F}_{rev}) \wedge normal \wedge \neg sep\_occ.$$

The transition diagram is described by the following formula:

$$tr^c_{rev}(\mathcal{F}^c_{rev}, \mathcal{A}^c_{rev}, \mathcal{F}^c{}'_{rev}) = \tag{18}$$

$$normal \supset \Big($$

$$\big((\neg sep\_occ \wedge \neg normal') \vee \tag{19}$$

$$\big(\neg sep \supset \big(normal' \wedge tr_{rev}\big(\mathcal{F}_{rev}, \mathcal{A}_{rev}, \mathcal{F}_{rev}{}'\big)\big)\big)\big) \wedge \tag{20}$$

$$\big(sep \supset \big(\big(\phi(\mathcal{F}) \wedge \psi(\tilde{\mathcal{F}})\big) \equiv normal'\big)\big) \wedge \tag{21}$$

$$\Big(sep \supset \bigwedge_{a\in\mathcal{A}} \neg a\Big) \wedge \tag{22}$$

$$\big(sep \supset (\mathcal{F}_{rev} \equiv \mathcal{F}'_{rev})\big) \wedge \tag{23}$$

$$\big(sep \supset (\neg sep\_occ \wedge sep\_occ')\big) \wedge \tag{24}$$

$$\Big(\big(sep\_occ' \equiv (sep\_occ \vee sep)\big)\Big) \wedge \tag{25}$$

$$(\neg normal \supset \neg normal'). \tag{26}$$

In this formula, there is a fundamental case distinction between "normal" and "don't care" mode. If the fluent *normal* holds, the transition may either continue "normally" or change into "don't care" (the latter only in the "forward" phase). If *normal* does not hold, we are in "don't care" mode, in which the transition can continue only in "don't care" mode, i.e., *normal* continues to be false, by virtue of (26).

In "normal" mode, we can either switch to "don't care" mode, provided that we are in the "forward" phase by (19), or continue in "normal" mode according to $tr_{rev}(\mathcal{F}_{rev}\mathcal{A}_{rev}\mathcal{F}'_{rev})$, provided that we are not in the "separating" transition (*sep* does not hold, (20)).

If we are in the "separating" transition (*sep* holds), we first test whether $\phi(\mathcal{F}) \wedge \psi(\tilde{\mathcal{F}})$ holds. If it does, we continue in "normal" mode, if it does not, we change to "don't care" mode, by means of (21). Furthermore, we assess that no other action occurs together with *sep* by (22). Note that this is only included for clarity, enforcing this is not strictly needed.

For the "separating" transition, we also have to copy the truth value of all fluent values in $\mathcal{F}_{rev}$ by (23). Note that copying the truth value of *normal* is not needed. If it was false, it remains false by (26), if it was true, it is determined by the validity of ϕ and ψ by (21). We then assert that *sep_occ* did not hold before the "separating" transition, and that it holds after it by (24). Finally, by (25) we make sure that *sep_occ* holds only after *sep* occurred or if it was true before.

We are now ready to formulate the goal. In addition to $goal_{rev}(\mathcal{F}_{rev})$ we require *sep_occ* to hold, and we allow the "don't care" mode:

$$goal^c_{rev}(\mathcal{F}^c_{rev}) = \big(goal_{rev}(\mathcal{F}_{rev}) \wedge sep\_occ\big) \vee \neg normal.$$

This goal is eventually established in a "normal" evolution if the separation action occurred and the goal of the planning problem P_{rev} from above, $\mathcal{F} \equiv \tilde{\mathcal{F}}$, is established; that is, the goal state is identical to the initial state on the original fluents $\mathcal{F}$. This means that from any state satisfying ϕ, which is reached by executing AS (the actions of the "forward" phase), the initial state can be reestablished by executing R (the actions of the "reverse" phase) in each possible evolution, provided that the initial state satisfies ψ. R is therefore a $\phi;\psi$-reverse plan for AS. Moreover, we can also show that any $\phi;\psi$-reverse plan for a given AS can be found in this way (for given ϕ and ψ).

More formally, the following statement holds.

Theorem 12. *Let P^c_{rev} be the planning problem defined relative to the action description D^c_{rev}. Then, $\langle A_i, \ldots, A_{n-1}\rangle$ is a $\phi;\psi$-reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geqslant i > 0$, is a conformant plan for P^c_{rev}.*

Therefore, we can generate all $\phi;\psi$-reverse plans by solving the conformant planning problem P^c_{rev} (imposing suitable bounds on the plan length). We remark that like finding $\phi;\psi$-reverse plans, conformant planning is Σ^p_3-complete [15,38] in general and Σ^p_2-complete if action executability is testable in polynomial time [15]. The above transformation complies with this because it carries polynomial-time action executability in D over to D^c_{rev}.

We next consider an example of the transformation.

Example 6.2. Consider the setting of Example 6.1, and suppose we want to find reverse actions for other actions (i.e., unconditional single-step RPIs (AS, R, ϕ, ψ) where $|R| = 1$). The associated planning problem P^c_{rev} where $\phi = \top$ and $\psi = \top$ admits the following conformant plans of length 3:

$$\big\langle \big\{throw(X, Y, Z)\big\}, \{sep\}, \big\{carry(X, Y)\big\}\big\rangle,$$

where $X \in \{a, b, c, d\}$, $Y, Z \in \{table, a, b, c, d\}$, and $Y \neq Z$. These plans give rise to a single-step, unconditional plan library.

Example 6.3. Consider the setting of Example 3.4, and suppose we want to find conditional reverse actions for other actions (i.e., single-step RPIs (AS, R, ϕ, ψ) where $|R| = 1$). A suitable conditional plan library consists of entries

$$\big(\langle throw1(X, Y)\rangle, \langle carry(X, Z)\rangle, true, on(B, Z)\big)$$

where X is a block and Y, Z are locations.

As described in Example 3.4, no unconditional reverse plans exist in this setting, so these items can only be applied if some percept is available which states that the block was on the location to which it will be carried as reversal.

In order to single out the effective $\phi; \psi$-reverse plans among them, an additional reachability test may be applied by checking the satisfiability of the formula $tr_{i-1}(\mathcal{F}, A)$ for the action sequence $\langle A_0, \ldots, A_{i-1}\rangle$ obtained from the conformant plan, or this condition may be encoded in P^c_{rev} using (quite a number of) additional fluents.

An alternative way would be following a generate and test approach as implemented by conformant planners in the area of action languages like CPLAN [16,20] and DLV$^{\mathcal{K}}$ [14], to (1) generate arbitrary (not necessarily conformant) plans P in P^c_{rev} with executions in normal mode (which correspond to circular trajectories in the original domain D), and then (2) to check for each P whether it is conformant. For step (1), we simply need to fix *normal* and $normal'$ in P^c_{rev} to *true* (and may simplify the resulting formulas). Let the resulting planning problem be ${P^c_{rev}}'$. More formally, we have the following result.

Theorem 13. *Let P^c_{rev} and ${P^c_{rev}}'$ be the planning problems defined relative to the action description D^c_{rev}. Then, $\langle A_i, \ldots, A_{n-1}\rangle$ is an effective $\phi; \psi$-reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geqslant i > 0$, is a plan for ${P^c_{rev}}'$ which is a conformant plan for P^c_{rev}.*

6.3. Offline reverse plan library construction

Using the results above, a reverse plan library for a collection $\phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_m$ of conditions can be built. Suggestive examples are conjunctions of literals ("terms") or clauses, of bounded size. Note that, due to the definition of a conditional reversal, the following conditions hold:

(*) For a given action domain $D(\mathcal{A}, \mathcal{F})$, formula $\phi(\mathcal{F})$ of the form $\phi_1(\mathcal{F}) \vee \phi_2(\mathcal{F})$, formula $\psi(\mathcal{F})$, and action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, it holds that R is a $\phi; \psi$-reverse plan for AS iff R is a $\phi_i; \psi$-reverse plan for AS, for $i \in \{1, 2\}$.

(**) For a given action domain $D(\mathcal{A}, \mathcal{F})$, formula $\phi(\mathcal{F})$, formula $\psi(\mathcal{F})$ of the form $\psi_1(\mathcal{F}) \vee \psi_2(\mathcal{F})$, and action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, it holds that R is a $\phi; \psi$-reverse plan for AS iff R is a $\phi; \psi_i$-reverse plan for AS, for $i \in \{1, 2\}$.

Then k-DNF reverse conditions for ϕ and ψ can be synthesized from terms of size at most k (in particular, clauses from literals), and any RPIs $(AS, R, \phi_1, \psi), \ldots, (AS, R, \phi_k, \psi)$ can be merged into a single RPI $(AS, R, \phi_1 \vee \cdots \vee \phi_k, \psi)$, and similarly RPIs $(AS, R, \phi, \psi_1), \ldots, (AS, R, \phi, \psi_k)$ can be merged into $(AS, R, \phi, \psi_1 \vee \cdots \vee \psi_k)$. More general merging rules can be applied to merge RPIs $(AS, R, \phi_1, \psi_1), \ldots, (AS, R, \phi_k, \psi_k)$ into a single RPI, (AS, R, ϕ, ψ) where $\phi = \phi_1 \vee \cdots \vee \phi_k$ and $\psi = \psi_1 \vee \cdots \vee \psi_k$, provided that additional conditions hold (e.g., that the trajectories of AS starting in S and ending in S' such that ψ holds at S and ϕ holds at S', are those starting in S and ending in S' such that $\psi_i(S)$ and $\phi_i(S')$ holds, for some $i \in \{1, \ldots, k\}$). We leave the construction of compact reverse plan libraries for further study.

7. Online reverse plan assembly

At runtime, when we do try to assemble a reverse plan, we can think of three increasingly expressive scenarios, depending on available state information in form of *percepts* π_j about some states S_j, $j = 0, 1, \ldots, i$, of the execution:

(1) There is no information about the current state, S_i and past states $S_0, S_1, \ldots, S_{i-1}$ whatsoever. In this situation, only unconditional reversal plans, assembled from unconditional RPIs, might be used.

(2) (Partial) information about the current state S_i is available, expressed by a formula $\pi_i(\mathcal{F})$ such that S_i is one of its models, but no information about the past states. In this case, we can also make use of conditional RPIs.
(3) (Partial) information about the whole execution history is available, formalized in terms of a sequence $\Pi = \pi_0, \ldots, \pi_i$ of formulas over fluent symbols, such that the state S_j is a model of $\pi_j(\mathcal{F})$, for each $j = 0, 1, \ldots, i$. Here, we might exploit an even larger set of RPIs.

Clearly scenario 3 generalizes the other ones, and scenario 2 in turn generalizes scenario 1; if no information is available about state S_j, then we set $\pi_j(\mathcal{F}) = \mathit{true}$. In the following subsections, we shall first concentrate on the third scenario, and then address briefly the second scenario.

