I N F S Y S
RESEARCH
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

UNDOING THE EFFECTS OF ACTION SEQUENCES

Thomas Eiter Esra Erdem Wolfgang Faber

INFSYS RESEARCH REPORT 1843-04-05
2004

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-04-05, 2004

UNDOING THE EFFECTS OF ACTION SEQUENCES

Thomas Eiter[1] Esra Erdem[2] Wolfgang Faber[3]

Abstract. When an agent has executed a single action or a sequence of actions, it may sometimes be desirable that the effects of this execution are undone and the agent gets back to the previous state. A prototypical scenario for this is in the context of plan execution in a nondeterministic environment. Upon detecting that after some steps an unintended state has been reached, backtracking by taking appropriate undo actions can be useful for error recovery. In this paper, we thus consider the problem of undoing the effects of an action sequence by means of a reverse plan, in a general logic-based action representation framework that can accommodate nondeterminism and concurrency. Intuitively, by executing a reverse plan for an action sequence AS at the state S reached after AS, she can always reach the state S she was at before executing AS. We formally define the notion of a reverse plan in terms of a logical specification and determine the computational complexity of the major reasoning tasks on it, namely the existence and recognition problem. Guided by these results, we then present algorithms for constructing reverse plans. Since this is intractable in general, we develop a knowledge compilation approach in which a reverse plan library is built offline by finding reverse plans for action sequences, and then a reverse plan for a given action sequence is efficiently constructed online using this library. For the former part, we describe methods based on conformant planning; for the latter, we present a polynomial time algorithm.

[1]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: eiter@kr.tuwien.ac.at

[2]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: esra@kr.tuwien.ac.at

[3]Mathematics Department, University of Calabria, 87030 Rende, Italy. E-mail: faber@mat.unical.it

Acknowledgements: We thank Mikhail Soutchanski for useful discussions. 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.

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., [11, 14, 21, 6]), and various reasoning problems about actions have been considered. The most prominent among them are temporal projection (inference about the 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).

This paper studies 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. An example application for this problem is plan execution in a nondeterministic environment. In particular, undoing the effects of an execution of an action sequence can be used to switch from one plan to another one (e.g., in the light of some new information, another plan, when executed at the initial state, may lead to the goal in less number of steps), to restart a plan (e.g. when the execution of the plan fails due to some undesired effects of some action), or to backtrack the execution of a plan until some point. Thus, it can be useful to recover from or to prevent plan failures, or to switch from a plan to a "better" one.

For instance, when a mobile agent moves from one computer to another, the agent may not execute an action as planned due to changes in environment. To recover from such a plan failure, the agent may try to switch from one plan to another plan in her repository. However, some of the executed actions may have effects that prevent the execution of other plans in the new environment. In such cases, as proposed in [15, 16], the agent can undo the effects of the plan executed so far, and then start executing another plan in her repository.

For another example, consider an agent situated in a nondeterministic environment, where the plan execution is monitored. Execution monitoring may help to reveal that things go wrong and to recover from any detected execution failure. Suppose that the monitoring agent detects a discrepancy between the actual state and the expected states of the world. Such a discrepancy usually implies a failure (at least in the agent's belief), for which a recovery should be sought. Suppose that the agent finds a diagnosis (a point of failure) of the discrepancy, as in [4]. One way to recover from the detected discrepancies, as proposed in [5], is to undo the plan execution until the diagnosed point of failure, and then to execute the remainder of the plan.

To tackle the problem of undoing the effects of actions, we consider a general logic-based framework as in [24]. 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. This framework can accommodate nondeterminism, concurrent actions and dynamic worlds. Such action representations can be obtained from domain descriptions in STRIPS-based languages or in more expressive action description languages, such as $\mathcal{C}+$ [14] or $\mathcal{K}$ [6]. This allows us to use systems like CCALC [18, 14], CPLAN [13, 8], and DLV$^{\mathcal{K}}$ [6] for reasoning about actions.

In this framework, the effects of an execution of an action sequence can be undone by executing one of its "reverse plans." 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).

The main contributions of this paper are as follows.

(1) We precisely describe the notions of a reverse action and a reverse plan for actions in the logic-based framework of [24]. By definition, an undo action and a reverse plan can be executed at any state. This is useful in particular when the agent can not fully observe the environment or when she finds herself at an unexpected state, and thus has difficulties to figure out what action to take for undoing the effects of some action. The notions of a reverse action and a reverse plan are extended to multi-step reversals, where the effects of an execution of an action sequence is undone, and to conditional reversals, where the effects of an execution of an action sequence are undone subject to some conditions on the current state.

(2) Due to their logic-based definitions, reverse plans for actions sequences can be readily computed, following Rintanen's approach [20], 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. Deciding the existence of conformant plans in the framework of [24] is Σ_3^p-complete, and thus of the same degree of complexity as computing a reverse plan for a given action sequence. For efficient online computation, we present a low-order polynomial time algorithm which assembles a reverse plan for an action sequence from a given reverse library. Under reasonable assumptions, it runs in linear time in the worst case.

(3) 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).

Undo actions and reverse plans, to our knowledge, have not been considered in greater detail for plan execution and recovery in a nondeterministic environment. In [15, 16], the authors use undo actions for execution of plans by mobile agents in a dynamic environment. This work illustrates the usefulness of undo actions. However, undo actions (one for each action) need to be specified (manually) by the user in advance. In this sense, our method of computing reverse plans is more general. On the other hand, the existing logic-based execution monitoring frameworks [12, 22, 23, 9, 10] do not consider undo actions (or reverse plans) for recovering from discrepancies. In this sense, our method of using execution reversals for recovery is complementary to the existing recovery techniques.

In the following, we first present the action representation framework and the planning framework we consider (Section 2). Then, we precisely describe the reverse of an action and a plan, and extend these definitions to handle various cases (Section 3), and 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 offline reversals, and how to exploit this library of reversals online for plan recovery (Section 6). We conclude with a discussion of the related work (Section 7) and future work (Section 8).

2 Preliminaries

We consider the action representation and planning framework described in [24], which is similar to that of [20].

2.1 Action representation framework

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

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

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

(i) the start state corresponds to an interpretation of (the symbols in) $\mathcal{F}$,[1]

(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 1 [14] 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

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

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: $\{inWater, wet\}, \{wet\}, \{\}$.

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

The action occurrences can be defined as follows:

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

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:

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

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

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

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

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 is the action occurrences that correspond to the interpretation of $\mathcal{A}_i$.

Example 2 The transition diagram for the action description of Example 1 is presented in Figure 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

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

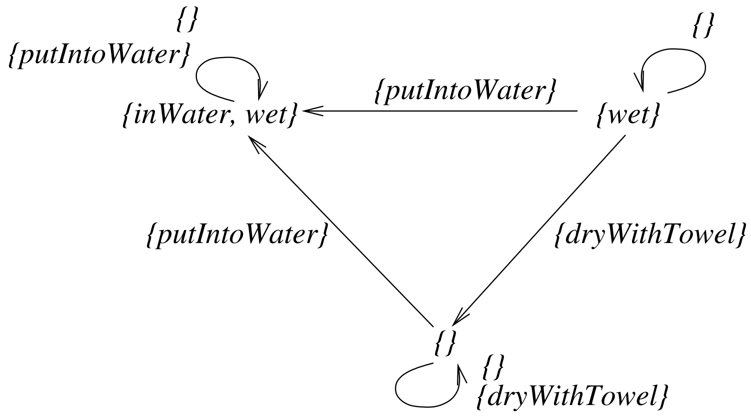


Figure 1: The transition diagram for the action description of Example 1.

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 $tr(\mathcal{F}_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, \mathcal{A}_1, \mathcal{F}_2)$.$\Box$

We can talk about more specific states, transitions, or trajectories by applying some "substitutions" to the formulas describing them. Consider, for instance, Example 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 by replacing every atom $p \in \mathcal{F}$ with $\perp$ 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 by replacing every atom $p_0 \in \mathcal{F}_0$ with $\perp$ 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 by replacing every atom $p_1 \in \mathcal{A}_1$ with $\perp$ 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 $tr(S_0, \mathcal{A}_0, \mathcal{F}_1) \wedge tr(\mathcal{F}_1, A_1, \mathcal{F}_2) \wedge tr(\mathcal{F}_2, A_2, S_3)$ can be replaced by $tr_3(S_0 \cup S_3, A_1 \cup A_2)$.

2.2 Planning framework

In a planning problem, an initial state is described by the formula $init(\mathcal{F})$ such that $init(\mathcal{F}) \models state(\mathcal{F})$, and the goal is described by the formula $goal(\mathcal{F})$.

