Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Practical Undoability Checking via Contingent Planning

Jeanette Daum, Álvaro Torralba, Jörg Hoffmann
Saarland University
Saarbrücken, Germany
s9jedaum@stud.uni-saarland.de ;
{lastname}@cs.uni-saarland.de

Patrik Haslum, Ingo Weber
NICTA and ANU/UNSW
Canberra/Sydney, Australia
{firstname.lastname}@nicta.com.au

Abstract

We consider a general concept of undoability, asking whether a given action can always be undone, no matter which state it is applied to. This generalizes previous concepts of invertibility, and is relevant for search as well as applications. Naïve undoability checking requires to enumerate all states an action is applicable to. Extending and operationalizing prior work in this direction, we introduce a compilation into contingent planning, replacing such enumeration by standard techniques handling large belief states. We furthermore introduce compilations for checking whether one can always get back to an at-least-as-good state, as well as for determining partial undoability, i. e., undoability on a subset of states an action is applicable to. Our experiments on IPC benchmarks and in a cloud management application show that contingent planners are often effective at solving this kind of problem, hence providing a practical means for undoability checking.

Introduction

One essential property of actions is *undoability*, whether or not their effects can be undone. Informally, an action a is undoable iff whenever a is applied in a state s, there exists a plan for returning to s from the resulting state. Undoability is relevant for a variety of search algorithms, because only non-undoable actions may lead to dead-end states. The special case of undoability in a single step (termed *invertibility* by Koehler and Hoffmann, 2000; and *symmetry* by Jonsson *et al.*, 2000), for which syntactic sufficient conditions exist, has been exploited, as has the related concept of *reversibility*, defined as always being able to return to the initial state (Chen and Giménez 2010). However, no practical support for recognizing undoability (or reversibility) in the general case has been provided.

Undoability is also important in a range of planning applications: For example, Williams and Nayak's (1997) reactive planner for autonomous spacecraft control had to meet the requirement that non-undoable actions can be used only when repairing failures, not in normal operation. Weber et al. (2012; 2013) argue that undoability checking is important in cloud management, as a tool for administrators dealing with (a complex and ever-growing set of) fixed APIs whose operations may have irreversible side effects depending on context. For example, many operations (adding a slave to a database server, scaling up or down an instance) require stopping and restarting a server, which may overwrite properties not restorable (using the API) later on. Weber et al. employ a PDDL-based undoability checker that naïvely enumerates the states an action is applicable in. They also propose a simple relevance pruning technique making this enumeration feasible in their domain, yet sacrificing soundness: undoability cannot be concluded with certainty.

However, state enumeration can be avoided by drawing on the power of planning under uncertainty to deal with large belief states. (This was also observed by Eiter et al. (2007; 2008), who proposed, but did not evaluate, a QBF encoding of bounded-length conformant planning for a related problem.) We devise a practical compilation of undoability checking into contingent planning. Given an action a to check, the initial belief state is the set of possible outcome states s' for all states s that a can be applied in. The goal is to get back, from every s', to the respective previous state s. Contingent planning allows the undo-plan to observe and branch on "environment variables", not touched by the action. This is required to decide undoability exactly, as different undo-actions may be needed in different states.

It is sometimes not possible, but also not necessary, to find an undo-plan that returns exactly to s for every state s. Hence, we also present compilations for two useful variations of undoability. First, whether a plan exists to return to a state that is at least as good as s, in terms of goal reachability. With no negative preconditions or goals, a state with more facts true is always better. We name this property *rectifiability*. Second, if an action is not undoable in every state, we determine *partial* undoability/rectifiability, by identifying a *subset* of states on which the action is undoable/rectifiable. Experiments on IPC benchmarks as well as Weber et al.'s cloud management application shows deciding undoability is feasible in many cases, though scalability is problematic in some domains.

Background

We base our formalization on the STRIPS framework. STRIPS planning tasks are tuples $\Pi = (F, A, I, G)$ of *facts*, *action set*, *initial state* $I \subseteq F$, and *goal* $G \subseteq F$. Each $a \in A$ is a triple $(pre(a), add(a), del(a))$ of *precondition*, *add list*, and *delete list*, each a subset of F. For simplicity, WLOG we

assume that $add(a) \cap pre(a) = \emptyset$ and $add(a) \cap del(a) = \emptyset$.

A *state* s is a subset of facts (those true in s). Action a is *applicable* to s if $pre(a) \subseteq s$. In that case, the outcome of applying a to s is denoted $s[\![a]\!]$, given by $s[\![a]\!] := (s \cup add(a)) \setminus del(a)$. Applicability for an action sequence $\langle a_1, \ldots, a_n \rangle$, and the outcome state $s[\![\langle a_1, \ldots, a_n \rangle]\!]$, are defined in the straightforward manner.

We say that state s is *reachable* if there exists an action sequence $\langle a_1, \ldots, a_n \rangle$ so that $I[\![\langle a_1, \ldots, a_n \rangle]\!] = s$. For action a, by $S[a]$ we denote the set of states s (including unreachable ones) to which a is applicable. We say that $p \in F$ is *mutex* with $q \in F$ if there exists no reachable state s such that $\{p, q\} \subseteq s$ (e. g. (Blum and Furst 1997; Fox and Long 1998; Rintanen 2000; Gerevini and Schubert 2001)).

We will use contingent planning (e. g. (Peot and Smith 1992)) to encode undoability of STRIPS actions. Precisely, for our purposes here a *contingent planning task* takes the form (F, A, ϕ_I, G). The *initial belief* ϕ_I is a propositional CNF formula over the atoms F. The *initial belief state* b_I is the set of states s where $s \models \phi_I$. The action set $A = A_r \cup A_o$, $A_r \cap A_o = \emptyset$, distinguishes *regular actions* A_r and *observation actions* A_o. Observation actions $a \in A_o$ are pairs $(pre(a), obs(a))$ where $pre(a)$ is as above, and $obs(a) \in F$. For regular actions, in addition to $pre(a)$, $add(a)$, and $del(a)$ as above, we allow conditional effects because these will be needed in our encoding: Each action $a \in A_r$ has a set $E(a)$ of conditional effects $e = (con(e), add(e), del(e))$. The outcome of applying a to s is defined as $s[\![a]\!] := (s \cup add(a) \cup \bigcup_{e \in E(a), con(e) \subseteq s} add(a)) \setminus (del(a) \cup \bigcup_{e \in E(a), con(e) \subseteq s} del(a))$. (In our actual encodings, conditional effects with contradictory adds/deletes have mutex conditions, so never occur together.)