Often, especially in the case of sensory input, $\pi_j(\mathcal{F})$ may be a conjunction of literals, representing the values of those fluents which are sensed. Thus, Π might just consist of a sequence $\pi_0, \pi_1, \ldots, \pi_i$ of such sensory inputs. Missing or noisy sensory input on S_j can be accommodated by setting $\pi_j = \mathit{true}$ (i.e., the empty conjunction); thus, the framework offers some robustness in this respect.

Note that these percepts are not necessarily results of sensing actions. They may also have been obtained by the monitoring framework, for example by checkpointing. If the initial state has been fully known, also this information can serve as a percept. If the percepts have been obtained through sensing actions, the plan that is being executed can be viewed as a branch of an incomplete conditional plan, which does not guarantee a goal state. The conditional plan may be incomplete, e.g., because the sensory input is not sufficient to take into account all contingencies, or, only a part of the conditional plan with less than some given number of steps is considered.

7.1. Using single-step reverse plan libraries

We first consider reverse plan libraries L which contain only single-step RPIs. In this case, a reverse plan for a given action sequence $AS = A_0, \ldots, A_{i-1}$ (occurring in a plan) from the reached state, S_i, can be constructed by the algorithm presented in Fig. 6. Here, the symbol $+$ denotes concatenation.

The following theorem states a bound on the running time of the algorithm, which is polynomial in many settings, and that it works properly. For that, we first make the notion of a reverse plan constructed from a reverse plan library for a particular state precise.

Definition 7.1. An action sequence R is a $\phi;\psi$*-reverse plan for an action sequence* AS *and a sequence* $\Pi = \pi_0, \ldots, \pi_{|AS|}$ *of percepts relative to a reverse plan library* L, if one of the following holds:

- there is an RPI (AS, R, ϕ, ψ) in L such that $\psi_{|AS|} \supset \phi$ and $\psi_0 \supset \psi$ holds;
- for some action sequences AS' and R', there is an RPI (AS', R', ϕ, ψ') in L such that $\pi_{|AS|} \supset \phi$ and $\pi_{|AS|-|AS'|} \supset \psi'$ holds and the following condition is satisfied: for the action sequences AS'' and R'' such that $AS = AS'' + AS'$ and $R = R' + R''$, R'' is for some ϕ'' a ϕ''; ψ-reverse plan for AS'' and $\Pi'' = \pi_0, \ldots, \pi_{|AS''|}$ relative to L.

Algorithm S-REVERSE(AS, Π, L)

Input: Action sequence $AS = \langle A_0, \ldots, A_{i-1}\rangle$, $i \geqslant 0$, sequence of formulas (percepts) $\Pi = \pi_0(\mathcal{F}), \ldots, \pi_i(\mathcal{F})$, single-step reverse plan library L;
Output: Reverse plan RP for AS from Π and L or "*no*" if none exists

```
(01)  RP := ϵ ; /* empty plan */
(02)  for each  j = i−1, i−2, ..., 0 do
(03)    if  some (⟨A⟩, R, φ, ψ) ∈ L exists such
(04)      that A=A_j, π_{j+1} ⊃ φ, and π_j ⊃ ψ then
(05)    begin
(06)      RP := RP + R;
(07)    end
(08)    else return "no";
(09)  return (RP)
```

Fig. 6. Algorithm S-REVERSE to compute execution reversals using a single-step plan library.

Theorem 14. (i) S-REVERSE(AS, Π, L) *has running time of order* $O(|AS| \cdot |L| \cdot eval_{\max}(\Pi, L))$, *where* $eval_{\max}(\Pi, L)$ *bounds the time to evaluate* $\pi_j \supset \phi$ *and* $\pi_j \supset \psi$ *for any* π_j *in* Π *and formulas* ϕ, ψ *in* L.[10]

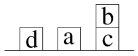
(ii) S-REVERSE(AS, Π, L) *correctly outputs, relative to* L, *a reverse plan* RP *for* AS *from* Π, *or it determines that such a reverse plan does not exist.*

An immediate corollary of this theorem is the following result.

Corollary 15. S-REVERSE(AS, Π, L) *is polynomial, if all percepts in* Π *are DNFs and all formulas in* L *are* k*-term DNFs, i.e.,* $\bigvee_{j=1}^{k} t_{i,j}$ *where* k *is bounded by a constant, or CNFs.*

In particular, this includes the case where the percepts and the formulas in L are conjunctions of literals (i.e., $k = 1$). Other relevant polynomial cases, e.g., for small formulas in Π or in L are small, can be easily found.

Example 7.1. Consider the setting of Example 6.1. Take L to be the derived single-step unconditional plan library of Example 6.2. Consider the plan P of Example 2.3, and the following percept S_2 at time stamp 2 (as in Example 4.2).



In addition, since there was a single initial state S_0, also this can serve as a percept at time stamp 0. In this situation, in order to produce a reverse plan for $\langle throw(a, c, d), throw(b, table, c)\rangle$, we call

$$\text{S-REVERSE}\big(\langle throw(a, c, d), throw(b, table, c)\rangle, (S_0, \top, S_2), L\big).$$

According to S-REVERSE, after initialization, for j=1, the only match is

$$\big(\langle throw(b, table, c)\rangle, \langle carry(b, table)\rangle, true, true\big).$$

Therefore, $RP := \langle carry(b, table)\rangle$. For $j = 0$, the only match in L is

$$\big(\langle throw(a, c, d)\rangle, \langle carry(a, c)\rangle, true, true\big);$$

hence, $RP := \langle carry(b, table), carry(a, c)\rangle$ is finally returned.

Example 7.2. Now consider a similar situation in the setting of Example 3.4, taking L to be the conditional plan library of Example 6.3. Here, we want to produce a reverse plan for the action sequence $\langle throw1(a, d), throw1(b, c)\rangle$. As in Example 6.1 assume that the percept π_2 is S_2, and the percept π_0 is S_0. Calling

$$\text{S-REVERSE}\big(\langle throw1(a, d), throw1(b, c)\rangle, (S_0, \top, S_2), L\big)$$

will not produce a reverse plan: for $j = 1$, the only available RPIs for $throw1(b, c)$ have a ψ condition, which is not $\top$. Since $\pi_1 = \top$ (as we have no information about time stamp 1), $\pi_1 \supset \psi$ cannot hold.

However, if we have a percept $\pi_1 = on(b, table) \wedge on(d, table)$ for time stamp 1, and call

$$\text{S-REVERSE}\big(\langle throw1(a, d), throw1(b, c)\rangle, (S_0, on(b, table) \wedge on(d, table), S_2), L\big)$$

there is one match for $j = 1$ in L:

$$\big(\langle throw1(b, c)\rangle, \langle carry(b, table)\rangle, true, on(b, table)\big),$$

since $\pi_1 \supset on(b, table)$. Therefore, $RP := \langle carry(b, table)\rangle$. For $j = 0$, the only match in L is

$$\big(\langle throw1(a, d)\rangle, \langle carry(a, c)\rangle, true, on(a, c)\big);$$

since $\pi_0 \supset on(a, c)$. Hence, $RP := \langle carry(b, table), carry(a, c)\rangle$ is returned.

[10] In this bound, and similarly in Theorem 16, string copy operations to assemble RP are not explicitly accounted; RP should be not much larger than AS anyway, and strong copy can be circumvented by returning a list of pointers to the strings which constitute RP.

7.1.1. Exploiting full state information

Given that π_i in Π describes precisely the current state S_i (i.e., S_i is the only model of π_i), and that some function exec(S, R) for computing the state which results from executing the action sequence $R = \langle B_1, \ldots, B_k \rangle$ starting at state S is available, we can modify Algorithm S-REVERSE(AS, Π, L) such that it also outputs the state S_0 to which we want to revert. For that, in line (01) we add $S := S_i$, and between lines (05) and (06) the statement $S :=$ exec(S, R), and return in line (09) S besides RP.

Furthermore, we can replace "$\pi_{i+1} \supset \phi$" in line (04) with $\phi(S) = \top$, and also use instead of "$\pi_i \supset \psi$" there the test whether $\psi(\text{exec}(S, R)) = \top$. Both these modifications increase the number of RPIs that might be successfully applied in each step. Since testing $\phi(S) = \top$ and $\psi(S) = \top$ can be done in linear time for arbitrary formulas ϕ and ψ, the resulting algorithm is polynomial, given that exec(S, R) is computable in polynomial time; we refer to [11] for a detailed discussion.

7.2. Using multi-step reverse plan libraries

When we consider a multi-step plan library, i.e., not necessarily a single-step plan library, finding a reverse plan RP is trickier since RP may be assembled from L in many different ways, and state conditions might exclude some of them. For instance, take

$$AS = \langle A, B, C \rangle,$$

and

$$L = \big\{ \big(\langle A, B \rangle, \langle D \rangle, \phi_1, \top \big), \big(\langle C \rangle, \langle E \rangle, \phi_2, \top \big), \big(\langle A \rangle, \langle F \rangle, \phi_3, \top \big), \big(\langle B, C \rangle, \langle G \rangle, \phi_4, \top \big) \big\}.$$

We can assemble the action sequence $\langle A, B, C \rangle$ from $\langle A, B \rangle$ and $\langle C \rangle$, or from $\langle A \rangle$ and $\langle B, C \rangle$. However, in the former case, ϕ_1 might be false at the state resulting from reversing C by E, while, in the latter case, ϕ_3 might be true at the state resulting from reversing the action sequence $\langle B, C \rangle$ by the action G. Therefore, we need to consider choices and constraints when building a reverse plan.

Fortunately, this is not a source of intractability, and a reverse plan from L can be found in polynomial time (if one exists) by the algorithm REVERSE in Fig. 7, which generalizes algorithm S-REVERSE.

The auxiliary array S in the algorithms in Fig. 7 is used for keeping information to which states S_j a reversal is established. The main algorithm, REVERSE, initializes every $S[j]$ $(j < i)$ of S to $\bot$ since this is false initially. The recursive algorithm REVERSE1 updates S whenever new knowledge is gained. For instance, if the action A_{i-1} can be reversed at state S_i, then we know that a reversal to S_{i-1} exists and modify $S[i-1]$ accordingly. Having this information available in S helps us find a reverse plan for the action sequence AS from L. Also, it prevents us explore the same search space over and over.

The algorithm REVERSE starts constructing a reverse plan for an action sequence $A_0, \ldots, A_{j-1}$ by considering its suffixes As. For efficiently determining all As in L, we can employ search structures such as a trie (or indexed trie) to represent L: consider each node of the trie labeled by an action, so that the path from the root to the node would describe an action sequence in reverse order. If the node describes an action sequence As such that (As, R, ϕ, ψ) is in L, then the node is linked to a list of all RPIs of form (As, R', ϕ', ψ') in L.

The next theorem bounds the running time of algorithm REVERSE and states its correctness.