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

$$\mathit{init}(\mathcal{F}_0) \wedge \bigwedge_{t=0}^{n-1} \mathit{tr}(\mathcal{F}_t, \mathcal{A}_t, \mathcal{F}_{t+1}) \wedge \mathit{goal}(\mathcal{F}_n)$$

as the sequence

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

of action occurrences A_i that correspond to the interpretation of $\mathcal{A}_i$.

Example 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., $\mathit{state}(\mathcal{F})$, can be defined by the conjunction of the following formulas:[2]

- 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:

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

- 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 $on(B, L)$):

$$supported(B).$$

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

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

$$throw(B, L, L1) \supset (on(B, L) \wedge \bigwedge_{B1} \neg on(B1, B)); \tag{3}$$

[2]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 $table$.

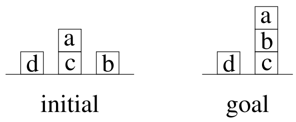
- the effects of $throw(B, L, L1)$, and inertia:

$$on(B, L1)' \supset (on(B, L1) \vee \bigvee_{L,L2} throw(B, L, L2)); \tag{4}$$

- no-concurrency:

$$throw(B, L, L1) \supset \bigwedge_{B1 \neq B, L2, L3} \neg throw(B1, L2, L3). \tag{5}$$

Consider, in this domain, the blocks a, b, c, d, and a planning problem $\mathcal{P}$ with the initial state and the goal state as follows:



A solution to the planning problem $\mathcal{P}$ is the following plan of length 3:[3]

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

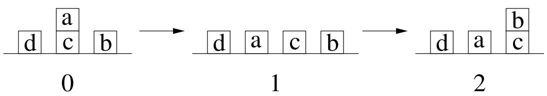
□

A *conformant plan* 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

$$\begin{array}{l} \forall \mathcal{F}_0, \ldots, \mathcal{F}_n \exists \mathcal{F}'_0, \ldots, \mathcal{F}'_n \;\; init(\mathcal{F}'_0) \\ \quad \wedge \bigwedge_{t=0}^{n-1} \left(init(\mathcal{F}_0) \wedge tr_t(\mathcal{F}, A) \supset tr(\mathcal{F}_t, A_t, \mathcal{F}'_{t+1}) \right) \\ \quad \wedge \left(init(\mathcal{F}_0) \wedge tr_n(\mathcal{F}, A) \supset goal(\mathcal{F}_n) \right). \end{array} \tag{6}$$

The formula above 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.

Example 4 Consider the blocks world domain, and the planning problem $\mathcal{P}$ described in Example 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



[3]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 $throw(B, L, L1)$.

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 (3) with the formula below describes the preconditions of actions:

$$carry(B, L) \supset \bigwedge_{B1} \neg on(B1, B), \tag{7}$$

and the conjunction of the following with (5) describes no-concurrency:

$$\begin{array}{l} (carry(B, L) \supset \\ \quad (\bigwedge_{B1,L2,L3} \neg throw(B1, L2, L3) \ \wedge \bigwedge_{B1 \neq B, L2} \neg carry(B1, L2))). \end{array} \tag{8}$$

The effects of actions, and inertia are defined, instead of (4), with the conjunction:

$$\begin{array}{l} (carry(B, L) \supset on(B, L)') \wedge \\ (on(B, L1)' \supset (on(B, L1) \vee carry(B, L1) \vee \bigvee_{L,L2} throw(B, L, L2))). \end{array} \tag{9}$$

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

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

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

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

3 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 are 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 executing a plan $\langle A_0, \ldots, A_{n-1} \rangle$ 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 (e.g. the rest of the plan fails to achieve the goal) and wants to go back to an earlier state S_i ($i < j$) so that she can try something else to reach the goal. One way to reach the earlier state S_i is to solve 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. Another way to reach S_i is to undo the sequence $\langle A_{i+1}, \ldots, A_{j-1} \rangle$ of actions at S_j. In the following, we will focus on the latter kind of backtracking.

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

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.

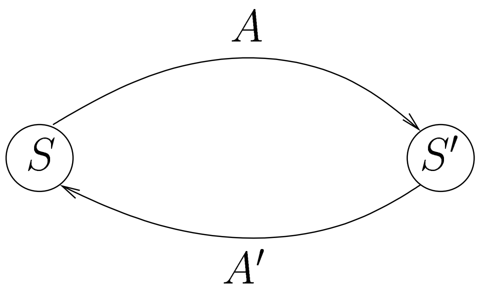


Figure 2: Action A' is a reverse action for action A.

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

$$\begin{aligned} &revAct(\mathcal{F}, \mathcal{F}'; A, A') = \\ &\quad tr(\mathcal{F}, A, \mathcal{F}') \supset (tr(\mathcal{F}', A', \mathcal{F}) \wedge \forall \mathcal{F}'' \, (tr(\mathcal{F}', A', \mathcal{F}'') \supset \mathcal{F} \equiv \mathcal{F}'')) \end{aligned} \tag{10}$$

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 Figure 2.)

Example 5 In the setting of Example 4, $carry(B, L)$ 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., (3), 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., (9), 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., (7), we can execute $carry(B, L)$ at state S'. Due to the deterministic effect of $carry(B, L)$ and due to inertia, i.e., (9), 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, $S'' = S$.□

Definition 2 *A* reverse plan *for an action A is a sequence $\langle A_0, \ldots, A_{m-1} \rangle$ ($m \geq 0$) of actions such that, for all $\mathcal{F}$ and $\mathcal{F}'$, the formula*

$$\begin{aligned} &revPlan(\mathcal{F}, \mathcal{F}'; A, [A_0, \ldots, A_{m-1}]) = \\ &\quad tr(\mathcal{F}, A, \mathcal{F}') \supset \forall \mathcal{F}_0, \ldots, \mathcal{F}_m \exists \mathcal{F}'_1, \ldots, \mathcal{F}'_m (\mathcal{F}_0 \equiv \mathcal{F}' \supset \\ &\qquad\qquad (\textstyle\bigwedge_{t=0}^{m-1} (tr_t(\mathcal{F}, A) \supset tr(\mathcal{F}_t, A_t, \mathcal{F}'_{t+1})) \wedge \\ &\qquad\qquad (tr_m(\mathcal{F}, A) \supset \mathcal{F}_m \equiv \mathcal{F}))) \end{aligned} \tag{11}$$

holds.

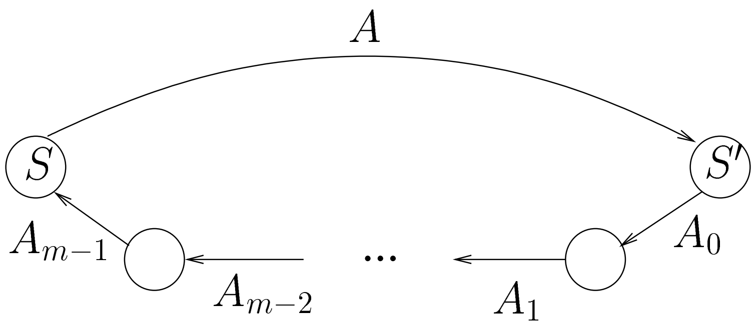


Figure 3: Action sequence $\langle A_0, \ldots, A_{m-1}\rangle$ is a reverse plan for action A.

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'. If the action sequence $\langle A_0, \ldots, A_{m-1}\rangle$ is executable at state S', then it always leads to S. (See Figure 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 6).

Definition 3 *A sequence* $\langle A'_0, \ldots, A'_{m-1}\rangle$ *(*$m \geq 0$*) 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 formula*

$$\begin{array}{l} multiRev(\mathcal{F}, \mathcal{F}'; [A_0, \ldots, A_{k-1}], [A'_0, \ldots, A'_{m-1}]) = \\ \quad \exists \mathcal{F}_0, \ldots, \mathcal{F}_k\ (\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A)\ \wedge \mathcal{F}' \equiv \mathcal{F}_k)\ \supset \\ \qquad \forall \mathcal{F}'_0, \ldots, \mathcal{F}'_m \exists \mathcal{F}''_1, \ldots, \mathcal{F}''_m (\ \mathcal{F}'_0 \equiv \mathcal{F}' \supset \\ \qquad\quad \bigwedge_{t=0}^{m-1} \Big(tr_t(\mathcal{F}', A') \supset tr(\mathcal{F}'_t, A'_t, \mathcal{F}''_{t+1})\Big) \wedge \\ \qquad\quad (tr_m(\mathcal{F}', A') \supset \mathcal{F}'_m \equiv \mathcal{F})) \end{array} \tag{12}$$

is true.