Action a is applicable to belief state b if a is applicable to all $s \in b$. Applying a regular action a to b yields $b[\![a]\!] := \{s[\![a]\!] \mid s \in b\}$. Applying an observation action a with $obs(a) = p$ to b splits b into two branches, $b[\![a, p]\!] := \{s \mid s \in b, s \models p\}$ and $b[\![a, \neg p]\!] := \{s \mid s \in b, s \models \neg p\}$; $b[\![a, l]\!]$ may be read as "applying a and observing l". An action tree T *solves* a belief state b if either of the following three cases holds: 1. T is empty and $\forall s \in b : s \models G$; 2. the root node of T contains a regular action $a \in A_r$ applicable in b, has a single child, and removing a from T yields a tree that solves $b[\![a]\!]$; or 3. the root node of T contains an observation action $a \in A_o$ applicable in b, has exactly two children, of which one is labeled with $obs(a)$ and the other is labeled with $\neg obs(a)$, and for each $l \in \{obs(a), \neg obs(a)\}$, the sub-tree rooted at the respective child solves $b[\![a, l]\!]$. T is a *plan* if it solves b_I.

Undoability

We first examine the most basic concept of undoability:

Definition 1 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, and $a \in A$ an action. Let a be an action, $s \in S[a]$ a state and $s' = s[\![a]\!]$: a is* undoable *in s if there is an action sequence $\overleftarrow{a_{s,s'}}$ such that $s'[\![\overleftarrow{a_{s,s'}}]\!] = s$. We say that a is* undoable *if it is undoable for all reachable states $s \in S[a]$, and we say that a is* universally undoable *if it is undoable for all $s \in S[a]$ (including the unreachable ones).*

The distinction between "undoable" (reachable states) and "universally undoable" (arbitrary states) is important. The former is appropriate when interested specifically in the instance Π. The latter is valid across *any* instance of the domain that contains the actions involved, i. e., a itself as well as the actions used in the undo sequences $\overleftarrow{a_{s,s'}}$; for example, this is true of all instances resulting from Π by changing only the initial state and goal. Furthermore, testing undoability would require to know everything about reachability in Π in the first place. Hence the method we propose tests universal undoability, which is interesting in its own right, and which serves as a sufficient criterion for undoability.

Previous work (Koehler and Hoffmann 2000; Jonsson *et al.* 2000) considered the special case of undoability by a single-step plan. This property has a syntactic sufficient condition: a is *invertible* if (1) each fact in $add(a)$ is mutex with at least one fact in $pre(a)$; (2) $del(a) \subseteq pre(a)$; and (3) there is an action $\overline{a} \in A$ such that $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$, $add(\overline{a}) = del(a)$, and $del(\overline{a}) = add(a)$. For example, a "move(l1 l2)" action a in Gripper is invertible through $\overline{a}$ = "move(l2 l1)". A "fly(b c1 c2 l1 l2)" action in ZenoTravel is not invertible, but is universally undoable by "refuel(b c2 l1 l2), fly(b c2 c1 l1 l2), refuel(b c1 l1 l2)".

Undoability relates to *reversibility*, as has been considered in several algorithmic and structural investigations of planning (e.g., Chen and Giménez 2010; Katz *et al.* 2013). Reversibility requires that, for any reachable state s, there exists an action sequence $\overleftarrow{a}$ such that $s[\![\overleftarrow{a}]\!] = I$. This is equivalent to undoability of all actions:

Proposition 1 *A STRIPS task $\Pi = (F, A, I, G)$ is reversible if and only if every $a \in A$ is undoable.*

Proof: For the if part, say $s \in S[a]$ is reachable. As Π is reversible, we have $\overleftarrow{a}$ such that $s[\![a]\!][\![\overleftarrow{a}]\!] = I$. Attaching to $\overleftarrow{a}$ the action sequence leading from I to s shows the claim. For the only-if part, say s is reached from I via $\langle a_1, \ldots, a_n \rangle$, and denote by s_i the state after execution of action a_i. Then $\langle \overleftarrow{a_{s_{n-1},s_n}}, \ldots, \overleftarrow{a_{s_2,s_1}}, \overleftarrow{a_{s_1,I}} \rangle$ reverts s to I. □

This result is useful because, previously, no automatic method for detecting reversibility was known, beyond the trivial case where all actions are invertible. Our universal-undoability test, a sufficient criterion for action undoability, yields with Proposition 1 a much more powerful method.

Undoability Checking

We consider the problem of checking whether an action a is universally undoable, i. e., undoable for all $s \in S[a]$. Our compilation uses initial state uncertainty to formulate this test compactly. In the compiled planning task, the outcome states s' become the initial belief, and the goal ensures that, for every outcome state s', the contingent plan leads us back to the previous state s corresponding to s'. To indicate these roles, we denote the initial belief of the compiled task by $\phi_{s'}$, denote its possible initial states by s', and denote its goal by G_s. We refer to plans for the compiled task as *undo plans*.

Assume the viewpoint of wishing to check whether a is undoable in s, without actually knowing what s is beyond the fact that a is applicable in s, i. e. $s \in S[a]$. What do we

know about the outcome state $s' := s[\![a]\!]$? We know that $add(a)$ and $pre(a) \setminus del(a)$ must be true, and we know that $del(a)$ must be false. What we do *not* know are the values of the *environment variables* $F^E(a) := F \setminus (pre(a) \cup add(a) \cup del(a))$ not touched by a at all. As for s itself, we know that $pre(a)$ must be true, but we do not know the values of the environment variables, nor those of the *overwritten variables* $F^O(a) := add(a) \cup (del(a) \setminus pre(a))$ which are set by a's effects but are not constrained by a's precondition.

We handle overwritten variables simply by enumeration, i. e., we create one compiled task for every value assignment to $F^O(a)$. While the number of such assignments is exponential, the exponent $|F^O(a)|$ can be expected to be small in practice.[1] In contrast, as STRIPS actions typically touch only a small fraction of the fact set F, the number of environment variables $|F^E(a)|$ will typically scale linearly with the size of the input. So these we need to handle in a compact way. Our compilation does so by maintaining, for each $p \in F^E(a)$, additional variables $WasT^p$, $WasF^p$, and Ok^p. These reflect whether p was true or false in the pre-application state s and whether in the current state of our undo plan we know that p has that same value again, respectively. To ascertain this behavior, we augment the original task's actions as follows:

Definition 2 *Let a be the action whose undoability we want to check, and α any STRIPS action. By $\alpha^{U[a]}$ we denote the action identical to α but with conditional effects $E(\alpha^{U[a]}) =$*

(i) $\{(\{WasT^p\}, \{Ok^p\}, \emptyset), (\{WasF^p\}, \emptyset, \{Ok^p\}) \mid p \in add(\alpha) \cap F^E(a)\} \cup$

(ii) $\{(\{WasF^p\}, \{Ok^p\}, \emptyset), (\{WasT^p\}, \emptyset, \{Ok^p\}) \mid p \in del(\alpha) \cap F^E(a)\}$.