Theorem 16. (i) REVERSE(AS, Π, L) *has running time* $\mathrm{O}(|AS|(|L| \cdot eval_{\max}(\Pi, \mathcal{A}) + \min(AS_{\max}(L), |AS|)))$*, where* $eval_{\max}(\Pi, L)$ *bounds the time to evaluate* $\pi_j \supset \phi$ *and* $\pi_j \supset \psi$ *for any* π_j *in* Π *and formulas* ϕ*,* ψ *in* L*; and* $AS_{\max}(L) = \max\{|As| \mid (As, R, \phi, \psi) \in L\}$.

(ii) REVERSE(AS, Π, L) *correctly outputs, relative to* L*, a reverse plan* RP *for* AS *and* Π *or it determines that such a plan does not exist.*

Corollary 17. REVERSE(AS, Π, L) *is polynomial, if all percepts in* Π *are DNFs and all formulas in* L *are* k*-term DNFs, i.e.,* $\bigvee_{j=1}^{k} t_{i,j}$ *where* k *is bounded by a constant, or CNFs.*

We remark that in an application setting, $|AS|$ as well as reverse plan R are expected to be small (bounded by a constant) and percepts π_i and the formulas ϕ, ψ consist of a few literals. In this case, the running time is $\mathrm{O}(|L|)$ i.e.,

Algorithm REVERSE(AS, Π, L)

Input: Action sequence $AS = \langle A_0, \ldots, A_{i-1}\rangle$, $i \geqslant 0$, sequence of formulas (percepts) $\Pi = \pi_0(\mathcal{F}), \ldots, \pi_i(\mathcal{F})$, reverse plan library L;
Output: Reverse plan RP for AS from Π and L or "*no*" if none exists

(01) **f**or each $j = 0, \ldots, i-1$ **do** $S[j] := \bot$;
(02) $S[i] := \top$; /* trivially, S_i is reversible to itself */
(03) $RP :=$ REVERSE1(i);
(04) **if** $RP =$ "*no*" **then return** "*no*"
(05) **else return** (RP,$S[0]$)

Algorithm REVERSE1(j)

Input: Integer j, $0 \leqslant j \leqslant i$ $(= |AS|)$;
Output: Reverse plan RP for $\langle A_0, \ldots, A_{j-1}\rangle$ from $\pi_0, \ldots, \pi_j$, or "*no*" if none exists

(01) **if** $j = 0$ **then return** ϵ ; /* empty plan for void reversal */
(02) **for each** $(As, R, \phi, \psi) \in L$ s.t. As is a suffix of $\langle A_0, \ldots, A_{j-1}\rangle$ and $S[j - As] = \bot$ **do**
(03) **if** $\pi_j \supset \phi$ and $\pi_{j-|AS|} \supset \psi$ **then**
(04) **begin**
(05) $S[j-|As|] := \top$; /* reversing to state S_j is possible */
(06) $RP :=$ REVERSE1($j-|As|$);
(07) **if** $RP \neq$ "*no*" **then return** $R + RP$
(08) **end**
(09) **return** "*no*"

Fig. 7. Algorithm REVERSE to compute execution reversals using a multi-step plan library.

linear in the size of the plan library L. If, moreover, only few of the entries in the reverse plan library match, then the running time can be drastically shorter.

7.2.1. Exploiting full state information

As in the case of a single-step plan library, we can take advantage of precise knowledge of the state S_i (i.e., S_i is the only model of the percept π_i) from which we want to roll back, and of the function exec(S, R). We can easily modify Algorithm S-REVERSE(AS, Π, L) such that it also outputs the state S_0 to which we want to revert to as follows. In line (02) of REVERSE we store in S_i in $S[i]$, and similarly in line (05) of REVERSE1 exec($S[j]$, R) in $S[j - |AS|]$.

Furthermore, the condition "$\psi_j \supset \phi$" in line (03) of REVERSE1 can be replaced by "$\phi(S[j]) = \mathit{true}$", and the test "$\psi_{j-|AS|} \supset \psi$" by "$\psi(\text{exec}(S[j], R)) = \mathit{true}$". Again, both modifications increase the possibilities to apply RPIs in finding a reversal, and lead to a polynomial time algorithm for arbitrary formulas ϕ and ψ, given that exec($S[j]$, R) is computable in polynomial time. Furthermore, under practical constraints a linear time algorithm results (see [11]).

8. Related work

Our method of undoing the effects of the execution of an action sequence is different from the existing one [23,24] mainly in two ways. First, in [23,24], the user needs to specify in advance the reverse action for each action. In our method, the reverse actions for an action can be computed offline by solving a conformant planning problem, and this information is stored in a reverse plan library. Second, in [23,24], a sequence of actions is undone by reversing each action. In our method, a sequence of actions is undone by computing online a reverse plan using the given reverse plan library. In a reverse plan library, an action may have many reverse actions or reverse plans, rather than a single reverse action; and action sequences whose length is greater than 1 may have reverse plans as well. When some actions in the given action sequence do not have reverse actions but reverse plans of length greater than 1, our online construction method allows us to find a reversal for the given action sequence. In such cases, a reversal for the given action sequence can not be found by the method of [23,24]. When some parts of the given action sequence have reverse plans of length 1, our online construction method allows us to find a shorter reversal for the given action sequence than the one computed by the method of [23,24]. In this sense, our method is more general.

In this paper, we have described our method of undoing the effects of actions in the context of execution monitoring, for recovering from discrepancies. According to this method, the agent finds a reverse plan to backtrack to a diagnosed point of failure and continues with the execution of the original plan from that point on. This recovery method is different from the plan recovery approaches of the other logic-based monitoring frameworks [9,18,35,36] as follows.

In [9], backtracking is not considered; instead, a new plan is computed so that executing it followed by the remaining plan would lead to a goal situation from the current situation. In [35], the authors consider restartable plans so that the agent can backtrack to a past nondeterministic choice point without having to compute a plan. After identifying the latest nondeterministic choice point, the agent executes the plan from that point on, to reach a goal situation from the current situation. If the agent cannot reach a goal situation then she identifies the next past nondeterministic choice point, and follows the recovery procedure as above. In [36], backtracking is considered in connection with inserting corrective plans as in [9], by a recursive recovery procedure like the one in [35]. The agent computes a plan from the current situation to reach the latest nondeterministic choice point. If executing the plan from that point on does not lead to a goal situation then the agent tries to recover by inserting a corrective plan at that point. If the agent cannot find a corrective plan then she finds the next past nondeterministic choice point, and follows the recovery procedure as above. In [18], backtracking is not considered. If a diagnosis is found then some predefined plans are executed to achieve the goals; otherwise, a new plan is computed to reach to a goal situation from the current situation.

Our method of backtracking by reversals to recover from discrepancies is complementary to the other recovery techniques mentioned above, as discussed in Section 4.

The idea of backtracking for recovery in execution monitoring is similar to "reverse execution" in program debugging [3,41], where every action is undone to reach a "stable" state. Our method is more general because it does not require an execution history to be able to undo the effects of actions. Finally, undoing and redoing actions on the database is at the heart of recovery in database management systems. However, in this context, a log of the action history is available, and the step-by-step undo actions are well known; thus, the problem addressed is significantly different.

As noted already in Section 1, topics on or related to undoing or reversing actions have been studied in various areas of computer science. A fairly comprehensive historic survey has been given in [26]. Among the more recent areas in which undoing actions have an application or have been studied are database systems [39], workflow and business process management [2,40], and computer supported cooperative work [37].

A particularly interesting field, in which problems on reversing actions have been studied recently, is on the fringe between programming languages, theoretical computer science, and physics. The basic observation, which has already been made in the 1960s, is that only logically irreversible operations in a computer necessarily dissipate energy. It is therefore of interest to know whether an algorithm, or in general, a machine model, is logically reversible. As examples we refer to [1,28], in which the concept of *Geometry of Interaction* is used to transform irreversible formalisms into reversible ones and to describe an implementation paradigm for functional programming languages, respectively. Complexity issues other than some undecidability issues were not addressed in these papers.

The complexity of planning in different action languages and the framework considered here has been studied e.g. in [5,15,27,31,38]. In particular, conformant planning, which as discussed in Section 6 is related to action reversal, under different assumptions about the underlying domain has been studied in [5,15,27,31,38]. The papers [5,31] focused on deterministic actions (and multiple possible initial states), while [15,38] also considered nondeterministic actions.

Table 2 shows the complexities of conformant planning problems related to action reversal, following from the results in [38]. Here "Plan Recognition" means deciding whether a sequence of actions P is a conformant plan for a planning problem $\mathcal{P}$, given P and $\mathcal{P}$ as input, which corresponds to Problem (P1), and "Plan Existence" means deciding whether some conformant plan P for $\mathcal{P}$ of length respectively 1, 2, or polynomial length exists, given $\mathcal{P}$ as

Table 2
Complexities of conformant planning problems, in terms of completeness (corresponding reversal problems in parentheses)

Conformant planning	$\|P\| = 1$	$\|P\| = 2$	$\|P\| > 2$
Plan Recognition ($\rightleftharpoons$ (P1))	Π_2^p (coNP)	Π_2^p (Π_2^p)	Π_2^p (Π_2^p)
Plan Existence ($\rightleftharpoons$ (P3))	Σ_3^p (Σ_2^p)	Σ_3^p (Σ_3^p)	Σ_3^p (Σ_3^p)

input, which corresponds to Problem (P3). Notice that for both problems the complexity is independent from the plan length; in particular hardness holds already for plans of length 1. This is in contrast to the complexity of execution reversals, which is always "easier" for plans of length 1, and in case of (P2) even for plan length 2.

9. Conclusion

We have formally defined the notions of an undo action and a reversal plan for an action sequence relative to a given action description, in the logic-based framework for action representation from [38]. By means of such an action or a plan, depending on information about the current and the original state (which might be zero), the effects of the action sequence are reversed so that the agent reaches the state she was at before taking that action sequence. Such reversals can be applied, for instance, to recover from failures and to optimize the behavior of an agent that tries to establish a goal by executing some plan.

In particular, based on undo actions and reverse plans, we have presented a new method for recovering from discrepancies in plan execution, that are detected by monitoring. According to this method, the agent finds a reverse plan to backtrack from the current state to a diagnosed point of failure, and continues with the execution of the original plan from that point on. By this way she can avoid an expensive (re)planning step. This recovery method is appealing given that such a reverse plan is short with respect to the overall plan to be executed. It is complementary to other recovery techniques used in execution monitoring, so it can be considered as a part of a richer monitoring framework where the most appropriate recovery method is picked by some heuristics.

