

# On Reversing Actions: Algorithms and Complexity\*

**Thomas Eiter**

Vienna University of Technology  
eiter@kr.tuwien.ac.at

**Esra Erdem**

Sabancı University  
esraerdem@sabanciuniv.edu

**Wolfgang Faber**

University of Calabria  
faber@mat.unical.it

## Abstract

Reversing actions is the following problem: After executing a sequence of actions, which sequence of actions brings the agent back to the state just before this execution (an action reversal). Notably, this problem is different from a vanilla planning problem since the state we have to get back to is in general unknown. It emerges, for example, if an agent needs to find out which action sequences are undoable, and which ones are committed choices. It has applications related to plan execution and monitoring in nondeterministic domains, such as recovering from a failed execution by partially undoing the plan, dynamically switching from one executed plan to another, or restarting plans. We formalize action reversal in a logic-based action framework and characterize its computational complexity. Since unsurprisingly, the problem is intractable in general, we present a knowledge compilation approach that constructs offline a reverse plan library for efficient (in some cases, linear time) online computation of action reversals. Our results for the generic framework can be easily applied for expressive action languages such as  $\mathcal{C}+$  or  $\mathcal{K}$ .

## 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., [Gelfond and Lifschitz, 1998; Giunchiglia *et al.*, 2004; Son and Baral, 2001; Eiter *et al.*, 2004]), 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).

We study another reasoning problem about actions, namely the problem of undoing the effects of an execution of an

action sequence, by executing other actions. For example, after doing the action  $go(home, office)$ , the action  $go(office, home)$  may reverse its effects and bring the agent back to her previous state. If this holds regardless of the state in which the agent was before doing  $go(home, office)$  and afterwards, then  $go(office, home)$  is called a *reverse action* for  $go(home, office)$ . If, more generally, a sequence of actions  $R = B_1, \dots, B_m$  is guaranteed to bring the agent back to the state before executing a sequence  $AS = A_1, \dots, A_n$ , then  $R$  is called a *reverse plan* for  $AS$ . For example,  $R = go(office, pub), go(pub, home)$  may be a reverse plan for  $AS = go(home, bus\_stop), go(bus\_stop, office)$ .

Undo actions are well-studied in the area of databases, where they are a standard method for error recovery. In a more general context of plan execution and recovery, [Hayashi *et al.*, 2002; 2004] use undo actions for execution of plans by mobile agents in a dynamic environment. However, the undo actions (one for each action) need to be specified (manually) by the user. It therefore is desirable to have tools which automatically generate undo actions, or more generally, reverse plans. This raises the following questions: given an action domain and an action  $A$ , does there exist a reverse action for  $A$ ? More generally, given a sequence of actions  $AS$ , does there exist a reverse plan for  $AS$ ? If so, how can a reverse action or plan be efficiently computed? From a computational point of view, can reverse actions or plans be fruitfully exploited for efficient backtracking in action execution?

Backtracking may be considered for various reasons, like to restart a plan (e.g., when the execution of the plan fails due to some undesired effects of an action in a nondeterministic environment), or to switch from the current plan to one which is better (or safer) in the light of new information. When the current state and the state we want to backtrack to are known, then the problem amounts to a vanilla planning problem, which is intractable in general. However, the problem is different if the backtrack state is unknown.

Motivated by these questions, we study computational aspects of action reversals. The main contributions of this paper are as follows.

- We formally define the notions of a reverse action and a reverse plan for actions. Rather than to commit to a particular action language capable of modelling nondeterministic effects, such as  $\mathcal{C}+$  [Giunchiglia *et al.*, 2004] or  $\mathcal{K}$  [Eiter *et al.*, 2004], we use here a generic transition-based

\*This work was supported by FWF (Austrian Science Fund) under project P16536-N04. The work of the second author was carried out while she visited TU Wien. The third author was funded by an APART grant of the Austrian Academy of Sciences.

framework for representing actions as in [Rintanen, 1999; Turner, 2002], using propositional logic as a specification language. Besides nondeterminism, it also accommodates concurrent actions and dynamic worlds. We extend the definitions to *conditional reversals*, considering also partial knowledge about the current state and the state before the execution.