If an action makes p true, then p is deemed to be Ok if it was true in s', and not-Ok if it was false in s'. The case where it makes p false is symmetric.

We need a last simple notation, associating an individual fact p with a straightforward observation action a^p:

Definition 3 *The* observer *of a fact p is the observation action a^p where $pre(a^p) = \emptyset$ and $obs(a^p) = p$.*

Such observation actions will enable the contingent undo plan to branch on the values of environment variables.

In what follows, we represent value assignments to $F^O(a)$ as subsets $O \subseteq F^O(a)$, i. e., the subset of facts assigned to true. Slightly abusing notation, we identify fact sets with fact conjunctions, and assume negated goals as a syntactic feature of contingent planning tasks. Negated goals can be compiled away with standard techniques (Gazen and Knoblock 1997); many tools (e. g. Contingent-FF) support them at PDDL level. Our compilation can then be stated as follows:

Definition 4 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, $a \in A$ an action, and $O \subseteq F^O(a)$. The* undoability-compilation *of a with respect to O is the contingent planning task $\Pi^U[a, O] := (F_{s's}, A_{s's}, \phi_{s'}, G_s)$ as follows:*

(i) $F_{s's} = F \cup \{WasT^p, WasF^p, Ok^p \mid p \in F^E(a)\}$;

(ii) $A_{s's} = A_r \cup A_o$ *where* $A_r = \{\alpha^{U[a]} \mid \alpha \in A\}$ *and* $A_o = \{a^p \mid p \in F^E(a)\}$;

(iii) $\phi_{s'} = add(a) \wedge (pre(a) \setminus del(a)) \wedge \{\neg p \mid p \in del(a)\} \wedge \bigwedge_{p \in F^E(a)} Ok^p \wedge \bigwedge_{p \in F^E(a)} (p \leftrightarrow WasT^p) \wedge \bigwedge_{p \in F^E(a)} (\neg p \leftrightarrow WasF^p)$; *and*

(iv) $G_s = pre(a) \cup \{Ok^p \mid p \in F^E(a)\} \cup O \cup \{\neg p \mid p \in F^O(a) \setminus O\}$.

Items (i) and (ii) should be clear given the discussion above. To understand the specification of $\phi_{s'}$, recall our knowledge about the outcome state s': $add(a)$ and $pre(a) \setminus del(a)$ must be true, and $del(a)$ must be false. The value of the environment variables $F^E(a)$ is unknown; $\phi_{s'}$ says that, at the start of the undo plan, every $p \in F^E(a)$ is Ok, and it sets $WasT^p$ and $WasF^p$ according to the value of p in every $s' \models \phi_{s'}$. The undo-goal G_s, finally, constrains $pre(a)$ and the environment variables as expected. Beyond that, we need to ensure that we can get back to the desired setting O of the overwritten variables in s. Note that this is the only point in the compilation where O plays a role (in s', these values are known simply because they are set by a's effects).

Theorem 1 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task. An action $a \in A$ is universally undoable if and only if, for all $O \subseteq F^O(a)$, $\Pi^U[a, O]$ is solvable.*

Proof Sketch: $\Rightarrow$: Say that a is universally undoable, and let $O \subseteq F^O(a)$. A plan T for $\Pi[a, O]$ can be constructed by observing the values of all environment variables, and attaching to each leaf node, which must correspond to a unique state $s' \models \phi_{s'}$, a corresponding sequence $\overleftarrow{a_{s,s'}}$ with $s'[\![\overleftarrow{a_{s,s'}}]\!] = s$, where s is chosen such that $s' = s[\![a]\!]$. By construction, $\overleftarrow{a_{s,s'}}$ results in a goal belief state of $\Pi[a, O]$.

$\Leftarrow$: Say that $s \in S[a]$ and $s' = s[\![a]\!]$. Let $O := F^O(a) \cap s$ be the assignment to a's overwritten variables in s. Let action tree T be a plan for $\Pi[a, O]$. Executing T on s', let $\langle a_1^U, \ldots, a_n^U \rangle$ be the resulting sequence of applicable regular actions from $A_{s's}$ which achieves G_s. Let $\overleftarrow{a_{s,s'}} := \langle a_1, \ldots, a_n \rangle$. By construction, $s'[\![\overleftarrow{a_{s,s'}}]\!] = s$. □

For undoability checking (as opposed to universal-undoability checking as in Theorem 1), our compilation can be strengthened with *invariants*, i.e., formulas that hold in all reachable states. Such invariants can be found by several known methods (e.g., Blum and Furst 1997; Fox and Long 1998; Gerevini and Schubert 2000; Scholz 2000; Rintanen 2000; Helmert 2009). If ψ is an invariant formula, we can replace $\phi_{s'}$ with $\phi_{s'} \wedge \psi$ in $\Pi^U[a, O]$. Furthermore, if $pre(a) \wedge \psi$ entails a specific value for some $p \in F^O(a)$, we can remove p from $F^O(a)$ as its value is known. In both cases, solvability of all $\Pi^U[a, O]$ remains a sufficient criterion for undoability of a. This is important because actions are often not universally undoable, but can be proven to be undoable in reachable states thanks to invariants. For example, in Barman, "leave(h c)" is undoable by "grasp(h c)", but only in reachable states. In the unreachable states s where "(holding h c)" and "(ontable c)" are both true, "leave(h c)" cannot be undone, because "grasp(h c)" deletes "(ontable c)". In other words, we cannot get back to the previous inconsistent state. Restricting $\phi_{s'}$ with the mutex invariant "$\neg$(holding h c) $\vee \neg$(ontable c)", we get rid of the

[1] An alternative is to introduce observation actions for the overwritten variables; we will get back to this below.

inconsistent states, resulting in solvable tasks $\Pi^U[a, O]$ and thus a proof of undoability.

Using contingent – rather than conformant – planning allows the undo plans to branch on environment variables. An example of why this is necessary is the "switch-off" action in IPC Satellite. To undo it, we need to apply "switch-on", which deletes calibration, so we need to recalibrate, turning the satellite from its current direction to a calibration target. The undo plan hence consists of a case distinction over the "pointing(.,.)" environment variables, encoding the direction of the satellite.[2]

In general, branching over environment variables is needed when actions have (a) different enabled preconditions or (b) different conflicts with the need to preserve environment variable values. To illustrate (b), say we wish to undo a which deletes p, a_1 adds p and environment variable q, and a_2 adds p but deletes q (and there are no other actions). Then a_1 can only be used if q was true beforehand, while a_2 can only be used if q was false beforehand.