An analysis of the computational complexity of undo actions and reverse plans has revealed that determining an undo action or reverse plan for an action sequence is intractable in general, and is complete for the class Σ_2^p respectively Σ_3^p in the Polynomial Hierarchy in general. The intractability may be explained, on the one hand, by the intractability of full propositional logic which underlies the framework, and, on the other hand, by the intrinsic complexity of nondeterminism; nonetheless, tractability can be gained under suitable restrictions. To cope with intractability and speed up reverse plan construction in practice, we have developed a two step approach in the spirit of knowledge compilation: First a reverse plan library is built offline by finding reverse plans for action sequences, and then reverse plans for the given action sequence are constructed online using this library. We have described some methods to build offline a reverse plan library, by solving some conformant planning problems or by finding models of some quantified Boolean formulas. We have also presented polynomial-time algorithms for online computation of a reverse plan from the library.

Some of the algorithms presented in this paper have been implemented after the completion of this article. The reduction of RPI generation to conformant planning has been adapted for the action language $\mathcal{K}$ [15] and implemented on top of the DLV$^{\mathcal{K}}$ [14] planner, exploiting its conformant planning support. Furthermore, also online reverse plan assembly has been implemented. The integration of the algorithms in a framework for reactive planning and execution monitoring is subject of ongoing work.

Acknowledgements

We thank Mikhail Soutchanski for useful discussions, and the anonymous referees for helpful suggestions to generalize and improve our framework. In particular, they suggested to consider a framework with sensory input and reversal conditional to constraints on the initial state. We further thank Ian Mackie for valuable comments and hints to work on reversible computation. This work was supported by FWF (Austrian Science Funds) under project P16536-N04, "An Answer Set Programming Framework for Reactive Planning and Execution Monitoring". The work of Wolfgang Faber was funded in part by an APART grant of the Austrian Academy of Sciences and by M.I.U.R. within projects "Potenziamento e Applicazioni della Programmazione Logica Disgiuntiva" and "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione".

Appendix A. Proofs for Section 5

Proof of Theorem 3. Let $D(\mathcal{A}, \mathcal{F})$ be an action domain description, let A and A' be two actions in $2^{\mathcal{A}}$, and let $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ be formulas.

Membership. The problem of deciding whether A' is a $\phi;\psi$-reverse action for A amounts to deciding whether the propositional formula $\phi(\mathcal{F}') \supset revAct(\mathcal{F}; \mathcal{F}'; A, A')$ is a tautology. Deciding whether a formula is a tautology is well known to be in coNP.

Hardness. Given a propositional formula $\alpha(\mathcal{X})$ on atoms $\mathcal{X} = \{X_1, \ldots, X_n\}$, deciding whether $\alpha(\mathcal{X})$ is a tautology is a coNP-complete problem. To show coNP-hardness, we need an action domain description $D(\mathcal{A}, \mathcal{F})$, two actions A and A' in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ obtainable from $\alpha(\mathcal{X})$ and $\mathcal{X}$ in polynomial time such that $\alpha(\mathcal{X})$ is a tautology iff A' is a $\phi;\psi$-reverse action for A. Without loss of generality, we may assume that $\alpha(\mathcal{X})$ holds whenever all atoms in $\mathcal{X}$ are mapped to true.

Let $\mathcal{F} = \mathcal{X}$ and $\mathcal{A} = \{A_0\}$. Then, take $state(\mathcal{F}) = \top$ and take

$$act(\mathcal{F}, \mathcal{A}, \mathcal{F}') = X'_1 \wedge \cdots \wedge X'_n \wedge \big((X_1 \wedge \cdots \wedge X_n) \vee \neg\alpha(\mathcal{X})\big).$$

Take $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $A = A' = \{\}$. Notice that the action domain $D(\mathcal{A}, \mathcal{F})$ is deterministic.[11]

Suppose that A' is a reverse action for A. Then, by definition, since the action domain description is deterministic, for any two states S and S', $tr(S, A, S')$ implies $tr(S', A', S)$, and thus $S = S'$. On the other hand, by the definition of *act*, if $tr(S, A, S')$ holds then every X_i is true at S', i.e., $S' = \{X_1, \ldots, X_n\}$, and one of the two conditions hold for S: $S = \{X_1, \ldots, X_n\}$ or $\neg\alpha(S)$. Then, due to the definition of a reverse action, $S = S'$ and, for every proper subset $\mathcal{X}_i$ of $\mathcal{X}$, $\alpha(\mathcal{X}_i)$. This, with the assumption on $\alpha(\mathcal{X})$, implies that $\alpha(\mathcal{X})$ is a tautology.

Conversely, suppose that $\alpha(\mathcal{X})$ is a tautology. Then, for any two states S and S', $tr(S, A, S')$ holds iff $S = S' = \{X_1, \ldots, X_n\}$. Therefore, A' is a reverse action for A. □

Proof of Theorem 4. Let $D(\mathcal{A}, \mathcal{F})$ be an action domain description, let A be an action in $2^{\mathcal{A}}$, let AS be a sequence of actions in $2^{\mathcal{A}}$, and let $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ be formulas.

Membership. Formula (9) with $|\langle A\rangle| = m = 1$ and $|AS| = k$ is:

$$\exists\mathcal{F}_0, \ldots, \mathcal{F}_k\ (\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A)\ \wedge \mathcal{F}' \equiv \mathcal{F}_k) \supset$$
$$\forall\mathcal{F}'_0, \mathcal{F}'_1 \exists\mathcal{F}''_1 \big(\mathcal{F}'_0 \equiv \mathcal{F}' \supset tr(\mathcal{F}'_0, A'_0, \mathcal{F}''_1) \wedge \big(tr(\mathcal{F}'_0, A'_0, \mathcal{F}'_1) \supset \mathcal{F}'_1 \equiv \mathcal{F}\big)\big).$$

In this case, $\mathcal{F}''_1$ can be avoided: The last part of the formula states that there exists a transition from $\mathcal{F}'_0$ when executing A'_0, and each state reached via such a transition is equivalent to $\mathcal{F}$. We can therefore equivalently require that a transition from $\mathcal{F}'_0$ to $\mathcal{F}$ exists when executing A'_0 and that each state reached by a transition from $\mathcal{F}'_0$ when executing A'_0 is equivalent to $\mathcal{F}$, resulting in:

$$\exists\mathcal{F}_0, \ldots, \mathcal{F}_k\ (\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A)\ \wedge \mathcal{F}' \equiv \mathcal{F}_k) \supset$$
$$\forall\mathcal{F}'_0, \mathcal{F}'_1 \big(\mathcal{F}'_0 \equiv \mathcal{F}' \supset tr(\mathcal{F}'_0, A'_0, \mathcal{F}) \wedge \big(tr(\mathcal{F}'_0, A'_0, \mathcal{F}'_1) \supset \mathcal{F}'_1 \equiv \mathcal{F}\big)\big).$$

This formula is equivalent to a Quantified Boolean Formula (QBF) of the form

$$\big(\forall\mathcal{X}\ \beta(\mathcal{X}, \mathcal{F}, \mathcal{F}')\big).$$

Then (10) with $|\langle A\rangle| = m = 1$ and $|AS| = k$ is equivalent to a QBF of the form

$$\forall\mathcal{X}\big(\neg\psi(\mathcal{F}) \vee \neg\phi(\mathcal{F}') \vee \beta(\mathcal{X}, \mathcal{F}, \mathcal{F}')\big).$$

This implies that the problem of deciding whether $\langle A\rangle$ is a $\phi;\psi$-reverse plan for AS reduces in polynomial time to the problem of deciding whether a QBF of the form

$$\forall\mathcal{X}\ \alpha(\mathcal{X})$$

where $\alpha(\mathcal{X})$ is quantifier-free, is true. The latter problem is well known to be in coNP.

Hardness. See the coNP-hardness proof for Theorem 3. □

Proof of Theorem 5. Let $D(\mathcal{A}, \mathcal{F})$ be an action domain description, let R and AS be sequences of actions in $2^{\mathcal{A}}$, and let $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ be formulas.

[11] An action domain is *deterministic* iff $\forall\mathcal{F}, \mathcal{A}, \mathcal{F}', \mathcal{F}''\ (tr(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge tr(\mathcal{F}, \mathcal{A}, \mathcal{F}'') \supset \mathcal{F}' \equiv \mathcal{F}'')$.

Membership. Formula (9) is a QBF of the form

$$\big(\exists\mathcal{X}\ \beta(\mathcal{X},\mathcal{F},\mathcal{F}')\big) \supset \forall\mathcal{Y}\ \exists\mathcal{Z}\ \gamma(\mathcal{Y},\mathcal{F},\mathcal{F}').$$

Hence, (10) is equivalent to the QBF

$$\forall\mathcal{X}\ \forall\mathcal{Y}\ \exists\mathcal{Z}\big(\neg\phi(\mathcal{F}')\vee\neg\psi(\mathcal{F})\vee\neg\beta(\mathcal{X},\mathcal{F},\mathcal{F}')\vee\gamma(\mathcal{Y},\mathcal{F},\mathcal{F}')\big).$$

This implies that the problem of deciding whether R is a $\phi;\psi$-reverse plan for AS reduces in polynomial time to the problem of deciding whether a QBF of the form

$$\forall\mathcal{X}\ \exists\mathcal{Y}\ \alpha(\mathcal{X},\mathcal{Y}) \tag{A.1}$$

where $\alpha(\mathcal{X},\mathcal{Y})$ is quantifier-free, is true. The latter problem is well known to be in Π_2^p.

Hardness. Deciding whether a QBF of the form (A.1), where $\alpha(\mathcal{X},\mathcal{Y})$ is quantifier-free, is true is a Π_2^p-complete problem. To show Π_2^p-hardness for $|R|=2$, we need an action domain description $D(\mathcal{A},\mathcal{F})$, two action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{X},\mathcal{Y})$, $\mathcal{X}$, and $\mathcal{Y}$ in polynomial time such that (A.1) is true iff R is a $\phi;\psi$-reverse plan for AS.

Let $\mathcal{F}=\mathcal{X}\cup\mathcal{Y}\cup\{0,1,2\}$ and let $\mathcal{A}=\{A\}$. Then, take

$$state(\mathcal{F}) = \ (0\vee 1\vee 2)\wedge(\neg 0\vee\neg 1)\wedge(\neg 0\vee\neg 2)\wedge(\neg 1\vee\neg 2)$$

(i.e., exactly one of 0, 1, and 2 is true) and take

$$\begin{aligned} act(\mathcal{F},\mathcal{A},\mathcal{F}') = {} & (\mathcal{X}\equiv\mathcal{X}')\wedge(A\equiv 0)\wedge \\ & \left(0\supset 1'\wedge\bigwedge_{Y_i\in\mathcal{Y}}Y_i\right)\wedge \\ & \big(1\supset 2'\wedge\alpha(\mathcal{X}',\mathcal{Y}')\big)\wedge \\ & \left(2\supset 0'\wedge\bigwedge_{Y_i\in\mathcal{Y}}Y_i'\right). \end{aligned}$$

Take $\phi(\mathcal{F})=\top$, $\psi(\mathcal{F})=\top$, $AS=\langle A\rangle$, and $R=\langle\emptyset,\emptyset\rangle$.