- We thoroughly analyze the complexity of action reversals, and characterize the complexity of recognizing and deciding existence of reverse actions and plans, both for plain as well as for conditional reversals.
- Unsurprisingly, action reversal is intractable in general. For monitoring applications, we therefore present a knowledge compilation method. It constructs offline a *reverse plan library* from which reversals can be online computed in polynomial (often, linear) time for important classes of instances.

Our results shed light on the complexity of action reversals, and can be easily customized to particular action languages like  $\mathcal{C}+$  or  $\mathcal{K}$ . Our algorithms for reverse plan assembly suggest action reversal as a complementary method for efficient backtracking (if no reverse plan exists, choose some other method). For further in-depth material and proofs for all results of this paper, we refer to [Eiter *et al.*, 2006].

## 2 Action Representation Framework

Following [Turner, 2002], let  $\mathcal{A}$  be a set of action symbols and let  $\mathcal{F}$  be a disjoint set of fluent symbols, which are viewed as propositional atoms. The set of states of an action domain is encoded by the models of a propositional formula  $state(\mathcal{F})$  over  $\mathcal{F}$ . Let  $act(\mathcal{F}, \mathcal{A}, \mathcal{F}')$  be a formula over  $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$ , where  $\mathcal{F}' = \{f' \mid f \in \mathcal{F}\}$ . Then

$$tr(\mathcal{F}, \mathcal{A}, \mathcal{F}') = state(\mathcal{F}) \wedge act(\mathcal{F}, \mathcal{A}, \mathcal{F}') \wedge state(\mathcal{F}') \quad (1)$$

encodes the set of transitions that corresponds to its models. That is, in a transition, the start state corresponds to an assignment  $S$  to  $\mathcal{F}$ ,<sup>1</sup> the (concurrent) action execution (or occurrence) to an assignment  $A$  to  $\mathcal{A}$ , and the end state to an assignment  $S'$  to  $\mathcal{F}'$ .

**Example 1** [Giunchiglia *et al.*, 2004] 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 action symbols  $\mathcal{A} = \{putIntoWater, dryWithTowel\}$ , the states can be described by the formula  $state(\mathcal{F}) = inWater \supset wet$ . Since there are three assignments to  $\mathcal{F}$  satisfying  $state(\mathcal{F})$  ( $\{inWater, wet\}$ ,  $\{\neg inWater, wet\}$ ,  $\{\neg inWater, \neg wet\}$ ) there are three states:  $\{inWater, wet\}$ ,  $\{wet\}$ ,  $\{\}$ . The action occurrences can be defined through

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

By the last line, *dryWithTowel* is executable if *inWater* is false, but not concurrently with *putIntoWater*. For example, the assignment  $\{\neg inWater, wet, dryWithTowel,$

<sup>1</sup>“Assignment to  $S$ ” means an assignment of truth values to the symbols in  $S$ .

$\neg putIntoWater, \neg inWater', \neg wet'\}$  satisfies  $tr(\mathcal{F}, \mathcal{A}, \mathcal{F}')$ ; therefore, it describes a transition from the state  $S = \{wet\}$  to the state  $S' = \{\}$  by executing the action  $A = \{dryWithTowel\}$ .  $\square$

The meaning of a domain description can be represented by a transition diagram, which is a directed labelled graph whose nodes are the states and whose edges correspond to action occurrences. A *trajectory* of length  $n$  is an alternating sequence  $S_0, A_0, S_1, \dots, S_{n-1}, A_{n-1}, S_n$  of states  $S_i$  and action occurrences  $A_i$ , such that  $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \dots, S_{n-1} \xrightarrow{A_{n-1}} S_n$  is a labelled path in the graph. The trajectory can be obtained from a corresponding model of the formula  $tr_n(\mathcal{F}, \mathcal{A}) = \bigwedge_{i=0}^{n-1} tr(\mathcal{F}_i, \mathcal{A}_i, \mathcal{F}_{i+1})$  where each  $\mathcal{F}_i$  (resp., each  $\mathcal{A}_i$ ) results by adding time stamp  $i$  to each  $f \in \mathcal{F}$  (resp., each  $a \in \mathcal{A}$ ).