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 Figure 4.) Note that $multiRev(\mathcal{F}, \mathcal{F}'; [A_0], [A'_0, \ldots, A'_m])$ is equivalent to $revPlan(\mathcal{F}, \mathcal{F}'; A_0, [A'_0, \ldots, A'_m])$.

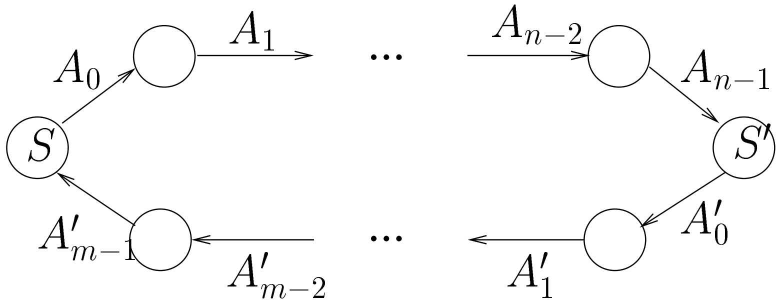


Figure 4: Action sequence $\langle A'_0, \ldots, A'_{m-1} \rangle$ is a reverse plan for action sequence $\langle A_0, \ldots, A_{n-1} \rangle$.

Example 6 In the setting of Example 4, a reverse plan for the action sequence

$$\langle \mathit{throw}(B1, L1, L2), \mathit{carry}(B1, L3), \mathit{throw}(B1, L3, L4), \mathit{carry}(B1, L5) \rangle$$

is $\langle \mathit{carry}(B1, L1) \rangle$. 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 reachable state. 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 conditions that do not hold at every reachable state. To make execution reversals applicable in such cases, we extend the concept of a reverse plan to a "conditional reverse plan" that takes state information into account.

Definition 4 *A sequence* $\langle A'_0, \ldots, A'_{m-1} \rangle$ *(*$m \geq 0$*) of actions is a* ϕ-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*

$$\phi(\mathcal{F}') \supset \mathit{multiRev}(\mathcal{F}, \mathcal{F}'; [A_0, \ldots, A_{k-1}], [A'_0, \ldots, A'_{m-1}]) \tag{13}$$

is true. Here $\phi(\mathcal{F}')$ *is a formula with all nonlogical symbols in* $\mathcal{F}'$*.*

Example 7 Consider a variant of Example 4 with another deterministic action of carrying a block B from a location L to another location $L1$: $\mathit{carry2}(B, L, L1)$. Note that $\langle \mathit{carry2}(B, L4, L1) \rangle$ is not a reverse plan for the action sequence

$$\langle \mathit{throw}(B, L1, L2), \mathit{carry}(B, L3), \mathit{throw}(B, L3, L4) \rangle,$$

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

4 Execution Monitoring

As already pointed out in the previous section, a prototypical application of execution reversal is execution monitoring. We briefly discuss and illustrate in this section how reverse plans can be fruitfully used in this context.

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.

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, 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.

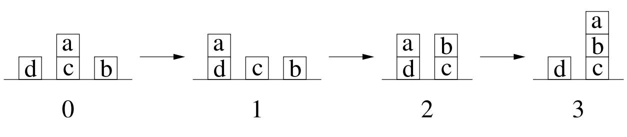
Steps 1–3 of the framework above are presented in Figure 5 as well.

Here is an example describing the framework above.

Example 8 Consider the blocks world described in Examples 3 and 4. Suppose that an agent is executing the plan P, i.e.,

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

and that she is given the following intended trajectory T_P:



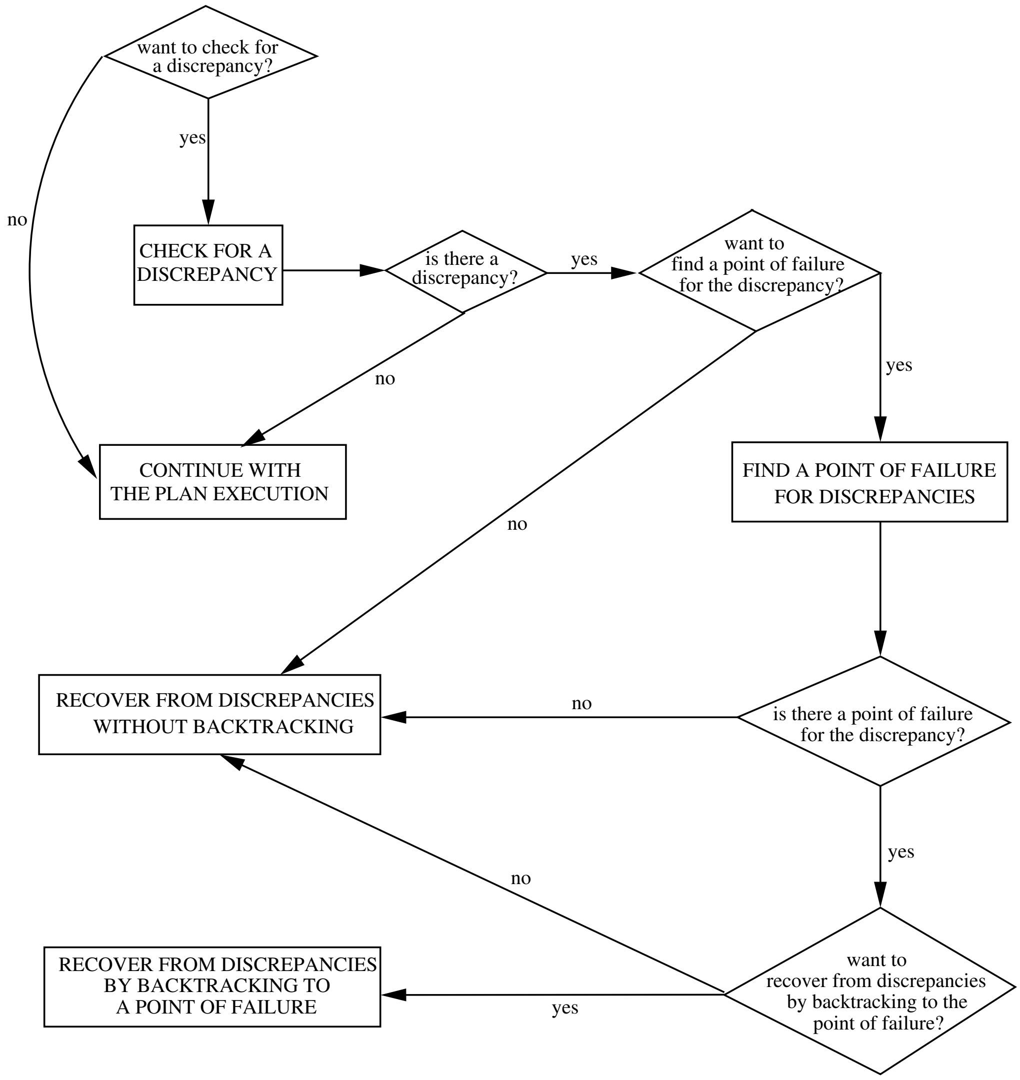
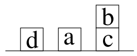


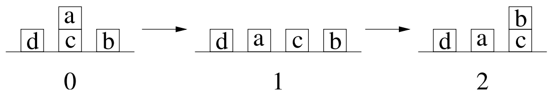
Figure 5: Steps 1–3 of the monitoring framework described in Section 4.

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 $throw(a, c, d)$.

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 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. $\Box$

As seen in Figure 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 [12, 22, 23], 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 [4]. In the following, we will concentrate on how backtracking to a diagnosed point of failure can be done, like in Example 8, and study this problem in the formal framework described in Section 2. In particular, we will study backtracking by plan reversals avoiding (re)planning.

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

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

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 a given formula ϕ composed of fluent symbols, recognizing whether R is a ϕ-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 a formula ϕ composed of fluent symbols such that R is a ϕ-reverse plan for AS, and that $\phi(S)$ holds for some state S reached by AS;

(P3) for a given action sequence AS, and a given formula ϕ composed of fluent symbols, deciding whether there exists an action sequence R of a polynomially bounded length such that R is a ϕ-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 $k \geq 1$, $\Sigma_k^p = \mathrm{NP}^{\Sigma_{k-1}^p}$ and $\Pi_k^p = \mathrm{coNP}^{\Sigma_{k-1}^p}$. At each level k ($k \geq 1$) of the hierarchy, the complexity classes Σ_k^p and Π_k^p are distinct, and they include all complexity classes at lower levels.