Suppose that R is a reverse plan for AS. Take any two states S and S', such that $tr(S,A,S')$. Then, due to the definition of *act*, every Y_i in $\mathcal{Y}$ is true at S, i.e., $\mathcal{Y}\subseteq S$. From S', we reach some state S'' such that $\alpha(\mathcal{X}\cap S'',\mathcal{Y}\cap S'')$ holds. Due to the definition of *act*, $S''\cap\mathcal{X}=S'\cap\mathcal{X}=S\cap\mathcal{X}$. Finally, due to the definition of a reverse plan, from each state S'' the state S (and only this state) can be reached. Note that, since $\mathcal{Y}\subseteq S$, each state S can be characterized by some subset $\mathcal{X}_S$ of $\mathcal{X}$, i.e., by the set $\mathcal{X}_S=\mathcal{X}\cap S$; for each such state S, state S'' can be characterized by some subset $\mathcal{Y}_{S''}$ of $\mathcal{Y}$, i.e., the set $\mathcal{Y}_{S''}=\mathcal{Y}\cap S''$, such that $\alpha(\mathcal{X}_S,\mathcal{Y}_{S''})$ holds. Then R is a reverse plan for the action AS iff formula (A.1) is true.

The hardness proof above can be easily extended to any $|R|\geqslant 2$: First introduce further fluents $3,4,\ldots,|R|$; and then modify the definition as follows:

$$\begin{aligned} act(\mathcal{F},\mathcal{A},\mathcal{F}') = {} & (\mathcal{X}\equiv\mathcal{X}')\wedge(A\equiv 0)\wedge \\ & \left(0\supset 1'\wedge\bigwedge_{Y_i\in\mathcal{Y}}Y_i\right)\wedge \\ & \big(1\supset 2'\wedge\alpha(\mathcal{X}',\mathcal{Y}')\big)\wedge \\ & \vdots \\ & \big(|R|-1\supset|R|'\wedge\mathcal{Y}\equiv\mathcal{Y}'\big)\wedge \\ & \left(|R|\supset 0'\wedge\bigwedge_{Y_i\in\mathcal{Y}}Y_i'\right). \qquad \square \end{aligned}$$

Proof of Theorem 6. Let $D(\mathcal{A},\mathcal{F})$ be an action domain description, and let A be an action in $2^{\mathcal{A}}$.

Membership. Facts (*) and (**) presented in Section 6.3 imply that without loss of generality, $\phi(\mathcal{F})$ and $\psi(F)$ are conjunctions of literals (i.e., symbols in $\mathcal{F}$, possible preceded by the classical negation symbol $\neg$). Suitable such $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ can be guessed in polynomial time. Furthermore, deciding, for some $\mathcal{F}$ and $\mathcal{F}'$, whether $\phi(F) \wedge tr(\mathcal{F}, A, \mathcal{F}') \wedge \phi(\mathcal{F}')$ holds is in NP. On the other hand, due to Theorem 3, we can decide whether A' is a ϕ; ψ-reverse action for A in coNP. Therefore, we can decide whether there exist formulas $\phi(\mathcal{F})$ and $\psi(F)$ and an action A' in $2^{\mathcal{A}}$ such that A' is a ϕ; ψ-reverse action for A in nondeterministic polynomial time with an NP oracle, and hence this decision problem is in Σ_2^p.

Hardness. The hardness can be shown by a reduction from 2-QSAT, i.e., the problem of deciding whether a QBF of the form

$$\exists \mathcal{Y}\, \forall \mathcal{X}\, \alpha(\mathcal{X}, \mathcal{Y}) \tag{A.2}$$

with $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ is true. To show Σ_2^p-hardness, we need an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action A in $2^{\mathcal{A}}$, each obtainable from $\alpha(\mathcal{X}, \mathcal{Y})$, $\mathcal{X}$, and $\mathcal{Y}$ in polynomial time, such that (A.2) is satisfiable iff there exist formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$ and an action A' in $2^{\mathcal{A}}$ such that A' is a ϕ; ψ-reverse action for A. Without loss of generality, we may assume that $\alpha(\mathcal{X}, \mathcal{Y})$ holds whenever all atoms in $\mathcal{X}$ are mapped to true.

Let $\mathcal{F} = \mathcal{X} \cup \mathcal{Y}$ and $\mathcal{A} = \{B, B_1, \ldots, B_m\}$. Take $state(\mathcal{F}) = \top$ and take

$$\begin{aligned} act(\mathcal{F}, \mathcal{A}, \mathcal{F}') = {} & \bigwedge_{i=1}^{m} \big((B_i \supset Y_i) \wedge (\neg B_i \supset \neg Y_i)\big) \wedge \\ & \left(B \supset \bigwedge_{X_i \in \mathcal{X}} X_i\right) \wedge \\ & \left(\neg B \supset \bigwedge_{i=1}^{m} \neg Y_i' \wedge \left(\bigwedge_{X_i \in \mathcal{X}} X_i' \vee \neg\alpha(\mathcal{X}', \mathcal{Y})\right)\right). \end{aligned}$$

Take $A = \{B\}$. For any state S, let $\sigma_S(\mathcal{F})$ be the conjunction $\bigwedge_{f \in S} f \wedge \bigwedge_{f \in \mathcal{F} \setminus S} \neg f$, and B_S be the subset of $\{B_1, \ldots, B_m\}$ such that $Y_i \in S$ iff $B_i \in B_S$.

Take any two states S and S' such that $tr(S, A, S')$. Suppose that $A' = B_{S'}$ is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$- reverse action for A. Due to the preconditions of A, every $Y_i \in \mathcal{Y}$ is false at S and every $X_i \in \mathcal{X}$ is true at S, i.e., $S = \mathcal{X}$, and thus $\sigma_S(\mathcal{F})$ must be $\sigma_{\mathcal{X}}(\mathcal{F})$. On the other hand, by the definition of a conditional reversal, at S', $\sigma_{S'}$ holds. Then, due to the preconditions of $B_{S'}$, the action $B_{S'}$ is executable at S', where S' contains some subset $\mathcal{Y}_{S'}$ of $\mathcal{Y}$ ($\mathcal{Y}_{S'} = \mathcal{Y} \cap S'$). Due to the effects of $B_{S'}$, executing $B_{S'}$ at S' leads to a state S'' such that $S'' = \mathcal{X}$ or $\alpha(\mathcal{X} \cap S'', \mathcal{Y}_{S'})$ does not hold. Since $B_{S'}$ is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$-reverse action for A, every such state S'' is identical to S. Then $S = S'' = \mathcal{X}$ and, for every $\mathcal{X}' \subset \mathcal{X}$, $\alpha(\mathcal{X}', \mathcal{Y}_{S'})$ is true. This, with the assumption on $\alpha(\mathcal{X}, \mathcal{Y})$, implies that (A.2) is true.

Conversely, suppose (A.2) is true, and let $\mathcal{Y}'$ be a subset of $\mathcal{Y}$ such that $\alpha(\mathcal{X}, \mathcal{Y}')$ is true. Then, for any two states S and S' such that $tr(S, A, S')$ holds and $S' = \mathcal{X} \cup \mathcal{Y}'$ (which implies $S = \mathcal{X}$), the action $B_{S'}$ is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$-reverse action for A. □

Proof of Theorem 7. Let $D(\mathcal{A}, \mathcal{F})$ be an action domain description, and let AS be a sequence of actions in $2^{\mathcal{A}}$.

Membership. Facts (*) and (**) presented in Section 6.3 imply that, without loss of generality, $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ can be taken to be conjunctions of literals that hold at the single states S_{n+1} and S_0, respectively. Let $AS = \langle A_0, \ldots, A_n \rangle$ such that every $A_i \in 2^{\mathcal{A}}$. Take $R = \langle A_{n+1}, A_{n+2} \rangle$ such that $A_{n+1}, A_{n+2} \in 2^{\mathcal{A}}$. Then R is a ϕ; ψ-reverse plan for AS iff there exist states S_0 and S_{n+1} such that $\psi(S_0)$ holds and

(i) execution of AS at S_0 leads to S_{n+1},
(ii) execution of AS leads to S_{n+1} only when executed at S_0,
(iii) execution of R at S_{n+1} leads to S_0,
(iv) whenever execution of A_{n+1} at S_{n+1} leads to S_{n+2}, execution of A_{n+2} at S_{n+2} leads to S_0, and
(v) execution of R at S_{n+1} always leads to S_0.

The conditions (i)–(v) can be checked with an NP oracle in polynomial time. Since $\phi(\mathcal{F})$, R, S_0 and S_{n+1} can be guessed in polynomial time, it follows that the problem of deciding that, for some formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$ and a sequence R of at most two actions in $2^{\mathcal{A}}$, R is a ϕ; ψ-reverse plan for AS is in Σ_2^p.

Hardness. The hardness can be shown by a reduction from 2-QSAT, i.e., the problem of deciding whether a QBF of the form (A.2) with $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ is true. To show Σ_2^p-hardness, we need an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action sequence AS consisting of actions in $2^{\mathcal{A}}$, each obtainable from $\alpha(\mathcal{X}, \mathcal{Y})$, $\mathcal{X}$, and $\mathcal{Y}$ in polynomial time, such that (A.2) is satisfiable iff there exist a formula $\phi(\mathcal{F})$ and an action sequence R of at most two actions in $2^{\mathcal{A}}$ such that R is a ϕ; ψ-reverse plan for AS. Without loss of generality, we may assume that $\alpha(\mathcal{X}, \mathcal{Y})$ holds whenever all atoms in $\mathcal{X}$ are mapped to true.

Let $\mathcal{F} = \mathcal{X} \cup \mathcal{Y} \cup \{0, 1, 2\}$ and $\mathcal{A} = \{B, B_1, \ldots, B_m\}$. Take

$$state(\mathcal{F}) = (0 \vee 1 \vee 2) \wedge (\neg 0 \vee \neg 1) \wedge (\neg 0 \vee \neg 2) \wedge (\neg 1 \vee \neg 2)$$

(i.e., exactly one of 0, 1, and 2 is true) and take

$$\begin{aligned} act(\mathcal{F}, \mathcal{A}, \mathcal{F}') = {} & \bigwedge_{i=1}^{m} \big((B_i \supset Y_i) \wedge (\neg B_i \supset \neg Y_i)\big) \wedge (B \equiv 0) \wedge \\ & \left(0 \supset 1' \wedge \bigwedge_{X_i \in \mathcal{X}} X_i\right) \wedge \\ & \left(1 \supset 2' \wedge \bigwedge_{i=1}^{m} \neg Y_i' \wedge \left(\bigwedge_{X_i \in \mathcal{X}} X_i' \vee \neg\alpha(\mathcal{X}', \mathcal{Y})\right)\right) \\ & (2 \supset 0' \wedge \mathcal{Y} \equiv \mathcal{Y}' \wedge \mathcal{X} \equiv \mathcal{X}'). \end{aligned}$$

Take $AS = \langle B \rangle$. For any state S, let $\sigma_S(\mathcal{F})$ be the conjunction $1 \wedge \bigwedge_{f \in S} f \wedge \bigwedge_{f \in \mathcal{F} \setminus S} \neg f$, and B_S be the subset of $\{B_1, \ldots, B_m\}$ such that $Y_i \in S$ iff $B_i \in B_S$.