On the other hand, branching over environment variables is not needed if neither of (a) or (b) apply, i. e., if the undo plan neither relies on the values of environment variables nor affects them. So, in many cases, the contingent undo plan makes do with very little or no branching. This is different for the overwritten variables. In principle, one could handle these in a similar manner, encoding their prior values as unknown facts p in $\phi_{s'}$ and including observation actions a^p for them (unless, that is, we can infer their values from $pre(a)$ and invariants). This would tackle the entire undoability check with a single call of contingent planning. However, in contrast to $p \in F^E(a)$, for $p \in F^O(a)$ we can *not* assume Ok^p in $\phi_{s'}$: the values of overwritten variables are affected by the action, and may have been different in the preceding state s. Hence the contingent plan would have to *establish* Ok^p for all $p \in F^O(a)$, which would necessarily involve including branches enumerating all value assignments to $F^O(a)$. It is more effective to do this enumeration outside the planner.

Anyway, as we shall see in the next section, once we modify undoability to rectifiability, i. e. require only that we can always get back to an at-least-as-good state, it suffices to consider a *single* assignment to the overwritten variables: The one where they are all assigned to true.

From Undoability to Rectifiability

We straightforwardly modify Definition 1:

Definition 5 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, and $a \in A$ an action. For $s \in S[a]$, a is* rectifiable in s *if there exists an action sequence $\overleftarrow{a_{s,s'}}$ such that $s[\![a]\!][\![\overleftarrow{a_{s,s'}}]\!] \supseteq s$. We say that a is* rectifiable *if it is rectifiable for all reachable states $s \in S[a]$, and we say that a is* universally rectifiable *if it is rectifiable for all $s \in S[a]$.*

Similar to the use of invertibility as a simple syntactic special case of undoability, prior work (Hoffmann 2005) has identified a special case of rectifiability. An action a is *at-least-invertible* if there is an action $\overline{a} \in A$ such that (1) $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$; (2) $add(\overline{a}) \supseteq del(a)$; (3) each fact in $del(\overline{a})$ is mutex with at least one fact in $pre(a)$. In this situation, (1) $\overline{a}$ is always applicable behind a, (2) re-achieves everything deleted by a, and (3) does not delete anything true prior to the application of a. Every invertible action is also at-least-invertible, so this syntactic special case generalizes the previous one. For example, the "rewind-movie" action a in the Movie domain is not undoable (and hence, in particular, not invertible) because it adds "movie-rewound", which is not deleted by any action. However, the only fact deleted by "rewind-movie" is "counter-at-zero". This can be re-achieved by the "reset-counter" action, which has empty preconditions and deletes and thus qualifies for the role of $\overline{a}$ as specified above.

It turns out that a simplified version of our previous compilation is suitable for checking rectifiability:

Definition 6 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, and $a \in A$.*

For any STRIPS action α, by $\alpha^{R[a]}$ we denote the action identical to α but with conditional effects $E(\alpha^{R[a]}) = \{(\{WasT^p\}, \{Ok^p\}, \emptyset) \mid p \in add(\alpha) \cap F^E(a)\} \cup \{(\{WasT^p\}, \emptyset, \{Ok^p\}) \mid p \in del(\alpha) \cap F^E(a)\}$.

The rectifiability-compilation *of a is the contingent planning task $\Pi^R[a] := (F_{s's}, A_{s's}, \phi_{s's}, G_s)$ as follows:*

(i) $F_{s's} = F \cup \{WasT^p, Ok^p \mid p \in F^E(a)\}$;

(ii) $A_{s's} = A_r \cup A_o$ *where* $A_r = \{\alpha^{R[a]} \mid \alpha \in A\}$ *and* $A_o = \{a^p \mid p \in F^E(a)\}$;

(iii) $\phi_{s'} = add(a) \wedge (pre(a) \setminus del(a)) \wedge \{\neg p \mid p \in del(a)\} \wedge \bigwedge_{p \in F^E(a)} Ok^p \wedge \bigwedge_{p \in F^E(a)} (p \leftrightarrow WasT^p)$; *and*

(iv) $G_s = pre(a) \cup \{Ok^p \mid p \in F^E(a)\} \cup F^O(a)$.

Relative to the previous augmented actions (Definition 2), we do not have to maintain the $WasF^p$ flags, because we don't care if an environment variable was previously false and has been made true by the undo plan. Relative to the previous compiled planning task (Definition 4), we drop the $WasF^p$ flags from $\phi_{s'}$, and we no longer need the input argument O enumerating assignments to the overwritten variables $F^O(a)$. Instead, we just constrain these variables to be true in the goal. This suffices to get the desired equivalence:

Theorem 2 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task. An action $a \in A$ is universally rectifiable if and only if $\Pi^R[a]$ is solvable.*

Proof Sketch: $\Rightarrow$: If a is universally rectifiable, to construct a plan for $\Pi^R[a]$ we observe the values of all environment variables as in Theorem 1, and attach to each leaf node corresponding to state s' a sequence $\overleftarrow{a_{s,s'}}$ where s is chosen such that $s' = s[\![a]\!]$ and $F^O(a) \subseteq s$. The latter property can be ensured WLOG because $s' = s[\![a]\!]$ is not affected by the values of the overwritten variables in s.

$\Leftarrow$: Say that $s \in S[a]$ and $s' = s[\![a]\!]$, that T is a plan for $\Pi^R[a]$, and that executing T on s' yields the sequence $\langle a_1^R, \ldots, a_n^R \rangle$ of regular actions. Then $\overleftarrow{a_{s,s'}} := \langle a_1, \ldots, a_n \rangle$ satisfies $s'[\![\overleftarrow{a_{s,s'}}]\!] \supseteq s$ as desired because, by construction, all environment variables true in s, as well as all overwritten variables, are true in $s'[\![\overleftarrow{a_{s,s'}}]\!]$. □

[2] Our approach (and implementation) finds this undo plan provided the at-least-one invariant over the "pointing(.,.)" facts has been added to $\phi_{s'}$. Else, $\phi_{s'}$ considers it possible that the satellite is not pointing anywhere, in which case one cannot recalibrate.

Intuitively, setting all overwritten variables to true is enough because, there being no need to rectify a false variable, this is the worst thing we may need to rectify. Given this, one may be tempted to think that a similar "worst-case" handling is possible for the environment variables. But this is not so because, for environment variables, being true is "bad" because the undo plan must ensure they are true again, while being false is "bad" because it enables less undo-plan actions (the latter is not so for overwritten variables whose values in s' are fixed anyhow). To illustrate, say we wish to undo a which deletes p, a_1 adds p and has the environment-variable precondition q, and a_2 adds p but deletes q (and there are no other actions). Then a_1 can only be used if q was true beforehand, while a_2 can only be used if q was false beforehand, so, like for undoability, different environment variable values may necessitate different actions.