An *action sequence* of length  $n$  is a sequence  $AS = \langle A_0, \dots, A_{n-1} \rangle$ , where each  $A_i$  ( $0 \leq i < n$ ) is a (concurrent) action occurrence. We use  $|AS|$  to denote the length of  $\mathcal{A}$ . Note that in general,  $|AS|$  is different from the total number of single action occurrences in  $AS$ .

In what follows,  $\mathcal{F} \equiv \mathcal{F}'$  denotes  $\bigwedge_{f \in \mathcal{F}} f \equiv f'$ .

## 3 Execution Reversals

After an agent executes an sequence  $\langle A_0, \dots, A_i \rangle$ , it may be sometimes desirable that the effects of the whole or part of the action sequence be undone, such that the agent is back in the state  $S_j$ ,  $j < i$ , which she had reached after executing the actions  $A_0, \dots, A_{j-1}$ .

An action can be undone by executing one of its “reverse actions” or by executing a “reverse plan”. We define a reverse of an action below relative to a given action description.

**Definition 1** An action  $A'$  is a reverse action for  $A$ , if, for all  $\mathcal{F}$  and  $\mathcal{F}'$ , the formula  $revAct(\mathcal{F}, \mathcal{F}'; A, A')$ , defined as

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

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

The formula above expresses the following condition about actions  $A$  and  $A'$ . Take any two states  $S, S'$  (described by the assignments to 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  $S$ .

Many of the benchmarks used in planning are from the transportation domain (logistics, blocks world, grid, etc.). E.g., moving from  $x$  to  $y$  is the reverse action of moving from  $y$  to  $x$ , putting down an object is the reverse of picking it up.

**Definition 2** A reverse plan for an action  $A$  is a sequence  $\langle A'_0, \dots, A'_{m-1} \rangle$ ,  $m \geq 0$ , of actions such that, for all  $\mathcal{F}$  and  $\mathcal{F}'$ , the following formula is true:

$$revPlan(\mathcal{F}, \mathcal{F}'; A, [A'_0, \dots, A'_{m-1}]) =$$

$$tr(\mathcal{F}, A, \mathcal{F}') \supset \\ \forall_{i=1}^m \mathcal{F}_i \exists_{j=1}^m \mathcal{F}'_j \left( \mathcal{F}_0 \equiv \mathcal{F}' \supset \right. \\ \left. \left( \bigwedge_{t=0}^{m-1} (tr_t(\mathcal{F}, A') \supset tr(\mathcal{F}_t, A'_t, \mathcal{F}'_{t+1})) \wedge \right. \right. \\ \left. \left. (tr_m(\mathcal{F}, A') \supset \mathcal{F}_m \equiv \mathcal{F}') \right) \right).$$

The formula above expresses the following condition about an action  $A$  and an action sequence  $\langle A'_0, \dots, A'_{m-1} \rangle$ . Take any two states  $S, S'$  (described by the assignments to fluents in  $\mathcal{F}$  resp.  $\mathcal{F}'$ ) such that executing  $A$  at  $S$  leads to  $S'$ . Then the action sequence  $\langle A'_0, \dots, A'_{m-1} \rangle$  is executable at state  $S'$ , and it always leads to  $S$ . The executability condition of  $\langle A'_0, \dots, A'_{m-1} \rangle$  is described above by the formula on the second line. Note that  $revPlan(\mathcal{F}, \mathcal{F}'; A, [A'_0])$  is equivalent to  $revAct(\mathcal{F}, \mathcal{F}'; A, A'_0)$ . For instance, a reverse plan for booking online a room may be first calling the hotel in which the room is reserved, and then cancelling the reservation.