Several problems on planning and reasoning about actions which are complete for these classes for $k \leq 3$, can be found in [2, 7, 20, 24]. In the first level of this hierarchy, $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \mathrm{coNP}$. Then, Σ_2^p-complete problems (resp. Σ_3^p-complete problems) are NP-complete even if an NP oracle (resp. a Σ_2^p oracle) for solving problems at no cost is available, and are thus rather difficult to solve. Similarly, Π_2^p-complete problems (resp. Π_3^p-complete problems) are coNP-complete even if an NP oracle (resp. Σ_2^p oracle) for solving problems at no cost is available.

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 [17, 19].

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 ϕ-reverse plan for another action sequence (i.e., (P1)) is easier than finding a ϕ-reverse plan for an action sequence (i.e., (P2) and (P3)). Finding a ϕ-reverse plan, where ϕ is given is harder than finding a ϕ-reverse plan for an arbitrary ϕ 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 $|R| \geq 2$, 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 polynomial-length bounded action sequences R and in case of (P2), an exponential number of formulas ϕ which need to be considered, (ii) the test that for all states S and S' such that $\phi(S')$ holds 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.

In problems (P1) and (P3), we do not check that the formula ϕ is actually satisfied at some state S reached after execution of AS (in this case, 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 is a formula from a polynomial size set of formulas, then only a polynomial number of candidates for ϕ-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 is a conjunction of literals which has 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-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. polynomial time. 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 $\phi(\mathcal{X})$ denotes a propositional combination 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-reverse action for an action:

Theorem 1 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, two actions A and A' in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether A' is a ϕ-reverse action for A is* coNP*-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$ and $A' = A$.*

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

Theorem 2 *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 a formula $\phi(\mathcal{F})$, deciding whether $\langle A \rangle$ is a ϕ-reverse plan for AS is* coNP*-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$ and $|AS| = 1$.*

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

Theorem 3 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, two action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether R is a ϕ-reverse plan for AS is Π_2^p-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$ and, moreover, $|R| \geq 2$ 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-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.

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

Theorem 4 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action A in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and an action A' in $2^{\mathcal{A}}$ such that A' is an effective ϕ-reverse action for A is Σ_2^p-complete.*

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

Theorem 5 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and a sequence R of at most two actions in $2^{\mathcal{A}}$ such that R is an effective ϕ-reverse plan for AS is Σ_2^p-complete. The Σ_2^p-hardness holds even if $|AS| = 1$.*

Intuitively, the reason is that in general a formula $\phi(\mathcal{F})$ can be considered which limits 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 a formula $\phi(\mathcal{F})$ and a ϕ-reverse plan R (of polynomial length) for an action sequence AS, the complexity increases by one level in the polynomial hierarchy:

Theorem 6 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and a sequence R of polynomial number of actions in $2^{\mathcal{A}}$ such that R is an effective ϕ-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-reverse plan R for an action sequences AS where AS and ϕ are given. For R containing a single action A, the problem is of the same difficulty as (P2).

Theorem 7 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether there exists a ϕ-reverse plan R for AS such that $|R| = 1$ is Σ_2^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$ 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 8 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether there exists a ϕ-reverse plan R for AS is Σ_3^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$ and $|R|$ is fixed to 2.*

6 Computation of Reverse Plans

We compute reverse plans in the spirit of knowledge compilation [3]: 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 6.2, 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 5 *A* reverse plan item (RPI) *is a tuple of the form* (AS, R, ϕ) *such that* R *is a* ϕ*-reverse plan for the (nonempty) action sequence* AS*, where* $\phi = \phi(\mathcal{F})$*. An RPI is* single-step*, if* $|AS| = 1$*, i.e.,* AS *consists of a single action, and* unconditional*, if* $\phi = true$*.*

Definition 6 *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 suggestive by the fact that, 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 6.2).

6.1 Offline RPI computation

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) by solving quantified Boolean formulas (QBFs), which has been proposed as a problem solving method in the planning domain earlier, e.g. [20].

Another possibility is to reduce the problem to 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 planning system, of which some are based on QBFs, can be used.

6.1.1 Basic transformation to conformant planning

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

$$tr_{rev}(\mathcal{F}_{rev}, \mathcal{A}, \mathcal{F}'_{rev}) = (tr(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge \tilde{\mathcal{F}} \equiv \tilde{\mathcal{F}}'). \quad (14)$$

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 $\tilde{\mathcal{F}}$:

$$init_{rev}(\mathcal{F}_{rev}) = (state(\mathcal{F}) \wedge \mathcal{F} \equiv \tilde{\mathcal{F}}). \tag{15}$$

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 $\mathcal{F}$):

$$goal_{rev}(\mathcal{F}_{rev}) = (\mathcal{F} \equiv \tilde{\mathcal{F}}). \tag{16}$$

Any conformant plan for P_{rev} of length ≥ 1 represents reverse plans:

Theorem 9 *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* ϕ-reverse plans for a given action sequence and formula ϕ.

6.1.2 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 need to 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. With this enhancement, we are in the position to compute also conditional RPIs (for a given formula ϕ, as in (P3)).

This modification of P_{rev} will be referred to as $P_{rev,\phi}$, and we will next state and describe it in more detail.

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):

$$\mathit{init}_{rev,\phi}(\mathcal{F}_{rev,\phi}) = \mathit{init}_{rev}(\mathcal{F}_{rev}) \wedge normal \wedge \neg sep\_occ$$

The transition diagram is described by the following formula:

$$
\begin{aligned}
& \mathit{tr}_{rev,\phi}(\mathcal{F}_{rev,\phi}, \mathcal{A}_\phi, \mathcal{F}'_{rev,\phi}) = && (17)\\
& \quad normal \supset (&& \\
& \qquad ((\neg sep\_occ \wedge \neg normal') \vee && (18)\\
& \qquad\; (\neg sep \supset (normal' \wedge \mathit{tr}_{rev}(\mathcal{F}_{rev}, \mathcal{A}, \mathcal{F}'_{rev})))) \wedge && (19)\\
& \qquad (sep \supset (\phi(\mathcal{F}) \equiv normal')) \wedge && (20)\\
& \qquad (sep \supset \bigwedge_{a \in \mathcal{A}} \neg a)) \wedge && (21)\\
& \qquad (sep \supset (\mathcal{F}_{rev} \equiv \mathcal{F}'_{rev})) \wedge && (22)\\
& \qquad (sep \supset (\neg sep\_occ \wedge sep\_occ')) \wedge && (23)\\
& \qquad (sep\_occ' \equiv (sep\_occ \vee sep)) \;\;) \wedge && (24)\\
& \quad (\neg normal \supset \neg normal') && (25)
\end{aligned}
$$

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 (25).

In "normal" mode, we can either switch to "don't care" mode, provided that we are in the "forward" phase by (18), or continue in "normal" mode according to $\mathit{tr}_{rev}(\mathcal{F}_{rev}, \mathcal{A}, \mathcal{F}'_{rev})$, provided that we are not in the "separating" transition (sep does not hold, (19)).

If we are in the "separating" transition (sep holds), we first test whether $\phi(\mathcal{F})$ holds. If it does, we continue in "normal" mode, if it does not, we change to "don't care" mode, by means of (20).

Furthermore, we assess that no other action occurs together with sep by (21). 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 (22). Note that copying the truth value of $normal$ is not needed. If it was false, it remains false by (25), if it was true, it is determined by the validity of ϕ by (20). We then assert that sep_occ did not hold before the "separating" transition, and that it holds after it by (23). Finally, by (24) 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:

$$goal_{rev,\phi}(\mathcal{F}_{rev,\phi}) = (goal_{rev}(\mathcal{F}_{rev}) \wedge sep\_occ) \vee \neg normal$$

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. R is therefore a ϕ-reverse plan for AS. Moreover, we can also show that any ϕ-reverse plan for a given AS can be found in this way (for a given ϕ).

More formally, the following statement holds.

Theorem 10 *Let $P_{rev,\phi}$ be the planning problem defined relative to the action description $D_{rev,\phi}$. Then, $\langle A_i, \ldots, A_{n-1}\rangle$ is a ϕ-reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geq i > 0$, is a conformant plan for $P_{rev,\phi}$.*

Therefore, we can generate all ϕ-reverse plans by solving the conformant planning problem $P_{rev,\phi}$ (imposing suitable bounds on the plan length). We remark that like finding ϕ-reverse plans, conformant planning is Σ_3^p-complete [24, 7] in general and Σ_2^p-complete if action executability is testable in polynomial time [7]. The above transformation complies with this because it carries polynomial-time action executability in D over to $D_{rev,\phi}$.

We next consider an example of the transformation.