Take any two states S and S' such that $tr(S, AS, S')$. Suppose that the sequence $R = \langle B_{S'}, \emptyset \rangle$ is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$-reverse plan for AS. Due to the definition of act, $S = \{0\} \cup \mathcal{X}$, and $1 \in S'$. On the other hand, by the definition of a conditional reversal, at S', $\phi(S')$ holds; S' contains some subset of $\mathcal{Y}$ ($\mathcal{Y}_{S'} = \mathcal{Y} \cap S'$). Then, due to the preconditions of $B_{S'}$, the action $B_{S'}$ is executable at S'. Due to the effects of $B_{S'}$, executing $B_{S'}$ at S' leads to a state S'' such that $2 \in S''$ and the following holds: $\mathcal{X} \subseteq S''$ or $\neg\alpha(\mathcal{X} \cap S'', \mathcal{Y}_{S'})$. By the definition of act, S'' leads to S''' such that $0 \in S'''$ and $S''' \setminus \{0\} = S'' \setminus \{2\}$. Then, the following holds: $\mathcal{X} \subseteq S'''$ or $\neg\alpha(\mathcal{X} \cap S''', \mathcal{Y}_{S'})$. Since R is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$-reverse plan for A, every such state S''' is identical to S. Then $S = S''' = \mathcal{X}$ and, for every $\mathcal{X}' \subset \mathcal{X}$, $\alpha(\mathcal{X}', \mathcal{Y}_{S'})$ is true. This, with the assumption on $\alpha(\mathcal{X}, \mathcal{Y})$, implies that (A.2) is true.

Conversely, suppose (A.2) is true, and let $\mathcal{Y}'$ be a subset of $\mathcal{Y}$ such that $\alpha(\mathcal{X}, \mathcal{Y}')$ is true. Then, for any two states S and S' such that $tr(S, AS, S')$, and that $S' = \mathcal{X} \cup \mathcal{Y}'$, the action sequence $\langle B_{S'}, \emptyset \rangle$ is a $\sigma_{S'}(\mathcal{F})$; $\sigma_S(\mathcal{F})$-reverse action for AS. □

Proof of Theorem 8. Let $D(\mathcal{A}, \mathcal{F})$ be an action domain description, and let AS be a sequence of actions in $2^{\mathcal{A}}$.

Membership. Due to Facts (*) and (**) presented in Section 6.3, without loss of generality, take $\phi(\mathcal{F})$ and $\psi(F)$ to be a conjunction of literals. Therefore, suitable $\phi(\mathcal{F})$, $\psi(\mathcal{F})$, and R such that R is a ϕ; ψ-reversal plan for AS can be guessed and, by Theorem 5, be checked in polynomial time with a Σ_2^p oracle. This proves membership in Σ_3^p.

Hardness. The hardness can be shown by a reduction from the problem of deciding whether a QBF of the form

$$\exists \mathcal{Z} \forall \mathcal{X}\, \exists \mathcal{Y}\, \alpha(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) \tag{A.3}$$

with $\mathcal{Z} = \{Z_1, \ldots, Z_m\}$ is true. To show Σ_3^p-hardness, we need an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action sequence AS consisting of actions in $2^{\mathcal{A}}$, each obtainable from $\alpha(\mathcal{Z}, \mathcal{X}, \mathcal{Y})$, $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{Y}$ in polynomial time, such that (A.3) is satisfiable iff there exist formulas $\phi(\mathcal{F})$, $\psi(\mathcal{F})$, and an action sequence R of polynomial number of actions in $2^{\mathcal{A}}$ such that R is a ϕ; ψ-reverse plan for AS.

Let $\mathcal{F} = \mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y} \cup \{0, 1, 2, 3\}$ and $\mathcal{A} = \{A, B_1, \ldots, B_m\}$. Take

$$state(\mathcal{F}) = (0 \vee 1 \vee 2 \vee 3) \wedge \bigwedge_{0 \leqslant i < j \leqslant 3} \neg i \vee \neg j$$

and take

$$act(\mathcal{F},\mathcal{A},\mathcal{F}') = \big(2 \supset (\mathcal{X} \equiv \mathcal{X}')\big) \wedge (A \equiv 0) \wedge \Big(0 \supset 1' \wedge \bigwedge_{W \in \mathcal{X}\cup\mathcal{Y}\cup\mathcal{Z}} W\Big) \wedge \Big(1 \supset 2' \wedge \bigwedge_{Z_j \in \mathcal{Z}} \big[(B_j \equiv Z_j) \wedge (Z_j \equiv Z_j')\big]\Big) \wedge \big(2 \supset 3' \wedge \alpha(\mathcal{Z},\mathcal{X},\mathcal{Y}')\big) \wedge \Big(3 \supset 0' \wedge \bigwedge_{W \in \mathcal{X}\cup\mathcal{Y}\cup\mathcal{Z}} W\Big).$$

Take $AS = \langle A\rangle$. Let us first consider reverse plans of length 3. For convenience, for every $i \in \{0,\ldots,3\}$, we will denote by S_i a state containing i. For any state S, let $\sigma_S(\mathcal{F})$ be the conjunction $1 \wedge \bigwedge_{f\in S} f \wedge \bigwedge_{f\in\mathcal{F}\setminus S} \neg f$; and A_1 be the subset of $\{B_1,\ldots,B_m\}$ such that, for every $i \in \{1,\ldots,m\}$, $Z_i \in S_1$ iff $B_i \in A_1$.

Take any two states S_0 and S_1 such that $tr(S_0, AS, S_1)$; this is possible due to the definition of *act*. Note that $\mathcal{Z}\cup\mathcal{X}\cup\mathcal{Y} \subseteq S_0$ (and thus in fact $S_0 = \mathcal{Z}\cup\mathcal{X}\cup\mathcal{Y}$). Suppose that $R = \langle A_1, \emptyset, \emptyset\rangle$ is a $\phi;\psi$-reverse plan for AS. Due to the definition of *act*, some subset of $\mathcal{Z}$ is contained in S_1. From S_1, we reach at a state S_2 where $S_2 \cap \mathcal{Z} = S_1 \cap \mathcal{Z}$; any subset of $\mathcal{X}$ can be contained in S_2. From any such state S_2, we reach at a state S_3 where $\alpha(\mathcal{Z}\cap S_1, \mathcal{X}\cap S_2, \mathcal{Y}\cap S_3)$ holds. Since R is a $\phi;\psi$-reverse plan for AS, for each S_2 some such S_3 must exist; and from S_3, we reach the state S_0 where $\mathcal{Z}\cup\mathcal{X}\cup\mathcal{Y} \subseteq S_0$. This is tantamount to (A.3) being true.

Now consider $\phi;\psi$-reverse plans $R = \langle A_1,\ldots,A_n\rangle$ of length greater than 3. We can always take the prefix $\langle A_1, A_2, A_3\rangle$ of R to be another $\phi;\psi$-reverse plan (as we have seen above). □

Proof of Theorem 9. Let $D(\mathcal{A},\mathcal{F})$ be an action domain description, let AS be a sequence of actions in $2^{\mathcal{A}}$, and let $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ be formulas.

Membership. The problem is in Σ_2^p, since a suitable R can be guessed and, by Theorem 4, checked in polynomial time with the help of an *NP* oracle.

Hardness. The hardness can be shown by a reduction from the problem of deciding whether a QBF of the form

$$\exists\mathcal{Z}\forall\mathcal{X}\ \alpha(\mathcal{Z},\mathcal{X}) \tag{A.4}$$

where $\alpha(\mathcal{Z},\mathcal{X})$ is quantifier-free, is true. To show Σ_2^p-hardness, we need an action domain description $D(\mathcal{A},\mathcal{F})$, an action sequence AS consisting of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{Z},\mathcal{X})$, $\mathcal{Z}$, and $\mathcal{X}$ in polynomial time, such that (A.4) is satisfiable iff there exists an action sequence R consisting of one action in $2^{\mathcal{A}}$ such that R is a $\phi;\psi$-reverse plan for AS.

Let $\mathcal{F} = \mathcal{X}\cup\{0\}$ and $\mathcal{A} = \mathcal{Z}\cup\{A\}$. Then, take $state(\mathcal{F}) = \top$, and take

$$act(\mathcal{F},\mathcal{A},\mathcal{F}') = (\mathcal{X} \equiv \mathcal{X}') \wedge (A \equiv 0) \wedge \big(\neg 0 \supset \alpha(\mathcal{Z},\mathcal{X}')\big).$$

Take $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $AS = \langle A\rangle$. Take any two states S and S' such that $tr(S, AS, S')$ holds. By definition, $S\setminus\{0\} = S'\setminus\{0\}$; any subset of $\mathcal{X}$ is contained in S and thus S'. State S is reached from S' by some action $A' \in 2^{\mathcal{A}}$ iff $\alpha(A'\cap\mathcal{Z}, \mathcal{X}\cap S)$ is true. Such an action A' exists iff (A.4) is true. □

Proof of Theorem 10. Let $D(\mathcal{A},\mathcal{F})$ be an action domain description, let AS be a sequence of actions in $2^{\mathcal{A}}$, and let $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$ be formulas.

Membership. The problem is in Σ_3^p, since a suitable R can be guessed and, by Theorem 5, checked in polynomial time with the help of a Σ_2^p oracle.

Hardness. The hardness can be shown by a reduction from the problem of deciding whether a QBF of the form

$$\exists\mathcal{Z}\forall\mathcal{X}\ \exists\mathcal{Y}\ \alpha(\mathcal{Z},\mathcal{X},\mathcal{Y}) \tag{A.5}$$

where $\alpha(\mathcal{Z},\mathcal{X},\mathcal{Y})$ is quantifier-free, is true. To show Σ_3^p-hardness, we need an action domain description $D(\mathcal{A},\mathcal{F})$, an action sequence AS consisting of actions in $2^{\mathcal{A}}$, and formulas $\phi(\mathcal{F})$ and $\psi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{Z},\mathcal{X})$, $\mathcal{Z}$,

and $\mathcal{X}$ in polynomial time, such that (A.5) is satisfiable iff there exists an action sequence R consisting of at least two actions in $2^{\mathcal{A}}$ such that R is a ϕ; ψ-reverse plan for AS.