We can further generalize the notion of reversing by considering action sequences, 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.

**Definition 3** A sequence  $\langle A'_0, \dots, A'_{m-1} \rangle$  ( $m \geq 0$ ) of actions is a reverse plan for an action sequence  $\langle A_0, \dots, A_{k-1} \rangle$  ( $k > 0$ ), if, for all  $\mathcal{F}$  and  $\mathcal{F}'$ , the following formula is true:

$$\begin{aligned} multiRev(\mathcal{F}, \mathcal{F}'; [A_0, \dots, A_{k-1}], [A'_0, \dots, A'_{m-1}]) = \\ \exists_{i=0}^k \mathcal{F}_i (\mathcal{F} \equiv \mathcal{F}_0 \wedge tr_k(\mathcal{F}, A) \wedge \mathcal{F}' \equiv \mathcal{F}_k) \supset \\ \forall_{j=0}^m \mathcal{F}'_j \exists_{h=1}^m \mathcal{F}''_h \left( \mathcal{F}'_0 \equiv \mathcal{F}'_0 \supset \right. \\ \left. \bigwedge_{t=0}^{m-1} (tr_t(\mathcal{F}', A') \supset tr(\mathcal{F}'_t, A'_t, \mathcal{F}''_{t+1})) \right. \\ \left. \wedge (tr_m(\mathcal{F}', A') \supset \mathcal{F}''_m \equiv \mathcal{F}') \right). \end{aligned}$$

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

So far, a reverse plan has been defined for an action sequence at any state reachable by that sequence. However, at some such states, an action sequence may not admit any reverse plan. That is, an action sequence may have a reverse plan under some conditions, that do not necessarily hold at every reachable state. Furthermore, if some information about the state which we want to reach by reversing actions is available, e.g., values of some fluents obtained by sensing, then a reverse plan might be possible depending on this information. To make execution reversals applicable in such situations, we generalize reverse plans to “conditional reverse plans” as follows.

**Definition 4** A sequence  $\langle A'_0, \dots, A'_{m-1} \rangle$  ( $m \geq 0$ ) of actions is a  $\phi; \psi$ -reverse plan for an action sequence  $\langle A_0, \dots, A_{k-1} \rangle$  ( $k > 0$ ) if, for any  $\mathcal{F}$  and  $\mathcal{F}'$ , the formula

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

is true, where  $\phi(\mathcal{F}')$  is over  $\mathcal{F}'$  and  $\psi(\mathcal{F})$  over  $\mathcal{F}$ .

For the case where  $\psi(\mathcal{F}) \equiv \top$ , we simply write  $\phi$ -reverse plan for  $\phi; \psi$ -reverse plan. For instance, a conditional reverse plan for booking a room may be first calling the hotel in which the room is reserved, and then cancelling the reservation, with the condition that another room is available at another hotel.

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

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

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

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

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

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

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

## 4 Complexity Results

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

(P1) for two given action sequences  $AS$  and  $R$ , and given formulas  $\phi$  and  $\psi$  over fluent symbols, recognizing whether  $R$  is a  $\phi; \psi$ -reverse plan for  $AS$ ;

(P2) for a given action sequence  $AS$ , deciding whether there exist an action sequence  $R$  of a polynomially bounded length, and formulas  $\phi$  and  $\psi$  over fluent symbols, such that  $R$  is a  $\phi; \psi$ -reverse plan for  $AS$ , and that  $\phi(S')$  holds for some state  $S'$  reached by  $AS$  from some state  $S$  such that  $\psi(S)$  holds;

(P3) for a given action sequence  $AS$  and formulas  $\phi$  and  $\psi$  over fluent symbols, deciding whether there exists an action sequence  $R$  of polynomially bounded length such that  $R$  is a  $\phi; \psi$ -reverse plan for  $AS$ .