Partial Undoability/Rectifiability

If an action is not undoable, or not rectifiable, on *all* states, naturally it is of interest to ask for *subsets* of states that work:

Definition 7 *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, $a \in A$, and $S \subseteq S[a]$. We say that a is undoable (rectifiable) in S if a is undoable (rectifiable) in all $s \in S$.*

In what follows, to avoid clumsy language we will mostly talk about undoability only, the understanding being that all claims apply also to rectifiability exactly as stated.

The main challenge of analyzing partial undoability is that we need to *find* the maximal set S such that action a can be proven undoable in every state in S. Since we have reduced undoability to the existence of a contingent plan that solves every possible initial state (i.e., b_I), partial undoability corresponds to finding a *partial contingent plan*, meaning a plan that solves a subset $S \subseteq b_I$ of the possible initial states and ensuring that the set S is maximal.

One approach to this problem comes from *online* contingent planners (Albore *et al.* 2009; Shani and Brafman 2011; Brafman and Shani 2012a; 2012b; Maliah *et al.* 2014). These tools interleave planning and execution. To simulate execution, the outcome of observation actions is chosen based on some preselected initial state. For our purposes, this results in a machinery choosing observation outcomes to narrow down the search space in the (offline) analysis of partial undoability. Given that observation actions are chosen in a manner trying to minimize their use while maximizing progress to the goal, this provides some support for obtaining large sets S of solved initial states.

However, we pursue a different approach, by compiling the maximal partial contingent planning problem into a standard contingent planning problem (i.e., generating a contingent plan that solves all initial states) but with a certain optimization objective. Given a contingent task Π, we denote this compiled task by $\Pi|_{\perp C}$. The plan cost function that we define ensures that a (complete) contingent plan for $\Pi|_{\perp C}$ with minimum cost corresponds to a partial contingent plan for Π that maximizes the coverage of initial states.

The compilation is simple: We introduce a *give-up* action a_{GiveUp} with empty precondition, effect G, and *cost* C. This allows the plan to achieve the goal anytime, at cost C. For this compilation to work as intended, we need to (1) suitably define the plan cost function, in particular the way in which the cost of AND nodes (observation actions) are aggregated from the costs of their children; and (2) select a suitable value for the constant C. We show how to do both so that a cost-optimal contingent plan for $\Pi|_{\perp C}$ applies the a_{GiveUp} action in a belief b only if b does not even have a weak plan (Cimatti *et al.* 2003); that is, if no sequence of observations and actions lead from b to the goal.

The plan cost function we use *averages* the costs of children of observation action nodes. Let a be the action at node N of a plan tree: the cost of N recursively defined as: $c(N) := c(a) + c(N')$, if a is a regular action (N is an OR node) and N' is the sole child of N; $c(N) := c(a) + \sum_{i=1}^{n} \frac{1}{n} c(N'_i)$, if a is an observation action (N is an AND node) and $N'_1, \ldots, N'_n$ are the children of N. The cost of an empty subtree is 0, and the cost of a plan tree T is the cost of the root node of T.

We note that the commonly used sum and max aggregation functions, which minimize the total cost of the plan tree and the cost of the most expensive branch, respectively, will not achieve the aim of our compilation. If Π does not have a complete contingent plan, every plan for $\Pi|_{\perp C}$ must contain a_{GiveUp}, and adding other actions can only increase plan cost. Thus, under sum or max aggregation, applying a_{GiveUp} at the root node would be an optimal plan for $\Pi|_{\perp C}$. In contrast, using average aggregation we can set the cost of the give-up action, C, so that the contingent plan gives up on belief b only if b is not solvable:

Theorem 3 *Let $\Pi = (F, A, \phi_I, G)$ be a contingent planning task. Let $B := 2^{2^{|F|}}$ be the size of the belief space, and let $C := (2^B - 1) * B + 1$. Let T_0 be a plan for $\Pi|_{\perp C}$ that is optimal under average aggregation. Then T_0 uses a_{GiveUp} only on beliefs that are not partially solvable.*

Proof Sketch: Say that b is partially solvable. Let π be a weak plan for b. Consider the action tree T constructed by following π and applying a_{GiveUp} to each observation-action outcome not contained in π. As the number of observation actions on T, and the number of regular actions in between observations, is bounded by B, $\frac{2^B-1}{2^B}(B + C)$ is an upper bound on the average-aggregation cost of T, and hence on that of T_0. So for any setting of C s.t. $C > \frac{2^B-1}{2^B}(B + C)$, T_0 cannot use a_{GiveUp} on b. The claim follows as $C > \frac{2^B-1}{2^B}(B + C)$ iff $C > (2^B - 1) * B$. □

Given this, we can test partial undoability/rectifiability by using the give-up compilation on top of our previous compilations: $\Pi^U[a, O]|_{\perp C}$ respectively $\Pi^R[a]|_{\perp C}$.[3] As average aggregation is not supported by standard contingent planning tools, we implemented it in Contingent-FF (CFF) (Hoffmann and Brafman 2005). CFF does not support action costs, so our implementation is native inside its AO* search algorithm. Also, as CFF's heuristic may return ∞ on partially solvable beliefs, we added a new heuristic (for partial

[3] We remark that, for $\Pi^U[a, O]$ and $\Pi^R[a]$ where environment variables can always be observed, the much smaller setting $C := (2^{|F^E(a)|} - 1) * 2^{|F|} + 1$ suffices for the claim of Theorem 3.

undoability only), which assumes all unknown facts are true and then runs the standard h^{FF} heuristic.

Experiments

We implemented our compilations at lifted-PDDL level, i. e., while each test pertains to a *ground action* a, the action sets of the compiled contingent planning tasks $\Pi^U[a, O]$ and $\Pi^R[a]$ are encoded as parameterized *action schemas*. This is important in practice, as otherwise the compiled task can become so large as to cause difficulties for planner parsers. We use binary mutexes and invariants found by Scholz' (2000) algorithm[4], and the FD translator (Helmert 2009) to generate the ground actions to be tested.

We focus in what follows on rectifiability testing, due to space restrictions and as rectifiability is arguably the more relevant property in practice. For the same reason, throughout we enhance our rectifiability tests with reachability information, not examining the behavior without it.

The reachability information we use are pairwise fact mutexes and exactly-1 invariants as detected by pddlcat. These are added to the initial belief $\phi_{s'}$ as previously described, which for mutexes $\neg p \vee \neg q$ where $p \in pre(a)$ is equivalent to restricting a with the implicit precondition $\neg q$. Note that in this case, $q \notin F^O(a)$ even if $q \in add(a)$. We also employ a simple form of admissible relevance pruning. Starting with $R_A := \emptyset$ and $R_F := del(a)$ where a is the action to be tested, we iteratively include into R_A actions a' whose add list intersects R_F, and into R_F facts appearing in $pre(a') \cup del(a')$. After reaching a fixed point, R_F contains all facts that an undo plan may need to make true, and R_A contains all actions that may be needed. We can thus, without affecting the outcome of the test, remove from $\Pi^R[a]$ actions and facts not in R_A and R_F, respectively, as well as observation actions for facts not in R_F.