Let $\mathcal{F} = \mathcal{X} \cup \mathcal{Y} \cup \{0, 1, 2\}$ and $\mathcal{A} = \mathcal{Z} \cup \{A\}$. Then, let

$$state(\mathcal{F}) = (0 \vee 1 \vee 2) \wedge \bigwedge_{0 \leqslant i < j \leqslant 2} (\neg i \vee \neg j)$$

and let

$$\begin{aligned} act(\mathcal{F}, \mathcal{A}, \mathcal{F}') = (\mathcal{X} \equiv \mathcal{X}') \wedge (A \equiv 0) \wedge \\ \left(0 \supset 1' \wedge \bigwedge_{W \in \mathcal{Y}} W\right) \wedge \\ \left(1 \supset 2' \wedge \alpha(\mathcal{Z}, \mathcal{X}', \mathcal{Y}')\right) \wedge \\ \left(2 \supset 0' \wedge \bigwedge_{W \in \mathcal{Y}} W'\right). \end{aligned}$$

Take $\phi(\mathcal{F}) = \top$, $\psi(\mathcal{F}) = \top$, and $AS = \langle A \rangle$.

Consider any states S and S' such that $tr(S, A, S')$ holds. Then $\mathcal{Y} \subseteq S$ holds, and S' coincides with S on $\mathcal{X}$. Take $R = \langle A_1, A_2 \rangle$ such that the following conditions hold: (1) The execution of an action $A_1 \subseteq \mathcal{Z}$ at S' leads to a state S'' such that $\alpha(A_1, \mathcal{X} \cap S', \mathcal{Y})$ is true; (2) $A_2 \in 2^{\mathcal{A} \setminus \{A\}}$. Note that executing A_2 at S'' leads to the single state S. Then R is a ϕ; ψ-reverse plan for AS iff QBF (A.5) is true.

Any reverse plan of length greater than two for AS is of the form

$$R = \langle A_{1,1}, A_{1,2}, A, A_{2,1}, A_{2,2}, \ldots, A, A_{n,1}, A_{n,2} \rangle,$$

where $n \geqslant 1$ and $A_{i,j} \subseteq \mathcal{Z}$, such that the following conditions hold: (1) The execution of an action $A_{i,1} \subseteq \mathcal{Z}$ at S' leads to a state S'' such that $\alpha(A_{i,1}, \mathcal{X} \cap S', \mathcal{Y})$ is true; (2) $A_{i,2} \in 2^{\mathcal{A} \setminus \{A\}}$. □

Appendix B. Proofs for Section 6

Proof of Theorem 11. Consider any conformant plan $P = \langle A_0, \ldots, A_{n-1} \rangle$ of length $n \geqslant 1$. Starting at any initial state $S_0 \cup \tilde{S}_0$, by executing the actions in P, some state $S_n \cup \tilde{S}_n$ is reached such that $S_n \equiv \tilde{S}_n$ holds. By (16), $S_0 \equiv \tilde{S}_0$ holds, and thus by (15), also $S_0 \equiv \tilde{S}_n$ holds. That is, by executing $A_0, \ldots, A_{n-1}$ when starting at state S in the action domain D, we reach S (and only S). Therefore, whatever state S_i, $i \leqslant n$, is reached by executing a prefix $A_0, \ldots, A_{i-1}$ of P starting from S, by taking the remainder $A_i, \ldots, A_{n-1}$, we arrive at state S (and only S) for sure. By definition, this means that $R = \langle A_i, \ldots, A_{n-1} \rangle$ is a reverse plan for $AS = \langle A_0, \ldots, A_{i-1} \rangle$ relative to D. □

Proof of Theorem 12. First, assume that $\langle A_i, \ldots, A_{n-1} \rangle$ is a ϕ;ψ-reverse plan of $\langle A_0, \ldots, A_{i-1} \rangle$ relative to D (where $n \geqslant i > 0$). We show that then $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1} \rangle$ is a conformant plan of P^c_{rev}, i.e. (11) holds for $init^c_{rev}(\mathcal{F}^c_{rev})$, $tr^c_{rev}(\mathcal{F}^c_{rev}, \mathcal{A}^c_{rev}, \mathcal{F}^{c\,\prime}_{rev})$, and $goal^c_{rev}(\mathcal{F}^c_{rev})$. Let $P = \langle A'_0, \ldots, A'_{i-1}, A'_i, A'_{i+1}, \ldots, A'_n \rangle$. Then (11) reads as follows:

$$\begin{aligned} &\forall \mathcal{F}_0, \ldots, \mathcal{F}_n \exists \mathcal{F}'_0, \ldots, \mathcal{F}'_{n+1} \;\; init^c_{rev}(\mathcal{F}'_0) \\ &\quad \wedge \bigwedge_{t=0}^{n} \left(init^c_{rev}(\mathcal{F}_0) \wedge tr^c_{rev,t}(\mathcal{F}A') \supset tr^c_{rev}(\mathcal{F}_t, A'_t, \mathcal{F}'_{t+1})\right) \\ &\quad \wedge \left(init^c_{rev}(\mathcal{F}_0) \wedge tr_{rev,n+1}(\mathcal{F}, A') \supset goal^c_{rev}(\mathcal{F}_{n+1})\right) \end{aligned} \tag{B.1}$$

The first conjunct is satisfiable: In the original domain D transitions exist, and thus also some state exists. Hence, a state S'_0 in D^c_{rev} exists such that $init^c_{rev}(S_0)$ holds.

For the second conjunct, consider any evolution $S_0, A'_0, S_1, \ldots, A'_t$, $0 \leqslant t \leqslant n$. Observe that if $t \leqslant i - 1$, then there exists a successor state S'_t in which *normal* does not hold (a "don't care" state). For $t = i$, there are three cases for evolutions: (i) State S_i is "don't care" (*normal* does not hold), then a successor "don't care" state exists by (26); (ii)

normal holds but $\phi(S_i \cap \mathcal{F}) \wedge \psi(S_i \cap \tilde{\mathcal{F}})$ does not hold, which means that a successor "don't care" state exists by (21) (note that $S_i \cap \tilde{\mathcal{F}} = S_0 \cap \mathcal{F}$ if *normal* holds); (iii) *normal* $\wedge\, \phi(S_i \cap \mathcal{F}) \wedge \psi(S_i \cap \tilde{\mathcal{F}})$ holds, which means that a successor state exists equal to state S_i, except for the fact that *sep_occ* holds by (23), (24). For $i < t \leqslant n$, there are two cases: (i) State S_t is "don't care" (*normal* does not hold), then a successor "don't care" state exists by (26); (ii) *normal* holds in state t, then by virtue of (10) and (9) the formula $tr_{rev}(S_t \cap \mathcal{F}_{rev} A_t \cap \mathcal{A}\mathcal{F}')$ is satisfiable, and hence some successor state S_{t+1} exists which means that (20) is satisfiable. Indeed, note that in any evolution of such a state S_t, $\phi(S_i \cap \mathcal{F})$ and $\psi(S_i \cap \tilde{\mathcal{F}})$ hold, and therefore (9) must hold. Furthermore, *sep_occ* holds at S_{t+1} by (25).

For the final conjunct similar considerations apply. Either state S_{n+1} is "don't care", i.e., $\neg$*normal* holds, or otherwise by virtue of (10) the formula $goal_{rev}(\mathcal{F}_{rev})$ and, as argued in the discussion above, *sep_occ* hold at S_{n+1}. Thus, $goal^c_{rev}(S_{n+1})$ holds.

In total, we have demonstrated that (B.1) is true. We therefore obtain that P is a conformant plan of P^c_{rev}.

Now, assume that $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geqslant i > 0$, is a conformant plan of P^c_{rev}. We show that $\langle A_i, \ldots, A_{n-1}\rangle$ is a $\phi;\psi$-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D.

Consider the trajectories $S_0, A_0, \ldots, A_{i-1}, S_i, \{sep\}, S_{i+1}, A_i, \ldots, A_{n-1}, S_{n+1}$ for P with respect to P^c_{rev}. We show now that (10) holds for $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$, where for any $j \geqslant 0$, $S^\star_j$ is the restrictions of S_j to the fluents in D.

Assume first that each trajectory contains a state S_j, where $0 < j \leqslant i$, in which *normal* does not hold. Then the antecedent of (9) is false, and hence the formula $multiRev(S^\star_0, S^\star_i; [A_0, \ldots, A_i], [A_{i_1}, \ldots, A_{n-1}])$ is true as well as (10) for $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$.

Next, assume that each trajectory contains a state S_j, where $0 < j \leqslant i+1$, in which *normal* does not hold. Then, either $j \leqslant i$ and as before the antecedent of (9) is false, or $j = i+1$ and by (21) either $\phi(S^\star_i)$ or $\psi(S^\star_0)$ does not hold (note that $\tilde{\mathcal{F}}$ has in S_i the same value as $\mathcal{F}$). In any case, (10) is true for $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$.

Note that in the situations considered so far, the action sequence is either not executable, or starts in a state not satisfying ψ, or leads only from states in which ψ holds to states in which ϕ does not hold.

Now assume that some trajectory exists in which *normal* holds at every state S_j, where $0 \leqslant j \leqslant i+1$. Here $\phi(S^\star_i)$ clearly holds, and the antecedent of (9) holds. Since P is a conformant plan, for each state S_k, $k \geqslant i+1$, by (20) a state S_{k+1} exists such that $tr^c_{rev}(S_k, A_{k-1}, S_{k+1})$ holds. By construction, also $tr(S^\star_k, A_{k-1} \cap \mathcal{A}, S^\star_{k+1})$ is then true. Finally, $goal^c_{rev}(S_{n+1})$ holds. Since *normal* must be true at S_{n+1}, it follows that $goal_{rev}(S_{n+1} \cap \mathcal{F}_{rev})$ holds. By construction of tr_{rev}, it follows $S^\star_0 \equiv S^\star_{n+1}$. Hence, the last conjunct of (9) is true.

In total, we have that for any $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$ where S_0 and S_i are from a trajectory of P with respect to P^c_{rev}, formula (10) holds. For all other values for $\mathcal{F}$ and $\mathcal{F}'$ the antecedent of (9) does not hold. Hence, (10) holds for all $\mathcal{F}$ and $\mathcal{F}'$. Therefore $\langle A_i, \ldots, A_{n-1}\rangle$ is a $\phi;\psi$-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D. □

Proof of Theorem 13. If $\langle A_i, \ldots, A_{n-1}\rangle$ is an effective $\phi;\psi$-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D, then some state S_i is reachable in some execution of $\langle A_0, \ldots, A_{i-1}\rangle$ for a state S_0 such that $S_0 \models \psi(S_0)$ and such that $\phi(S_i)$ holds, and by Theorem 12 $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$ is a conformant plan of P^c_{rev}. Hence, the latter has a trajectory $S_0, A_0, \ldots, A_{n-1}, S_{n+1}$ such that *normal* is true at each state S_j, $0 \leqslant j \leqslant n+1$. By definition of $P^c_{rev}{}'$, P is thus a plan for $P^c_{rev}{}'$. This proves the only if direction.