Problem	$ R  = 1$	$ R  = 2$	$ R  > 2$
(P1)	coNP	$\Pi_2^p$	$\Pi_2^p$
(P2)	$\Sigma_2^p$	$\Sigma_2^p$	$\Sigma_3^p$
(P3)	$\Sigma_2^p$	$\Sigma_3^p$	$\Sigma_3^p$

Table 1: Complexities of (P1)–(P3), in terms of completeness

For our discussion of the computational complexities of these problems, recall the following sequence of classes from the polynomial hierarchy: First,  $\Sigma_0^p = \Pi_0^p = P$ ; and for all  $k \geq 1$ ,  $\Sigma_k^p = NP^{\Sigma_{k-1}^p}$  and  $\Pi_k^p = coNP^{\Sigma_{k-1}^p}$ . Each complexity class at each level  $k$  ( $k \geq 1$ ) of the hierarchy, includes all complexity classes at lower levels. For further background on complexity, we refer the reader to [Papadimitriou, 1994].

The complexity results for problems (P1)–(P3) are summarized in Table 1. According to these results, checking whether an action sequence is a  $\phi; \psi$ -reverse plan for another action sequence (i.e., (P1)) is easier than finding a  $\phi; \psi$ -reverse plan for an action sequence (i.e., (P2) and (P3)). Finding a  $\phi; \psi$ -reverse plan, where  $\phi$  and  $\psi$  are given is harder than finding a  $\phi; \psi$ -reverse plan for arbitrary  $\phi$  and  $\psi$  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  $\Sigma_3^p$ -completeness of (P2) and (P3) is due to the following intermingled sources of complexity:

(i) the exponentially many action sequences  $R$  of a polynomially-bounded length and, in case of (P2), the exponentially many formulas  $\phi$  and  $\psi$  which need to be considered,

(ii) the test that for all states  $S$  and  $S'$  such that  $\phi(S')$  and  $\psi(S)$  hold and  $S'$  is reached from  $S$  after execution of  $AS$ , every execution of  $R$  which starts in  $S'$  ends in  $S$ , and (iii) the test that each partial execution of  $R$  starting in some state  $S'$  as in (ii) can be continued with the next action, i.e., the execution is not “stuck.”

Membership of problem (P1) in  $\Pi_2^p$  is straightforward from Definitions 3 and 4. The  $\phi; \psi$ -reverse plan property is easily rewritten to a prefix quantified Boolean formula (QBF) with  $\forall \exists$  pattern; evaluating such a formula is well-known to be in  $\Pi_2^p$ . Problem (P3) is thus in  $\Sigma_3^p$ , since a  $\phi; \psi$ -reverse plan can first be guessed and then checked with a  $\Pi_2^p$  oracle. In problem (P2),  $\phi$  and  $\psi$  are w.l.o.g. conjunctions of literals and can be guessed along with the plan; the extra condition is checkable with an NP oracle. Hardness is shown by suitable reductions of evaluating QBFs.

When limiting the length of the reverse plan, some quantifiers vanish. Informally, when  $|R| = 1$ , source (iii) disappears, and similarly when  $|R| = 2$  for (P2). The reason is that if  $R = \langle A_1 \rangle$  and if the current state  $S'$  and the state  $S$  to which we want to get back are known, then in the light of (ii) we just need to check whether  $\langle S', A_1, S \rangle$  is a valid transition, which is polynomial. In the case of (P2) and  $R = \langle A_1, A_2 \rangle$ , we similarly just need to check after reaching  $S''$  from  $S'$  by executing  $A_1$  whether  $\langle S'', A_2, S \rangle$  is a valid transition. Combined with other properties, this yields the  $\Sigma_2^p$  upper bound.

In problems (P1) and (P3), we do not check that the formulas  $\phi$  and  $\psi$  are actually satisfied at some states  $S'$  and

$S$ , respectively, such that  $S'$  is reached from  $S$  by execution of  $AS$  (if no such states exist, the problem is trivially solved). Checking this condition changes the complexity of (P1) when  $|R| = 1$  from coNP to  $D^P$  (which is the “conjunction” of NP and coNP); 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 bounded by a constant) and contains no parallel actions, and  $\phi, \psi$  are formulas from a polynomial size set of formulas, then only a polynomial number of candidates for  $\phi; \psi$ -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  $\Sigma_2^p$ -complete.