Example 9 Consider the setting of Example 4, and suppose we want to find reverse actions for other actions (i.e., unconditional single-step RPIs (AS, R, ϕ) where $|R| = 1$). The associated planning problem $P_{rev,\phi}$ where $\Phi = \top$ admits the following conformant plans of length 3:

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

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.□

In order to single out the effective ϕ-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 $P_{rev,\phi}$ 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 [13, 8] and $\text{DLV}^{\mathcal{K}}$ [6], to (1) generate arbitrary (not necessarily conformant) plans P in $P_{rev,\phi}$ 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 $P_{rev,\phi}$ to $true$ (and may simplify the resulting formulas). Let the resulting planning problem be $P^{norm}_{rev,\phi}$. More formally, we have the following result.

Theorem 11 *Let $P_{rev,\phi}$ and $P^{norm}_{rev,\phi}$ be the planning problem defined relative to the action description $D_{rev,\phi}$. Then, $\langle A_i, \ldots, A_{n-1}\rangle$ is an effective ϕ-reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geq i > 0$, is a plan for $P^{norm}_{rev,\phi}$ which is a conformant plan for $P_{rev,\phi}$.*

6.1.3 Reverse plan library construction

Using the results above, a reverse plan library for a collection $\phi_1, \ldots, \phi_m$ of reverse 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,

(*) for a given action domain $D(\mathcal{A}, \mathcal{F})$, a formula $\phi(\mathcal{F})$ of the form $\phi_1(\mathcal{F}) \vee \phi_2(\mathcal{F})$, action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, R is a ϕ-reverse plan for AS iff R is a ϕ_1-reverse plan for AS and R is a ϕ_2-reverse plan for AS.

Then k-DNF reverse conditions can be synthesized from terms of size at most k (in particular, clauses from literals), and any RPIs $(AS, R, \phi_1), \ldots, (AS, R, \phi_k)$ can be merged into a single RPI $(AS, R, \phi_1 \vee \cdots \vee \phi_k)$. We leave the construction of compact reverse plan libraries for further study.

6.2 Online reverse plan assembly

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 Figure 6. Here, the symbol + denotes concatenation, and exec(S,R) computes the state which results from executing the action sequence $R = \langle B_1, \ldots, B_k\rangle$ starting at state S.

The following theorem states that the algorithm runs in polynomial time and 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 *An action sequence R is a ϕ*-reverse plan for an action sequence AS from a state S relative to a reverse plan library L*, if one of the following holds:*

- *there is an RPI (AS, R, ϕ) in L such that $\phi(S)$ holds;*

Algorithm S-REVERSE(AS, S_i, L)

Input: Action sequence $AS = \langle A_0, \ldots, A_{i-1}\rangle$, $i \geq 0$,
state S_i, single-step reverse plan library L;
Output: Reverse plan RP from L for AS from state S_i
and resulting state, S_0, or "no" if none exists

S := S_i; RP := ϵ ; /* empty plan */
for each $j = i-1, i-2, \ldots, 0$ **do**
  if some $(\langle A\rangle, R, \phi) \in L$ exists
    s.t. A=A_j $\wedge$ $\phi(S)$= $\top$ **then**
  begin
    RP := RP + R;
    S := exec(S,R); /* undo A_j */
  end
  else return "no";
return (RP, S)

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

- *for some action sequences AS', R', there is an RPI (AS', R', ϕ) is in L such that $\phi(S)$ holds and the following condition is satisfied: for the action sequences AS'' and R'' such that $AS = AS'' + AS'$ and $R = R' + R''$, and for the state S' reached after execution of R' at S, R'' is a ψ-reverse plan for AS'' from the state S' relative to L.*

Theorem 12 (i) S-REVERSE(AS, S_i, L) *is a polynomial-time algorithm. Its running time is of order $O(|AS|(|L| \cdot \phi_{\max}(L) + exec_{\max}(L)))$, where $\phi_{\max}(L)$ and $exec_{\max}(L)$ bound the times to evaluate $\phi(S)$ for formula ϕ in L and exec(S,R) for any state S and reverse plan R in L, respectively.*

(ii) S-REVERSE(AS, S_i, L) *correctly outputs, relative to L, a reverse plan RP for AS from S_i, and the resulting state S_0; or it determines that such a reverse plan does not exist.*

Example 10 Consider the setting of Example 4. Take L to be the derived single-step unconditional plan library of Example 9. Consider the plan P of Example 3, the observed state S_2 at time stamp 2, and the evolution E_P of S_2 as in Example 8.

Then, as described earlier, the point of failure is state S_0. To produce a reverse plan for the action sequence $\langle throw(a, c, d), throw(b, table, c)\rangle$, to reach the point of failure from S_2, we call

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

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

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

Therefore, $RP := \langle carry(b, table) \rangle$ and S is computed as the state of time stamp 1 in E_P. For $j = 0$, the only match in L is

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

hence, $S = S_0$ and $RP := \langle carry(b, table), carry(a, c) \rangle$ are finally returned.□

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

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

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 Figure 7, which generalizes algorithm S-REVERSE.

The auxiliary array S in the algorithms in Figure 7 is used to keep state information when available, i.e., each $S[j]$ contains information about the state S_j before the action occurrences A_j. The main algorithm, REVERSE, initializes every $S[j]$ ($j < i$) of S to $\perp$ since we do not have information about the states $S_0, \ldots, S_{i-1}$ 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 can get some information about state S_{i-1} and modify $S[j-1]$ accordingly. Having such state 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 labelled by an action so that the path from the root to the node would describe an action sequence. If the node describes an action sequence As such that (As, R, ϕ) is in L then the node is linked to a list of all RPIs (As', R, ϕ) in L where As=As'.

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

Theorem 13 (i) REVERSE(AS, S_i, L) *is a polynomial-time algorithm. Its running time is of order* $O(|AS|(|L| \cdot \phi_{\max}(L) + exec_{\max}(\mathcal{A})) + \min(AS_{\max}(L), |AS|)$*, where* $\phi_{\max}(L)$ *bounds the time to evaluate* $\phi(S)$*, for any formula* ϕ *in L;* $exec_{\max}(L)$ *bounds the time to evaluate exec(S,R), for any state S and reverse plan R in L; and* $AS_{\max}(L) = \max\{|As| \mid (As, R, \phi) \in L\}$.

```
Algorithm REVERSE(AS, S_i, L)

Input:   Action sequence AS = ⟨A_0, ..., A_{i-1}⟩, i ≥ 0,
         state S_i, reverse plan library L;
Output:  Reverse plan RP from L for AS from state S_i
         and resulting state, S_0, or "no" if none exists

  Set S[0] := ⊥, ..., S[i-1] := ⊥ and S[i] := S_i;
  RP := REVERSE1(i);
  if RP = "no" then return "no"
  else return (RP, S[0])
```

```
Algorithm REVERSE1(j)

Input:   integer j, 0 ≤ j ≤ i (=|AS|);
Output:  Reverse plan RP for ⟨A_0, ..., A_{j-1}⟩ from S[j],
         or "no"

 if j = 0 then return ε ; /* empty plan */
 for each (As, R, φ) ∈ L such that
             S[j-|As|] = ⊥ and As is a suffix of ⟨A_0, ..., A_{j-1}⟩ do
   if φ(S_j) = ⊤ then
      begin
        S[j-|As|] := exec(S[j], R);   /* undo As from S_j */
        RP := REVERSE1(j-|As|);
        if RP ≠ "no" then return R + RP
      end
 return "no"
```

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

(ii) REVERSE(AS, S_i, L) *correctly outputs, relative to L, a reverse plan RP for AS from* S_i*, and the resulting state* S_0*; or it determines that such a plan does not exist.*

We remark that in an application setting, $|AS|$ as well as reverse plan R are expected to be small (bounded by a constant) and formulas ϕ consists of a few literals (i.e., a single literal, or a clause or term of size bounded by a constant. In this case, the running time is $O(|L|+exec_{\max}(L))$, assuming a linear time implementation of the transition function $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$, is 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 Related Work

Our method of undoing the effects of the execution of an action sequence is different from the existing one [15, 16] mainly in two ways. First, in [15, 16], 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 [15, 16], 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 [15, 16]. 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 [15, 16]. 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 [12, 22, 23, 10] as follows.

In [12], 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 [22], 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 [23], backtracking is considered in connection with inserting corrective plans as in [12], by a recursive recovery procedure like the one in [22]. 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 [10], 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 [25, 1], 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.

8 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 [24]. By means of such an action or a plan, regardless of the current state, 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 non-determinism; 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.

Implementation of the algorithms above, including the generation of conditional reverse plan libraries, is in our future agenda. These implementations then will be used in a framework for reactive planning and execution monitoring based on answer set programming, which is pursued in an ongoing project.

Acknowledgments

We thank Mikhail Soutchanski for useful discussions. 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.