All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time/memory cut-offs for each call of contingent planning set to 10 minutes/4 GB.

Contingent Planners

For full (as opposed to partial) rectifiability testing, we ran CFF and the offline version of CLG (Albore *et al.* 2009). PO-PRP (Muise *et al.* 2014) is not applicable because our compiled tasks are not "simple" (unknown literals occur as effect conditions). The compilation approach of CLG, on the other hand, is perfectly suitable in principle as the contingent width of our compiled tasks is 1. However, the compiled tasks are quite large and, despite the width 1, CLG's translation step often is prohibitive. Furthermore, there is a bug in CLG's implementation which sometimes leads to invalid undo plans.[5] So the main planner in our experiments here is CFF, which turned out to almost consistently outperform CLG. We include a brief summary of CLG's performance below. We run CFF without helpful actions pruning so as to allow proving unsolvability (non-rectifiability).

[4]Implemented in the pddlcat tool: http://users.cecs.anu.edu.au/~patrik/un-hsps.html.

[5]According to CLG's authors, the bug is known but would be very difficult to fix.

For partial rectifiability testing, we ran CFF with the compiled tasks $\Pi^R[a]|_{\perp C}$; as the exclusive focus here is finding plans, rather than proving their absence, we do use helpful actions pruning. Regarding the alternative approach, online contingent planning on the tasks $\Pi^R[a]$, we did not experiment with CLG because it may return invalid undo plans and we have no means for plan checking (which is itself a hard problem in contingent planning). We did run experiments with SDR (Brafman and Shani 2012b), which is more recent anyhow and represents the state of the art in this area.

IPC Benchmarks

We run all STRIPS domains from IPC'98 to IPC'14. For domains that occur several times, we use the most recent instance suite; for domains that have both an optimal and a satisficing suite, we use the optimal one (due to scaling, see below). As many benchmarks have large numbers of ground actions, rather than testing all of these, for every ground action a we first test whether a is at-least-invertible, and if so (record and) discard a. From the remaining ground actions, we randomly select 50 *per action schema*. It makes sense to organize these experiments per schema, as ground actions of different schemas typically exhibit characteristically different behavior. We omit pre-grounded domains (no action schemas), and we select only four instances from each test suite, namely the smallest-index one, the largest-index one, and two instances at equal spacing (with respect to the IPC indexing) in between. Overall, this still results in 48800 compiled contingent planning tasks. Consider Table 1.

Of the 35 domains, in 9 the at-least-invertible criterion applies to all actions (Blocksworld, Driverlog, Elevators, Gripper, Logistics, Miconic, Movie, Transport, VisitAll). In Tidybot, the compiled tasks are too large for CFF to parse. In 4 domains, nothing or not much is gained over the invertibility test (Mprime, the Pipesworlds, Tetris). In 4 domains, we find undo plans (Miconic, Satellite, Scanalyzer, Zenotravel). In the remaining 17 domains, many actions are proved non-rectifiable. In 6 domains this happens reliably (Childsnack, Floortile, Hiking, PegSol, Sokoban, TPP); in the other 11 domains, the ability to prove non-rectifiability varies depending on action schema and instance size. Note that non-rectifiability here really means "not universally rectifiable". Because the found invariants do not always rule out every unreachable state, tests are not guaranteed to be on reachable states only. A case in point is the GED domain, where all actions are in fact undoable in reachable states but many actions are found to be non-rectifiable because testing includes non-reachable states.

The "YES" cases are usually proved quickly. CFF finds most undo plans in < 1 second, except in Satellite, where it finds complex undo plans for the "switch-off" (20 regular actions, 6 observation actions, tree depth 10) and "switch-on" actions (26 regular, 6 observation, tree depth 10), in 261.2 and 265.5 seconds, respectively.

The majority of successful "NO" cases are proved either (a) by CFF's pre-processor or (b) CFF's heuristic function returning ∞ on the initial state. (a) happens iff the goal is not reachable in a relaxed planning graph even assuming all environment variables are true. (b) happens iff there is

		YES			NO					
Domain	Action schema name	%I	%Y	#A	%PN	%HN	%SN	%S	C	CA
Barman	(refill/clean/fill)-shot/shake	0	0	–	0	0	0	0	0	0
Barman	empty-shaker/pour-*	0	0	–	0	100	0	100	4	4
ChildSnack	make/put/serve*	0	0	–	100	0	0	100	4	4
Depots	lift	0	0	–	0	25	0	25	1	1
Floortile	paint-down/paint-up	0	0	–	100	0	0	100	4	4
Freecell	*home*	0	0	–	100	0	0	100	4	4
Freecell	move	63	0	–	100	0	0	100	4	4
Freecell	sendtonewcol/sendtofree	71	0	–	100	0	0	100	4	4
GED	begin-cut	0	0	–	0	100	0	100	4	4
GED	continue-cut	0	0	–	0	0	0	0	0	0
GED	continue-inv-splice-1(a/b)	0	0	–	20	35	0	56	4	2
GED	continue-splice-1	0	0	–	22	35	0	58	4	2
GED	end*/begin*/reset-1/ continue-(inv)-splice-2	0	0	–	0	50	0	50	2	2
Grid	move	0	0	–	0	0	0	0	0	0
Grid	unlock	0	0	–	100	0	0	100	4	4
Hiking	walk-together	0	0	–	100	0	0	100	4	4
Miconic	depart	0	100	3.0	0	0	0	100	4	4
Mprime	drink	86	0	–	0	0	0	0	0	0
Mprime	feast	0	0	–	0	0	0	0	0	0
NoMystery	drive	0	0	–	100	0	0	100	4	4
Parking	move-car-to-car	0	0	–	5	0	0	5	3	0
Parking	move-car-to-curb	0	0	–	0	0	0	0	0	0
Parking	move-curb-to-car	93	0	–	100	0	0	100	4	4
Pegsolitaire	(end/jump-*)-move	0	0	–	0	100	0	100	4	4
PipesNoTank	pop*/push-(start/unitpipe)	0	0	–	0	0	0	0	0	0
PipesNoTank	push-end	0	0	–	1	0	0	1	1	0
PipesTank	(push/pop)-start	0	0	–	0	0	0	0	0	0
PipesTank	(push/pop)-unitarypipe	15	0	–	0	0	0	0	0	0
PipesTank	pop-end	0	0	–	4	0	0	4	2	0
PipesTank	push-end	0	0	–	1	0	0	1	1	0
Rovers	drop	0	0	–	0	50	0	50	2	2
Rovers	sample-soil/sample-rock	0	0	–	100	0	0	100	4	4
Rovers	take-image	76	0	–	0	25	0	33	2	1
Satellite	switch-off	0	25	26.0	0	0	0	25	1	1
Satellite	switch-on	0	25	32.0	0	0	0	25	1	1
Scanalyzer	analyze-4	3	21	3.0	0	17	0	38	2	1
Scanalyzer	rotate-4	2	22	3.0	0	19	0	42	2	1
Sokoban	push-to-goal	0	0	–	21	79	0	100	4	4
Sokoban	push-to-nongoal	0	0	–	46	54	0	100	4	4
Storage	go-out/move	0	0	–	0	0	0	0	0	0
Storage	lift/drop	0	0	–	0	0	25	25	1	1
Tetris	move*	0	0	–	0	0	0	0	0	0
ThoughtFul	col-to-home	0	0	–	11	0	0	11	4	0
ThoughtFul	col-to-home-b	0	0	–	56	0	0	56	4	0
ThoughtFul	move-col-to-col	30	0	–	12	0	0	12	4	0
ThoughtFul	move-col-to-col-c	0	0	–	38	0	0	38	3	0
ThoughtFul	tal-to-*/move-col-to-col-b	0	0	–	100	0	0	100	4	4
ThoughtFul	turn-deck-a	0	0	–	14	3	0	16	4	0
ThoughtFul	turn-deck/home-to-*	0	0	–	0	0	0	0	0	0
Tidybot	*	0	0	–	0	0	0	0	0	0
TPP	load	0	0	–	0	99	0	99	4	3
TPP	unload/buy	0	0	–	100	0	0	100	4	4
Woodworking	(cut-brd/do-saw)-large	0	0	–	75	0	0	75	3	3
Woodworking	(cut-brd/do-saw)-(sm/med)	0	0	–	100	0	0	100	4	4
Woodworking	do-(immer/spray)-varnish	21	0	–	0	0	6	6	2	0
Woodworking	do-glaze	0	0	–	0	100	0	100	4	4
Woodworking	do-grind	9	0	–	4	0	6	9	2	0
Woodworking	do-plane	9	0	–	39	0	3	42	4	0
Zenotravel	fly	18	100	3.0	0	0	0	100	4	4
Zenotravel	zoom	0	100	3.6	0	0	0	100	4	4