**Tractable cases.** Also tractability can be gained in certain cases. For example, if  $\phi$  and  $\psi$  are conjunctions of literals which have a single model and the description of transitions  $tr(\mathcal{F}, A, \mathcal{F}')$  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  $\phi; \psi$ -reverse plan without parallel actions for a short action sequence is feasible in polynomial time. Thus in particular, reversal of the current state in the execution of an action sequence is possible in polynomial time under these conditions.

## 5 Computation of Reverse Plans

We compute reverse plans in the spirit of knowledge compilation [Cadoli and Donini, 1997]: 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 later, such a construction of a reverse plan for an action sequence, from the reverse plan library, can be done efficiently.

We define reverse plan items and libraries as follows.

**Definition 5** A reverse plan item (RPI) is a tuple of the form  $(AS, R, \phi, \psi)$  such that  $R$  is a  $\phi; \psi$ -reverse plan for the (nonempty) action sequence  $AS$ , where  $\phi = \phi(\mathcal{F})$  and  $\psi = \psi(\mathcal{F})$ . An RPI is single-step, if  $|AS| = 1$ , i.e.,  $AS$  consists of a single action, and unconditional, if  $\phi = \psi = \text{true}$ .

**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).

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  $(AS, R, \phi, \psi)$  by

**Algorithm REVERSE**( $AS, \Pi, L$ )

**Input:** Action sequence  $AS = \langle A_0, \dots, A_{i-1} \rangle, i \geq 0$ ,  
sequence of formulas (percepts)  $\Pi = \pi_0(\mathcal{F}), \dots,$   
 $\pi_i(\mathcal{F})$ , reverse plan library  $L$ ;  
**Output:** Reverse plan  $RP$  for  $AS$  from  $\Pi$  and  $L$  or “no”

(01) **for each**  $j = 0, \dots, i-1$  **do**  $S[j] := \perp$ ;  
(02)  $S[i] := \top$ ; /\* trivially,  $S_i$  is reversible to itself \*/  
(03)  $RP := \text{REVERSE1}(i)$ ;  
(04) **if**  $RP = \text{“no”}$  **then return** “no”  
(05) **else return**  $(RP, S[0])$

Figure 1: Algorithm REVERSE to compute execution reversals using a multi-step plan library.

solving QBFs, which has been proposed as a problem solving method in the planning domain earlier, e.g., [Rintanen, 1999]. Another possibility is to reduce the problem to solving conformant planning problems defined relative to a modification of  $D$ . Due to space reasons, we cannot give the details, and instead focus on the more interesting problem of online reverse plan computation.

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

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

2. (Partial) information about the current state  $S_i$  is available, expressed by a formula  $\pi_i(\mathcal{F})$  such that  $S_i$  is one of its models, but no information about the past states. In this case, we can also make use of conditional RPIs.

3. (Partial) information about the whole execution history is available, formalized in terms of a sequence  $\Pi = \pi_0, \dots, \pi_i$  of formulas over fluent symbols, such that the state  $S_j$  is a model of  $\pi_j(\mathcal{F})$ , for each  $j = 0, 1, \dots, i$ . Here, we might exploit an even larger set of RPIs.

Clearly, scenario 3 generalizes the other ones; due to space limitations, we thus focus here on this general scenario.

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  $L = \{(\langle A, B \rangle, \langle D \rangle, \phi_1, \top), (\langle C \rangle, \langle E \rangle, \phi_2, \top), (\langle A \rangle, \langle F \rangle, \phi_3, \top), (\langle B, C \rangle, \langle G \rangle, \phi_4, \top)\}$ . 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,  $\phi_1$  might be false at the state resulting from reversing  $C$  by  $E$ , while, in the latter case,  $\phi_3$  might be true at the state resulting from reversing the action sequence  $\langle B, C \rangle$  by the action  $G$ . Thus, 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 1.