Appendix: Proofs of Theorems

8.1 Proof of Theorem 1

Theorem 1 *Given an action domain description* $D(\mathcal{A}, \mathcal{F})$, *two actions* A *and* A' *in* $2^{\mathcal{A}}$, *and a formula* $\phi(\mathcal{F})$, *deciding whether* A' *is a* ϕ*-reverse action for* A *is* coNP*-complete. The hardness holds even if* $\phi(\mathcal{F}) = \top$ *and* $A' = A$.

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

Membership. The problem of deciding whether A' is a ϕ-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 $\psi(\mathcal{X})$ on atoms $\mathcal{X} = \{X_1, \dots, X_n\}$, deciding whether $\psi(\mathcal{X})$ 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 a formula $\phi(\mathcal{F})$ obtainable from $\psi(\mathcal{X})$ and $\mathcal{X}$ in polynomial time such that $\psi(\mathcal{X})$ is a tautology iff A' is a ϕ-reverse action for A. Without loss of generality, we may assume that $\psi(\mathcal{X})$ holds whenever all atoms in $\mathcal{X}$ are mapped to true.

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

$$\mathit{act}(\mathcal{F}, \mathcal{A}, \mathcal{F}') = X_1' \wedge \cdots \wedge X_n' \wedge ((X_1 \wedge \cdots \wedge X_n) \vee \neg\psi(\mathcal{X})).$$

Take $\phi(\mathcal{F}) = \top$, and take $A = A' = \{\}$. Notice that the action domain $D(\mathcal{A}, \mathcal{F})$ is deterministic.[4]

[4] An action domain is *deterministic* iff

$$\forall \mathcal{F}, \mathcal{A}, \mathcal{F}', \mathcal{F}'' \ (\mathit{tr}(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge \mathit{tr}(\mathcal{F}, \mathcal{A}, \mathcal{F}'') \supset \mathcal{F}' \ \equiv \ \mathcal{F}'').$$

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 $\neg\psi(S)$. Then, due to the definition of a reverse action, $S = S'$ and, for every proper subset $\mathcal{X}_i$ of $\mathcal{X}$, $\psi(\mathcal{X}_i)$. This, with the assumption on $\psi(\mathcal{X})$, implies that $\psi(\mathcal{X})$ is a tautology.

Conversely, suppose that $\psi(\mathcal{X})$ 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. □

8.2 Proof of Theorem 2

Theorem 2 *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 a formula $\phi(\mathcal{F})$, deciding whether $\langle A\rangle$ is a ϕ-reverse plan for AS is* coNP*-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$ and $|AS| = 1$.*

Proof. 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 $\phi(\mathcal{F})$ be a formula.

Membership. Formula (12) with $|\langle A\rangle| = m = 1$ and $|AS| = k$ is:

$$\begin{array}{l}\exists \mathcal{F}_0, \ldots, \mathcal{F}_k \ (\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A) \ \wedge \mathcal{F}' \equiv \mathcal{F}_k) \ \supset \\ \qquad \forall \mathcal{F}'_0, \ldots, \mathcal{F}'_m \ (\ \mathcal{F}'_0 \equiv \mathcal{F}' \supset (tr(\mathcal{F}'_0, A'_0, \mathcal{F}'_1) \supset \mathcal{F}'_1 \equiv \mathcal{F}))\end{array}$$

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

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

Then (13) with $|\langle A\rangle| = m = 1$ and $|AS| = k$ is equivalent to a QBF of the form

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

This implies that the problem of deciding whether $\langle A\rangle$ is a ϕ-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 1. □

8.3 Proof of Theorem 3

Theorem 3 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, two action sequences AS and R consisting of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether R is a ϕ-reverse plan for AS is Π_2^p-complete. The hardness holds even if $\phi(\mathcal{F}) = \top$ and, moreover, $|R| \geq 2$ is fixed.*

Proof. 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 $\phi(\mathcal{F})$ be a formula.

Membership. Formula (12) is a QBF of the form

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

Hence, (13) is equivalent to the QBF

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

This implies that the problem of deciding whether R is a ϕ-reverse plan for AS reduces in polynomial time to the problem of deciding whether a QBF of the form

$$\forall \mathcal{X}\ \exists \mathcal{Y}\ \alpha(\mathcal{X}, \mathcal{Y}) \tag{26}$$

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 (26), 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 a formula $\phi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{X}, \mathcal{Y})$, $\mathcal{X}$, and $\mathcal{Y}$ in polynomial time such that (26) is true iff R is a ϕ-reverse plan for AS.

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

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

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

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

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

Suppose that R is a reverse plan for AS. Take any two states S and S', such that $\mathit{tr}(S, A, S')$. 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 (26) is true.

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

$$\begin{array}{rl} \mathit{act}(\mathcal{F}, \mathcal{A}, \mathcal{F}') = & (\mathcal{X} \equiv \mathcal{X}') \wedge (A \equiv 0) \wedge \\ & (0 \supset 1' \wedge \bigwedge_{Y_i \in \mathcal{Y}} Y_i) \wedge \\ & (1 \supset 2' \wedge \alpha(\mathcal{X}', \mathcal{Y}')) \wedge \\ & \vdots \\ & (|R| - 1 \supset |R|' \wedge \mathcal{Y} \equiv \mathcal{Y}') \wedge \\ & (|R| \supset 0' \wedge \bigwedge_{Y_i \in \mathcal{Y}} Y_i'). \end{array}$$

8.4 Proof of Theorem 4

Theorem 4 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and an action A in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and an action A' in $2^{\mathcal{A}}$ such that A' is an effective ϕ-reverse action for A is Σ_2^p-complete.*

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

Membership. Fact (*) presented in Section 6.1 implies that without loss of generality, $\phi(\mathcal{F})$ is a conjunction of literals (i.e., symbols in $\mathcal{F}$, possible preceded by the classical negation symbol $\neg$). A suitable such $\phi(\mathcal{F})$ can be guessed in polynomial time. Furthermore, deciding, for some $\mathcal{F}$ and $\mathcal{F}'$, whether $\mathit{tr}(\mathcal{F}, A, \mathcal{F}') \wedge \phi(\mathcal{F}')$ holds is in NP. On the other hand, due to Theorem 1, we can decide whether A' is a ϕ-reverse action for A in coNP. Therefore, we can decide whether there exist a formula $\phi(\mathcal{F})$ and an action A' in $2^{\mathcal{A}}$ such that A' is a ϕ-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

$$\exists \mathcal{Y} \, \forall \mathcal{X} \; \alpha(\mathcal{X}, \mathcal{Y}) \tag{27}$$

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 (27) is satisfiable iff there exist a formula $\phi(\mathcal{F})$ and an action A' in $2^{\mathcal{A}}$ such that A' is a ϕ-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 $\mathit{state}(\mathcal{F}) = \top$ and take

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

Take $A = \{B\}$. For any state S, let $\phi(S)$ 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 $\phi(S')$-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., $S = \mathcal{X}$. On the other hand, by the definition of a conditional reversal, at S', $\phi(S')$ 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 ϕ-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 (27) is true.

Conversely, suppose (27) 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 $tr(S, A, S')$, and that $S' = \mathcal{X} \cup \mathcal{Y}'$, the action $B_{S'}$ is a $\phi(S')$-reverse action for A. □

8.5 Proof of Theorem 5

Theorem 5 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and a sequence R of at most two actions in $2^{\mathcal{A}}$ such that R is an effective ϕ-reverse plan for AS is Σ_2^p-complete. The Σ_2^p-hardness holds even if $|AS| = 1$.*

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

Membership. Fact (*) presented in Section 6.1 implies that, without loss of generality, $\phi(\mathcal{F})$ can be taken as a conjunction of literals that holds at a single state S_{n+1}. 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-reverse plan for AS iff there exist states S_0 and S_{n+1} such that

(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 formula $\phi(\mathcal{F})$ and a sequence R of at most two actions in $2^{\mathcal{A}}$, R is a ϕ-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 (27) 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 (27) 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-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

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

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

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

Take $AS = \langle B \rangle$. For any state S, let $\phi(S)$ 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 $\textit{tr}(S, AS, S')$. Suppose that $R = \langle B_{S'}, \emptyset \rangle$ is a $\phi(S')$-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 ϕ-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 (27) is true.