For the if direction, by Theorem 12 $\langle A_i, \ldots, A_{n-1}\rangle$ is a $\phi;\psi$-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D. Since $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$ is a plan for $P^c_{rev}{}'$, by construction P^c_{rev} has a trajectory $S_0, A_0, \ldots, A_{n-1}, S_{n+1}$ such that *normal* is true at each state S_j, $0 \leqslant j \leqslant n+1$. Hence, $\phi(S_i \cap \mathcal{F})$ and $\psi(S_i \cap \tilde{\mathcal{F}})$ hold. Consequently, some execution of $\langle A_0, \ldots, A_{i-1}\rangle$ exists which results in a state S_i such that $\phi(S_i)$ holds. This shows that $\langle A_i, \ldots, A_{n-1}\rangle$ is effective. □

Appendix C. Proofs for Section 7

Proof of Theorem 14. The proof is similar to the proof of Theorem 16. □

Proof of Theorem 16. (i) First observe that, upon calling REVERSE1(i) in Algorithm REVERSE(AS, Π, L), where $AS = \langle A_0, \ldots, A_{i-1}\rangle$,

(O1) line (06) of REVERSE1(i) is executed at most $|AS| = i$ times in total, and thus each action in $A_0, \ldots, A_{i-1}$ is covered by exactly one undo;

(O2) the total number of recursive calls of REVERSE1(j), $j < i$, is at most i.

According to a trie data structure representation of the reverse plan library L, as described in Section 7, for a given action sequence $\langle A_0, \ldots, A_j \rangle$ all matching suffixes As in L (and all associated tuples (As, R, ϕ, ψ)) can be determined in time $O(\min(AS_{\max}(L), |AS|))$. Then, the tuples (AS, R, ϕ, ψ) in the for-loop of REVERSE1 (line (02)) can be found in $O(\min(AS_{\max}(L), |AS|))$ time. For each tuple, checking whether $\psi_j \supset phi$ and $psi_{j-|AS|} \supset \psi$ is feasible in $eval_{\max}(L)$ time; hence, the total time for such checks is bounded by $|L| \cdot eval_{\max}(L)$. Hence, modulo recursive calls of REVERSE1, the body of REVERSE1 takes $O(|L| \cdot eval_{\max}(L) + \min(AS_{\max}(L), |AS|))$ time.

On the other hand, REVERSE1(0) takes constant time. Then, due to (O1) and (O2), REVERSE1(i) takes in total $O(i(|L| \cdot eval_{\max}(L) + \min(AS_{\max}(L), |AS|)))$ time.

(ii) If R_i is a $\phi_j;\psi_j$-reverse plan for an action sequence AS_j and sequence of percepts $\Pi_j = \pi_{j,0}, \ldots, \pi_{j,|AS_j|}$, for $j \in \{1, 2\}$, then $R_2 + R_1$ is a ϕ_2-reverse plan for $AS = AS_1 + AS_2$ and $\Pi = \Pi_1 + \Pi_2$ if $\psi_{2,|AS_2|} \supset \phi_2$ holds and $\psi_{2,0} \supset \phi_1$ hold. Thus, the algorithm REVERSE is clearly sound, i.e., produces only correct reverse plans for AS and Π relative to L.

On the other hand, REVERSE is also complete and finds some $\phi;\psi$-reverse plan for AS and Π relative to L, if one exists. Indeed, whenever R_1'' and R_2'' are $\phi_1'';\psi$- and $\phi_2'';\psi$-reverse plans, respectively, for the same action sequence AS'' and percept sequence Π'' relative to L, then they can both serve the role of R'' in Definition 7.1 and be composed with any R' as described there to a $\phi;\psi$-reverse plan for AS and Π relative to L. Therefore, by induction on the length $|AS| \geqslant 1$ of the sequence to be reversed, one can easily show that REVERSE(AS, Π, L) outputs some $\phi;\psi$-reverse plan R for AS and Π relative to L, if one exists. □

References

[1] S. Abramsky, A structural approach to reversible computation, Theoret. Comput. Sci. 347 (3) (2005) 441–464.
[2] A. Agostini, G.D. Michelis, M. Loregian, Undo in workflow management systems, in: [40], pp. 321–335.
[3] H. Agrawal, R.A. DeMillo, E.H. Spafford, An execution backtracking approach to program debugging, IEEE Software 8 (3) (1991) 21–26.
[4] J.E. Allchin, M.S. McKendry, Synchronization and recovery of actions, SIGOPS Oper. Syst. Rev. 19 (1) (1985) 32–45.
[5] C. Baral, V. Kreinovich, R. Trejo, Computational complexity of planning and approximate planning in the presence of incompleteness, Artificial Intelligence 122 (1–2) (2000) 241–267.
[6] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison–Wesley, 1987, 363 pp.
[7] M. Cadoli, F.M. Donini, A survey on knowledge compilation, AI Comm. 10 (3–4) (1997) 137–150.
[8] R.B. Dannenberg, A structure for efficient update, incremental redisplay and undo in graphical editors, Softw. Pract. Exper. 20 (2) (1990) 109–132.
[9] G. De Giacomo, R. Reiter, M. Soutchanski, Execution monitoring of high-level robot programs, in: Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), 1998, pp. 453–465.
[10] T. Eiter, E. Erdem, W. Faber, Diagnosing plan execution discrepancies in a logic-based action framework, Technical Report INFSYS RR-1843-04-03, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, August 2004.
[11] T. Eiter, E. Erdem, W. Faber, Undoing the effects of action sequences, Technical Report INFSYS RR-1843-04-05, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, December 2004.
[12] T. Eiter, E. Erdem, W. Faber, On reversing actions: Algorithms and complexity, in: M.M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, January 2007, pp. 336–341.
[13] T. Eiter, E. Erdem, W. Faber, J. Senko, A logic-based approach to finding explanations for discrepancies in optimistic plan execution, Fundamenta Informaticae (2007), in press. Preliminary report [10].
[14] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning, II: The DLV$^{\mathcal{K}}$ system, Artificial Intelligence 144 (1–2) (2002) 157–211.
[15] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning: Semantics and complexity, ACM TOCL 5 (2) (2004) 206–263.
[16] P. Ferraris, E. Giunchiglia, Planning as satisfiability in nondeterministic domains, in: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00), July 30–August 3, 2000, Austin, TX, USA, AAAI Press/The MIT Press, 2000, pp. 748–753.
[17] M. Fichtner, A. Großmann, M. Thielscher, Intelligent execution monitoring in dynamic environments, in: Proc. of IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating, 2003.
[18] M. Fichtner, A. Großmann, M. Thielscher, Intelligent execution monitoring in dynamic environments, Fundamenta Informaticae 57 (2–4) (2003).
[19] M. Gelfond, V. Lifschitz, Action languages, Electron. Trans. Artif. Intell. 2 (3–4) (1998) 193–210.
[20] E. Giunchiglia, Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism, in: A.G. Cohn, F. Giunchiglia, B. Selman (Eds.), Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), April 12–15, Breckenridge, CO, USA, Morgan Kaufmann, 2000, pp. 657–666.
[21] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, H. Turner, Nonmonotonic causal theories, Artificial Intelligence 153 (1–2) (2004) 49–104.

[22] R. Goldman, M. Boddy, Expressive planning and explicit knowledge, in: Proc. AIPS-96, 1996, pp. 110–117.
[23] H. Hayashi, K. Cho, A. Ohsuga, Mobile agents and logic programming, in: N. Suri (Ed.), Proceedings of the Sixth International Conference on Mobile Agents (MA 2002), Barcelona, Spain, October 22–25, 2002, in: Lecture Notes in Computer Science, vol. 2535, Springer, 2002, pp. 32–46.
[24] H. Hayashi, K. Cho, A. Ohsuga, A new HTN planning framework for agents in dynamic environments, in: J. Dix, J.A. Leite (Eds.), Post-Proceedings of the 4th International Workshop on Computational Logic and Multi-Agent Systems (CLIMA IV), Fort Lauderdale, FL, USA, January 6–7, 2004, in: Lecture Notes in Computer Science, vol. 3259, Springer, 2004, pp. 108–133.
[25] D.S. Johnson, A catalog of complexity classes, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. A, Elsevier Science, 1990. Chapter 2.
[26] G.B. Leeman Jr, A formal approach to undo operations in programming languages, ACM Trans. Program. Lang. Syst. 8 (1) (1986) 50–87.
[27] P. Liberatore, The complexity of the language A, Electron. Trans. Artif. Intell. 1 (1997) 13–38.
[28] I. Mackie, The geometry of interaction machine, in: POPL, 1995, pp. 198–208.
[29] N. McCain, H. Turner, Satisfiability planning with causal theories, in: A.G. Cohn, L. Schubert, S.C. Shapiro (Eds.), Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Morgan Kaufmann, 1998, pp. 212–223.
[30] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[31] J. Rintanen, Constructing conditional plans by a theorem-prover, J. Artificial Intelligence Res. 10 (1999) 323–352.
[32] S.A. Sloman, D.A. Lagnado, Do we "do"?, Cognitive Sci. 29 (2005) 2–39.
[33] D.E. Smith, D.S. Weld, Conformant graphplan, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98), AAAI Press/The MIT Press, July 1998, pp. 889–896.
[34] T.C. Son, C. Baral, Formalizing sensing actions—a transition function based approach, Artificial Intelligence 125 (1–2) (2001) 19–91.
[35] M. Soutchanski, Execution monitoring of high-level temporal programs, in: Proc. of the IJCAI'99 Workshop on Robot Action Planning, 1999.
[36] M. Soutchanski, High-level robot programming and program execution, in: Proc. of ICAPS'03 Workshop on Plan Execution, 2003.
[37] C. Sun, Undo as concurrent inverse in group editors, ACM Trans. Comput.-Hum. Interact. 9 (4) (2002) 309–361.
[38] H. Turner, Polynomial-length planning spans the polynomial hierarchy, in: Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02), in: Lecture Notes in Computer Science, vol. 2424, Springer, 2002, pp. 111–124.
[39] J.D. Ullman, Principles of Database and Knowledge Base Systems, Computer Science Press, 1989.
[40] W.M.P. van der Aalst, A.H.M. ter Hofstede, M. Weske (Eds.), Business Process Management, International Conference, BPM 2003, Proceedings, Eindhoven, The Netherlands, June 26–27, 2003, Lecture Notes in Computer Science, vol. 2678, Springer, 2003.
[41] M.V. Zelkowitz, Reversible execution, Comm. ACM 16 (9) (1973) 566.