Table 1: Rectifiability checking results with CFF. We omit action schemas where all ground actions are at-least-invertible, and domains where this applies to all schemas. YES: proved rectifiable, NO: proved non-rectifiable. %I: global fraction of at-least-invertible actions. As fractions of the actions selected per schema, across IPC instances: %Y: proved YES, %PN: proved NO by pre-process, %HN: proved NO by initial-state heuristic value, %SN: proved NO by search, %S: successfully analyzed (proved YES or NO). #A: average number of actions in undo plan. C/CA: number of IPC instances (out of the 4 selected per domain) where some/all actions are successfully analyzed.

an assignment to the environment variables for which the goals are not relaxed reachable. Few "NO" cases are proved by (c) the actual search. This is not surprising as the original problems are already large, so the yet larger compilation cannot be proved unsolvable by belief-space search. Success in the "NO" case is therefore largely determined by whether or not (a) or (b) apply. Concretely, 52% of the successful "NO" cases are of category (a), proved in mean/max time 0.1/0.9 seconds; 47% of cases are of category (b), proved in mean/max time $6.1/32.8$ seconds; and 1% of cases are of category (c), proved in mean/max time $31.2/203$ seconds.

The trivial alternative to our methods, enumeration of all $s \in S[a]$, is of course infeasible. The slightly less trivial method, doing this in the projection onto the relevant facts R_F, would work in some cases but not in general either. The average (maximum) number of relevant environment variables is 174.4 (1508) over all cases in our experiments, and is 93.2 (485) over the cases successfully tackled with CFF. Note that, while "NO" cases (a) could be captured using simple techniques, "NO" cases (b) genuinely rely on reasoning enabled by the encoding into planning under uncertainty.

The performance of CLG is almost consistently worse than that of CFF. A major advantage of CFF is its ability to handle large non-undoable instances, thanks to (a) and (b) above. For such instances, CLG's compilation either explodes in size and runs out of memory, or it does not carry over to classical planning the structure allowing CFF to easily detect unsolvability. The single case where CLG successfully analyzes more ground actions than CFF is Woodworking "do-grind", where CLG proves 29% of the ground actions non-undoable (vs. CFF's 9%)

Analyzing partial rectifiability using CFF and our give-up compilation turns out to be hard because, prior to giving up, search must explore all plan trees whose average cost is below the give-up cost C. With large values of C, this was infeasible in our experiments. Setting $C := 3$, we find partial undo-plans for Depots "lift", Grid "move", Rovers "take-image", Thoughtful "turn-deck-a" and Woodworking "do-plane". Each of these undo plans works "in half of the cases": the undo plan applies one sensing action and solves one case while giving up on the other case. The single exception is Woodworking, where the undo plans apply two sensing actions and solve three of the four possible cases.

The online contingent planner SDR finds partial undo plans in Depots, Grid, Parking, and Satellite. The only improvement over CFF is in Parking, for some instances of "move-car-to-car". For the remaining 19 domains of interest (those with non-rectifiable actions), in 7 domains we could not get SDR to run, in 8 domains SDR always runs out of time or memory, and in 4 domains SDR always gets stuck in a dead end, i. e., the preselected initial state is unsolvable and hence no undo plan is found.

Cloud Management

Weber et al.'s (2012; 2013) cloud management application is interesting from our perspective as a realistic use case, established and investigated in practically oriented prior work. For administrators, being aware of action undoability is of the essence, as it is most embarrassing to cause a lasting outage by incorrect API usage. Support for analyzing undoability is valuable as the set of APIs to deal with is complex, grows continually, and involves many operations with irre-

versible (side) effects depending on context. Furthermore, the API is the *only* method available to manipulate the resources – although Web frontends and other tools exist, they offer the same API operations through other channels.

Weber et al. devise an encoding of 35 API operations, from Amazon Web Services (AWS) Elastic Compute Cloud (EC2), into a PDDL domain file. The APIs, each encoded as an action schema, manipulate cloud resources such as virtual machines (VM) and virtual harddisks (VHD). Action effects concern resource status (e. g., start VM_1, delete VHD_2), relationship (e. g., connect VM_1 to VHD_3), or attributes (change VM_2 from *small* to *large*). Preconditions capture requirements, e. g., a VM's size can only be changed if it is *stopped*.