Conversely, suppose (27) 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 $\textit{tr}(S, AS, S')$, and that $S' = \mathcal{X} \cup \mathcal{Y}'$, the action sequence $\langle B_{S'}, \emptyset \rangle$ is a $\phi(S')$-reverse action for AS. □

8.6 Proof of Theorem 6

Theorem 6 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$ and a sequence AS of actions in $2^{\mathcal{A}}$, deciding whether there exist a formula $\phi(\mathcal{F})$ and a sequence R of polynomial number of actions in $2^{\mathcal{A}}$ such that R is an effective ϕ-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.*

Proof. 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 fact (*) presented in Section 6.1, without loss of generality, take $\phi(\mathcal{F})$ to be a conjunction of literals. Therefore, suitable $\phi(\mathcal{F})$ and R such that R is a ϕ-reversal plan for AS can be guessed and, by Theorem 3, 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

$$\exists \mathcal{Z} \forall \mathcal{X}\, \exists \mathcal{Y}\, \alpha(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) \tag{28}$$

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 (28) is satisfiable iff there exist a formula $\phi(\mathcal{F})$ and an action sequence R of polynomial number of actions in $2^{\mathcal{A}}$ such that R is a ϕ-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

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

and take

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

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 $\phi(S)$ 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 $\mathit{tr}(S_0, AS, S_1)$; this is possible due to the definition of *act*. Note that $\mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y} \subseteq S_0$. Suppose that $R = \langle A_1, \emptyset, \emptyset \rangle$ is a ϕ-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 reverse plan for AS, for each S_2 some such S_3 must exist; and from S_3, we reach at the state S_0 where $\mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y} \subseteq S_0$. This is tantamount to (28) being true.

Now consider ϕ-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 ϕ-reverse plan (as we have seen above). $\Box$

8.7 Proof of Theorem 7

Theorem 7 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether there exists a ϕ-reverse plan R for AS such that $|R| = 1$ is Σ_2^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$ and $|AS| = 1$.*

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

Membership. The problem is in Σ_2^p, since a suitable R can be guessed and, by Theorem 2, 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

$$\exists \mathcal{Z} \forall \mathcal{X}\ \alpha(\mathcal{Z}, \mathcal{X}) \tag{29}$$

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 a formula $\phi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{Z}, \mathcal{X})$, $\mathcal{Z}$, and $\mathcal{X}$ in polynomial time, such that (29) is satisfiable iff there exists an action sequence R consisting of one action in $2^{\mathcal{A}}$ such that R is a ϕ-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

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

Take $\phi(\mathcal{F}) = \top$ 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 (29) is true. □

8.8 Proof of Theorem 8

Theorem 8 *Given an action domain description $D(\mathcal{A}, \mathcal{F})$, a sequence AS of actions in $2^{\mathcal{A}}$, and a formula $\phi(\mathcal{F})$, deciding whether there exists a ϕ-reverse plan R for AS is Σ_3^p-complete. Hardness holds even if $\phi(\mathcal{F}) = \top$ and $|R|$ is fixed to 2.*

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

Membership. The problem is in Σ_3^p, since a suitable R can be guessed and, by Theorem 3, 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

$$\exists \mathcal{Z} \forall \mathcal{X}\ \exists \mathcal{Y}\ \alpha(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) \tag{30}$$

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 a formula $\phi(\mathcal{F})$, each obtainable from $\alpha(\mathcal{Z}, \mathcal{X})$, $\mathcal{Z}$, and $\mathcal{X}$ in polynomial time, such that (30) 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-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

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

and let

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

Take $\phi(\mathcal{F}) = \top$ 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-reverse plan for AS iff QBF (30) 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 $n \geq 1$ 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) $A_{i,2} \in 2^{\mathcal{A} \setminus \{A\}}$. □

8.9 Proof of Theorem 9

Theorem 9 *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.*

Proof. Consider any conformant plan $P = \langle A_0, \ldots, A_{n-1} \rangle$ of length $n \geq 1$. 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 (15), $S_0 \equiv \tilde{S}_0$ holds, and thus by (14), 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, $i \leq n$, 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. □

8.10 Proof of Theorem 10

Theorem 10 *Let $P_{rev,\phi}$ be the planning problem defined relative to the action description $D_{rev,\phi}$. Then, $\langle A_i, \ldots, A_{n-1} \rangle$ is a ϕ-reverse plan for $\langle A_0, \ldots, A_{i-1} \rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1} \rangle$, $n \geq i > 0$, is a conformant plan for $P_{rev,\phi}$.*

Proof. First, assume that $\langle A_i, \ldots, A_{n-1}\rangle$ is a ϕ-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D (where $n \geq i > 0$). 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 $P_{rev,\phi}$, i.e. (6) holds for $init_{rev,\phi}$, $tr_{rev,\phi}$ and $goal_{rev,\phi}$. Let $P = \langle A'_0, \ldots, A'_{i-1}, A'_i, A'_{i+1}, \ldots, A'_n\rangle$. Then (6) reads as follows:

$$\begin{aligned}&\forall \mathcal{F}_0, \ldots, \mathcal{F}_n \exists \mathcal{F}'_0, \ldots, \mathcal{F}'_{n+1}\ \ init_{rev,\phi}(\mathcal{F}'_0)\\ &\quad \wedge \textstyle\bigwedge_{t=0}^{n} \Big(init_{rev,\phi}(\mathcal{F}_0) \wedge tr_{rev,\phi,t}(\mathcal{F}, A') \supset tr_{rev,\phi}(\mathcal{F}_t, A'_t, \mathcal{F}'_{t+1})\Big)\\ &\quad \wedge \Big(init_{rev,\phi}(\mathcal{F}_0) \wedge tr_{rev,\phi,n+1}(\mathcal{F}, A') \supset goal_{rev,\phi}(\mathcal{F}_{n+1})\Big)\end{aligned} \tag{31}$$

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

For the second conjunct, consider any evolution $S_0, A'_0, S_1, \ldots, A'_t$, $0 \leq t \leq n$. Observe that if $t \leq i-1$, 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 (25); (ii) $\phi(S_i \cap \mathcal{F})$ does not hold, then a successor "don't care" state exists by (20); (iii) $\phi(S_i \cap \mathcal{F})$ holds, then a successor state exists, which is equal to state i, except for the fact that sep_occ holds by (22),(23). For $i < t \leq n$, 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 (25); (ii) $normal$ holds in state t, then by virtue of (13) and (12) 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 (19) is satisfiable. Indeed, note that in any evolution of such a state S_t, $\phi(S_i \cap \mathcal{F})$ holds, and therefore (12) must hold. Furthermore, sep_occ holds at S_{t+1} by (24).

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 (13) the formula $goal_{rev}(\mathcal{F}_{rev})$ and, as argued in the discussion above, sep_occ hold at S_{n+1}. Thus, $goal_{rev,\phi}(S_{n+1})$ holds.

In total, we have demonstrated that (31) is is true. We therefore obtain that P is a conformant plan of $P_{rev,\phi}$.

Now, assume that $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geq i > 0$, is a conformant plan of $P_{rev,\phi}$. We show that $\langle A_i, \ldots, A_{n-1}\rangle$ is a ϕ-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 $P_{rev,\phi}$. We show now that (13) holds for $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$, where for any $j \geq 0$, $S^\star_j$ is the restrictions of S_j to the fluents in D.

Assume first that each trajectory contains a state S_j, where $0 < j \leq i$, in which $normal$ does not hold. Then the antecedent of (12) 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 (13) for $\mathcal{F} = S^\star_0$ and $\mathcal{F}' = S^\star_i$.

Next, assume that each trajectory contains a state S_j, where $0 < j \leq i+1$, in which $normal$ does not hold. Then, either $j \leq i$ and as before the antecedent of (12) is false, or $j = i+1$ and by (20) $\phi(S^\star_i)$ does not hold. In any case, (13) 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 leads only to states in which ϕ does not hold.

Now assume that some trajectory exists in which $normal$ holds at every state S_j, where $0 \leq j \leq i+1$. Here $\phi(S^\star_i)$ clearly holds, and the antecedent of (12) holds. Since P is a conformant plan, for each state S_k, $k \geq i+1$, by (19) a state S_{k+1} exists such that $tr_{rev,\phi}(S_k, A_{k-1}, S_{k+1})$ holds. By construction, also $tr(S_k^\star, A_{k-1} \cap \mathcal{A}, S_{k+1}^\star)$ is then true. Finally, $goal_{rev,\phi}(S_{n+1})$ 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 $S_0^\star \equiv S_{n+1}^\star$. Hence, the last conjunct of (12) is true.