The auxiliary array  $S$  in the algorithms is used for keeping

**Algorithm REVERSE1**( $j$ )

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

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

Figure 2: Algorithm REVERSE1, in the scope of REVERSE.

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

The algorithm REVERSE starts constructing a reverse plan for an action sequence  $\langle A_0, \dots, A_{j-1} \rangle$  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 in reverse order. If the node describes an action sequence  $As$  such that  $(As, R, \phi, \psi)$  is in  $L$ , then the node is linked to a list of all RPIs of form  $(As, R', \phi', \psi')$  in  $L$ .

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

**Theorem 3** (i)  $\text{REVERSE}(AS, \Pi, L)$  has running time  $O(|AS|(|L| \cdot \text{eval}_{\max}(\mathcal{A}) + \min(AS_{\max}(L), |AS|)))$ , where  $\text{eval}_{\max}(\Pi, L)$  bounds the time to evaluate  $\pi_j \supset \phi$  and  $\pi_j \supset \psi$  for any  $\pi_j$  in  $\Pi$  and formulas  $\phi, \psi$  in  $L$ ; and  $AS_{\max}(L) = \max\{|As| \mid (As, R, \phi, \psi) \in L\}$ .

(ii)  $\text{REVERSE}(AS, \Pi, L)$  correctly outputs, relative to  $L$ , a reverse plan  $RP$  for  $AS$  and  $\Pi$  or it determines that such a reverse plan does not exist.

**Corollary 4**  $\text{REVERSE}(AS, \Pi, L)$  is polynomial, if all percepts in  $\Pi$  are DNFs and all formulas in  $L$  are  $k$ -term DNFs, i.e.,  $\bigvee_{j=1}^k t_{i,j}$  where  $k$  is bounded by a constant, or CNFs.

We remark that in an application setting,  $|AS|$  as well as reverse plan  $R$  are expected to be small (bounded by a constant) and percepts  $\pi_i$  and the formulas  $\phi, \psi$  consist of a few literals. In this case, the running time is  $O(|L|)$  i.e., linear in the size of the reverse plan library  $L$ . If, moreover, only few of the entries in the reverse plan library match, then the running time can be drastically shorter.

## 6 Related Work

Our work on undoing the execution of an action sequence has been partly motivated by [Hayashi *et al.*, 2002; 2004], where the user has to provide the reversal information. Here, we describe a method which allows for automatic discovery of this knowledge. Moreover, we describe a flexible online assembly of reverse plans from a reverse plan library. While [Hayashi *et al.*, 2002; 2004] just consider single actions and associated reverse actions, this library may contain arbitrary conditional action sequences, which the reverse plan algorithm can flexibly use. Our work is further motivated by approaches to plan recovery in logic-based monitoring frameworks [De Giacomo *et al.*, 1998; Soutchanski, 1999; 2003; Fichtner *et al.*, 2003]. However, they either do not consider action reversals or define it in a different way, usually combined with goal reachability.

The idea of backtracking for recovery in execution monitoring is similar in spirit to “reverse execution” in program debugging [Zelkowitz, 1973; Agrawal *et al.*, 1991], where all actions are undone to reach a “stable” state. Our method is more general, since no execution history is required a priori. Undoing and redoing actions on the database is at the heart of recovery in database management systems. However, also in this context, a log of the action history is available, and so the problem is significantly different.

The complexity of planning in different action languages and the framework considered here has been studied e.g. in [Baral *et al.*, 2000; Liberatore, 1997; Turner, 2002; Rintanen, 1999; Eiter *et al.*, 2004]. Conformant planning is deciding, given an action domain and formulas  $init(\mathcal{F})$  and  $goal(\mathcal{F})$  describing the initial state (not necessarily unique) and a goal state, respectively, whether there exists some action sequence  $AS$  whose execution in every initial state makes  $goal$  true. This problem is related to finding a reverse plan, and has similar complexity for plans of polynomial length ( $\Sigma_3^P$ -completeness). However, the problem is different: In action reversal, we lack a (known) goal to establish. Moreover, conformant planning is  $\Sigma_3^P$ -complete already for plans of length 1, and recognizing conformant plans of this length is  $\Pi_2^P$ -complete [Turner, 2002], differing from the results in Table 1.