In total, we have that for any $\mathcal{F} = S_0^\star$ and $\mathcal{F}' = S_i^\star$ where S_0 and S_i are from a trajectory of P with respect to $P_{rev,\phi}$, formula (13) holds. For all other values for $\mathcal{F}$ and $\mathcal{F}'$ the antecedent of (12) does not hold. Hence, (13) holds for all $\mathcal{F}$ and $\mathcal{F}'$. Therefore $\langle A_i, \ldots, A_{n-1}\rangle$ is a ϕ-reverse plan of $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D. □

8.11 Proof of Theorem 11

Theorem 11 *Let $P_{rev,\phi}$ and $P_{rev,\phi}^{norm}$ be the planning problem defined relative to the action description $D_{rev,\phi}$. Then, $\langle A_i, \ldots, A_{n-1}\rangle$ is an effective ϕ-reverse plan for $\langle A_0, \ldots, A_{i-1}\rangle$ relative to D if and only if $\langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$, $n \geq i > 0$, is a plan for $P_{rev,\phi}^{norm}$ which is a conformant plan for $P_{rev,\phi}$.*

Proof. If $\langle A_i, \ldots, A_{n-1}\rangle$ is an effective ϕ-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 $\langle A_0, \ldots, A_{i-1}\rangle$ such that $\phi(S_i)$ holds, and by Theorem 10 $P = \langle A_0, \ldots, A_{i-1}, \{sep\}, A_i, \ldots, A_{n-1}\rangle$ is a conformant plan of $P_{rev,\phi}$. 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, $0 \leq j \leq n+1$. By definition of $P_{rev,\phi}^{norm}$, P is thus a plan for $P_{rev,\phi}^{norm}$. This proves the only if direction.

For the if direction, by Theorem 10 $\langle A_i, \ldots, A_{n-1}\rangle$ is a ϕ-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 $P_{rev,\phi}^{norm}$, by construction $P_{rev,\phi}$ has a trajectory $S_0, A_0, \ldots, A_{n-1}, S_{n+1}$ such that $normal$ is true at each state S_j, $0 \leq j \leq n+1$. Hence, $\phi(S_i \cap \mathcal{F})$ holds. 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. □

8.12 Proof of Theorem 12

Theorem 12

(i) S-REVERSE(AS, S_i, L) *is a polynomial-time algorithm. Its running time is of order $O(|AS|(|L| \cdot \phi_{\max}(L) + exec_{\max}(L)))$, where $\phi_{\max}(L)$ and $exec_{\max}(L)$ bound the times to evaluate $\phi(S)$ for formula ϕ in L and exec(S,R) for any state S and reverse plan R in L, respectively.*

(ii) S-REVERSE(AS, S_i, L) *correctly outputs, relative to L, a reverse plan RP for AS from S_i, and the resulting state S_0; or it determines that such a reverse plan does not exist.*

Proof. See the proof of Theorem 13.

8.13 Proof of Theorem 13

Theorem 13

(i) REVERSE(AS, S_i, L) *is a polynomial-time algorithm. Its running time is of order* $O(|AS|(|L| \cdot \phi_{\max}(L) + exec_{\max}(\mathcal{A})) + \min(AS_{\max}(L), |AS|)$, *where* $\phi_{\max}(L)$ *bounds the time to evaluate* $\phi(S)$, *for any formula* ϕ *in* L; $exec_{\max}(L)$ *bounds the time to evaluate exec(*S*,*R*), for any state* S *and reverse plan* R *in* L; *and* $AS_{\max}(L) = \max\{|As| \mid (As, R, \phi) \in L\}$.

(ii) REVERSE(AS, S_i, L) *correctly outputs, relative to* L, *a reverse plan* RP *for* AS *from* S_i, *and the resulting state* S_0; *or it determines that such a plan does not exist.*

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

(O1) exec($S[j], R$) 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 Subsection 6.2, 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)) can be determined in time $O(\min(AS_{\max}(L), |AS|))$. Then, the tuples (AS, R, ϕ) in the for-loop of REVERSE1 can be found in $O(\min(AS_{\max}(L), |AS|)$ time. For each tuple, checking whether $\phi(S_j) = \top$ is feasible in $\phi_{\max}(L)$ time; hence, the total time for such checks is bounded by $|L| \cdot \phi_{\max}(L)$. Hence, modulo calls of exec($S[j], R$) and recursive calls of REVERSE1, the body of REVERSE1 takes $O(|L| \cdot \phi_{\max}(L) + \min(AS_{\max}(L), |AS|)$ time.

On the other hand, REVERSE1(0) takes constant time; at each call of REVERSE1(j), exec($S[j], R$) takes at most $exec_{\max}(L)$ time. Then, due to (O1) and (O2), REVERSE1(i) takes in total $O(i(|L| \cdot \phi_{\max}(L) + exec_{\max}(L) + \min(AS_{\max}(L), |AS|))$ time.

(ii) If R_i is a ϕ_j-reverse plan for an action sequence AS_j, for $j \in \{1, 2\}$, then $R_2 + R_1$ is a ϕ_2-reverse plan for $AS_1 + AS_2$ from the state S' resulting after $AS_1 + AS_2$, if $\phi_2(S')$ holds and $\phi_1(S)$ holds at the state S that results after executing R_2 starting at S'. Thus, the algorithm REVERSE is clearly sound, i.e., produces only correct reverse plans for AS from the current state S_i relative to L. On the other hand, REVERSE is also complete and finds any ϕ-reverse plan for AS from S_i relative to L. Indeed, whenever R_1 and R_2 are ϕ_1- and ϕ_2-reverse plans, respectively, for the same action sequence AS and $\phi_1(S')$ = $\phi_2(S')$ = $\top$ holds at some state S' reached by AS, then R_1 and R_2 must reverse S' to the same state S (otherwise, by definition not both can be valid reverse plans at the same time). Hence, regardless of how tuples in the for-loop of REVERSE1 are considered, the array $S[0, ..., i]$ is filled with unique values (in fact, with the states of the execution resulting from AS). Therefore, by induction on the length $|AS| \geq 1$ of the sequence to be reversed, one can easily show that REVERSE(AS,S_i,L) outputs some ϕ-reverse plan R for AS relative to L from S_i, if one exists. □

References

[1] Hiralal Agrawal, Richard A. DeMillo, and Eugene H. Spafford. An execution backtracking approach to program debugging. *IEEE Software*, 8(3):21–26, 1991.

[2] Chitta Baral, Vladik Kreinovich, and Raúl Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1-2):241–267, 2000.

[3] Marco Cadoli and Franceso M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

[4] Thomas Eiter, Esra Erdem, and Wolfgang 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.

[5] Thomas Eiter, Esra Erdem, and Wolfgang Faber. Plan reversals for recovery in execution monitoring. In *Proc. 10th International Workshop on Nonmonotonic Rea soning (NMR-2004), Action and Causality Track*, pp. 147–154, June 2004.

[6] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, II: The DLV$^{\mathcal{K}}$ system. *Artificial Intelligence*, 144(1–2):157–211, 2002.

[7] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM TOCL*, 5(2):206–263, 2004.

[8] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. 17th National Conference on Artificial Intelligence (AAAI'00)*, pp. 748–753. AAAI Press / The MIT Press, 2000.

[9] Matthias Fichtner, Axel Großmann, and Michael Thielscher. Intelligent execution monitoring in dynamic environments. In *Proc. IJCAI 2003 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating*, 2003.

[10] Matthias Fichtner, Axel Großmann, and Michael Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2–4), 2003.

[11] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2(3-4):193–210, 1998.

[12] Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning*, pp. 453–465, 1998.

[13] Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Proc. 7th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, pp. 657–666. Morgan Kaufmann, 2000.

[14] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.

[15] Hisashi Hayashi, Kenta Cho, and Akihiko Ohsuga. Mobile agents and logic programming. In *Proc. 6th International Conference on Mobile Agents (MA 2002)*, LNCS 2535, pp. 32–46. Springer, 2002.

[16] Hisashi Hayashi, Kenta Cho, and Akihiko Ohsuga. A new HTN planning framework for agents in dynamic environments. In *Post-Proc. 4th International Workshop on Computational Logic and Multi-Agent Systems (CLIMA IV)*, LNCS 3259, pp. 108–133. Springer, 2004.

[17] David S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Pub., 1990.

[18] Norman McCain and Hudson Turner. Satisfiability planning with causal theories. In *Proc. 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pp. 212–223. Morgan Kaufmann, 1998.

[19] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[20] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[21] Tran Cao Son and Chitta Baral. Formalizing sensing actions – A transition function based approach. *Artificial Intelligence*, 125(1–2):19–91, 2001.

[22] Mikhail Soutchanski. Execution monitoring of high-level temporal programs. In *Proc. IJCAI 1999 Workshop on Robot Action Planning*, 1999.

[23] Mikhail Soutchanski. High-level robot programming and program execution. In *Proc. ICAPS 2003 Workshop on Plan Execution*, 2003.

[24] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, LNCS 2424, pp. 111–124, 2002.

[25] Marvin V. Zelkowitz. Reversible execution. *Communications of ACM*, 16(9):566, 1973.