## 7 Conclusion

We formally defined undo actions and reverse plans for an action sequence, in the logic-based framework for action representation from [Turner, 2002]. As we have shown, determining an undo action or reverse plan for an action sequence is intractable in general (more precisely, complete for the class  $\Sigma_2^P$  respectively  $\Sigma_3^P$  in the Polynomial Hierarchy). The intractability is explained, on the one hand, by the intractability of propositional logic underlying the framework, and, on the other hand, by the intrinsic complexity of non-determinism; nonetheless, tractability is gained under suitable restrictions. To cope with intractability, we presented a knowledge compilation approach by which undo actions and reverse plans can be efficiently constructed (under suitable conditions, in linear time) from a reverse plan library. An implementation of

the compilation algorithms, including the generation of conditional reverse plan libraries, is currently in progress.

## References

- [Agrawal *et al.*, 1991] H. Agrawal, R. A. DeMillo, and E. H. Spafford. An execution backtracking approach to program debugging. *IEEE Software*, 8(3):21–26, 1991.
- [Baral *et al.*, 2000] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1-2):241–267, 2000.
- [Cadoli and Donini, 1997] M. Cadoli and F. M. Donini. A Survey on Knowledge Compilation. *AI Communications*, 10(3-4):137–150, 1997.
- [De Giacomo *et al.*, 1998] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. KR*, pp. 453–465, 1998.
- [Eiter *et al.*, 2006] T. Eiter, E. Erdem, and W. Faber. Undoing the effects of action sequences. Tech. Report INFSYS RR-1843-04-05, TU Wien, A-1040 Vienna, Austria, 2006.
- [Eiter *et al.*, 2004] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *TOCL*, 5(2):206–263, 2004.
- [Fichtner *et al.*, 2003] M. Fichtner, A. Großmann, and M. Thielscher. Intelligent execution monitoring in dynamic environments. *Fund. Inform.*, 57(2–4), 2003.
- [Gelfond and Lifschitz, 1998] M. Gelfond and V. Lifschitz. Action Languages. *ETAI*, 2(3-4):193–210, 1998.
- [Giunchiglia *et al.*, 2004] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic Causal Theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.
- [Hayashi *et al.*, 2002] H. Hayashi, K. Cho, and A. Ohsuga. Mobile agents and logic programming. In *Proc. MA*, pp. 32–46, 2002.
- [Hayashi *et al.*, 2004] H. Hayashi, K. Cho, and A. Ohsuga. A new HTN planning framework for agents in dynamic environments. In *Proc. CLIMA*, pp. 108–133, 2004.
- [Liberatore, 1997] P. Liberatore. The complexity of the language  $\mathcal{A}$ . *Electron. Trans. Artif. Intell.*, 1:13–38, 1997.
- [Papadimitriou, 1994] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [Rintanen, 1999] J. Rintanen. Constructing Conditional Plans by a Theorem-Prover. *JAIR*, 10:323–352, 1999.
- [Son and Baral, 2001] T.C. Son and C. Baral. Formalizing Sensing Actions – A Transition Function Based Approach. *Artificial Intelligence*, 125(1–2):19–91, 2001.
- [Soutchanski, 1999] M. Soutchanski. Execution monitoring of high-level temporal programs. In *Proc. IJCAI Workshop on Robot Action Planning*, 1999.
- [Soutchanski, 2003] M. Soutchanski. High-level robot programming and program execution. In *Proc. ICAPS Workshop on Plan Execution*, 2003.
- [Turner, 2002] H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. JELIA*, pp. 111–124.
- [Zelkowitz, 1973] M. V. Zelkowitz. Reversible execution. *Communications of ACM*, 16(9):566, 1973.