For the undoability tests, Weber et al. employ a classical planner. Towards that end, they first create a set of objects, essentially just one representing each type (the underlying assumption being that, within each type, objects behave symmetrically). To test an action a, they select a set of actions U_a whose adds cover $del(a)$ and whose deletes cover $add(a)$. They deem the facts F_a touched by U_a as relevant, fix all other facts (environment variables) to be false, and create one classical planning task for every pre/post state pair (s, s') over F_a where $pre(a) \subseteq s$. They consider a to be undoable iff all these planning tasks are solvable.

In brief, Weber et al.'s test is naïvely enumerative, but uses a simple pruning technique to consider only a subset $S \subseteq S[a]$ of states a is applicable to. The pruning technique is, obviously, not sound, in that S may overlook a state a is not undoable on. So the test may report a to be undoable even if that is not so. Note furthermore that the test is agnostic to reachability information, and may report a to be not-undoable due to unreachable pre/post states. One example where this happens are actions of the form "start-instance", whose precondition "(instanceStopped ?inst)" implies that no IP is associated with ?inst. Without this implicit precondition, the action is not undoable. Weber et al. propose that the API provider (or the system admin) interacts with the undoability checker to add implicit preconditions. Consequently, they have two PDDL domain files, D^W with the implicit preconditions, and $D^{W/O}$ without them.

Weber et al.'s tool is quite effective performance-wise. Using FF (Hoffmann and Nebel 2001), the total planning time across all a is 3 seconds in each of $D^{W/O}$ and D^W. On D^W, the tool correctly classifies all actions as undoable, except for "create-autoscaling-group" which is correctly classified as non-undoable. On $D^{W/O}$, the actions with missing implicit preconditions are incorrectly classified as being non-undoable. So, in the domain the tool was built for, it does not suffer from its lack of soundness, but does suffer from its lack of reachability information. Using our technology, the same tests can be performed in little more runtime, with soundness guarantee, and automatically finding the necessary reachability information.

Weber et al.'s PDDL contains some universally quantified preconditions, which we transformed to STRIPS in the usual manner (Gazen and Knoblock 1997). On these STRIPS versions of $D^{W/O}$ and D^W, Weber et al.'s tool yields the same analysis results in slightly worse runtime. We created a canonical initial state instantiating the objects with their (obvious) initial properties. Running CFF on the contingent planning tasks $\Pi^U[a]$ checking undoability takes 15.3 seconds total time for $D^{W/O}$ and 15.7 seconds for D^W, classifying all actions correctly. In particular, the actions with missing implicit preconditions in $D^{W/O}$ are determined to be undoable. Running CFF on the tasks $\Pi^R[a]$ to check rectifiability takes 17.3 (16.1) seconds for $D^{W/O}$ (D^W), and determines that "create-autoscaling-group" is (not undoable but) rectifiable. The trivial sound method, naïve enumeration of states in the projection onto the relevant facts R_F, would be feasible here, yet would incur substantial overhead: the average (maximum) size of R_F is 9.8 (22).

Related Work

The best-known prior work on undoability detection, in the AI planning community, pertains to the simple sufficient criteria, invertible and at-least-invertible actions (Koehler and Hoffmann 2000; Hoffmann 2005), that we already discussed. Hoffmann (2005) introduces the additional notion of actions whose adds are static and that delete only their own preconditions. If every action either satisfies this criterion, or is at-least-invertible, then the state space (may contain non-rectifiable actions but) cannot contain dead end states.

The only prior work on general undoability checking is that by Eiter et al. (2007; 2008). They propose an encoding into conformant planning under uncertainty. They did not implement this approach though, and it is quite different in the specifics. They propose to create a library of "reverse plan items (RPI)", action sequences $\overrightarrow{a}$ along with corresponding reverse sequences $\overleftarrow{a}$ that will always undo $\overrightarrow{a}$. Once created, the library can be used during plan execution, to identify undo plans if needed. To find the RPIs, Eiter et al. design a fixed-plan-length QBF encoding of a conformant problem guaranteeing that, for some index i along the plan, the prefix $\overrightarrow{a}$ up to i will be undone by the postfix $\overleftarrow{a}$ behind i. This forces the conformant planner to find "cycles", and thus valid RPIs. Towards partial undoability, Eiter al al. consider "conditional" RPIs, and a modified QBF encoding allowing $\overrightarrow{a}$ to mark states as "don't care", freeing $\overleftarrow{a}$ from the obligation to undo them.

For our purposes, i. e., deciding whether a specific action a is undoable, Eiter et al.'s approach could be adapted by forcing the use of a in the first time step of their QBF encoding, allowing us to look for reverse sequences $\overleftarrow{a}$ for a. However, the length of $\overleftarrow{a}$ is fixed as part of the encoding, and $\overleftarrow{a}$ is a conformant plan so will fail in those cases where different actions must be taken depending on the values of the environment variables.

Conclusion

Undoability checking can be encoded into contingent planning. In practice, perhaps not surprisingly, the problem can often be solved, even proved unsolvable, with reasonable effort, due to the particular nature of "typical" actions (which affect only a few state variables) and the corresponding undo problems. Performance does remain a challenge in some domains though. The most important challenge, in our view, re-

mains more effective *partial undoability* checking, arguably the most useful tool in practice. Towards this end, it may be worth looking into *assumption-based planning* (Davis-Mendelow *et al.* 2013), where the planner imposes restrictions on the initial belief, identifying a solvable subset of initial states. The current incarnation pertains to conformant planning, yet an extension to contingent planning and undoability checking appears promising and feasible.

Acknowledgments. This work was partially supported by the EU FP7 Programme under grant agreement no. 295261 (MEALS). NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1623–1628, 2009.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.

Ronen I. Brafman and Guy Shani. A multi-path compilation approach to contingent planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, 2012.

Ronen I. Brafman and Guy Shani. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research*, 45:565–600, 2012.

Hubie Chen and Omer Giménez. Causal graphs and structurally restricted planning. *Journal of Computer and System Sciences*, 76(7):579–592, 2010.

Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2):35–84, 2003.

Sammy Davis-Mendelow, Jorge A. Baier, and Sheila A. McIlraith. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*, 2013.

Thomas Eiter, Esra Erdem, and Wolfgang Faber. On reversing actions: Algorithms and complexity. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 336–341, 2007.

Thomas Eiter, Esra Erdem, and Wolfgang Faber. Undoing the effects of action sequences. *Journal of Applied Logic*, 6(3):380–415, 2008.

Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

B. Cenk Gazen and Craig A. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 221–233, 1997.

Alfonso Gerevini and Lenhart Schubert. Inferring state constraints in DISCOPLAN: Some new results. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00)*, pages 761–767, 2000.

Alfonso Gerevini and Lenhart Schubert. DISCOPLAN: an efficient on-line system for computing planning domain invariants. In *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 433–436, 2001.

Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009.

Jörg Hoffmann and Ronen Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pages 71–80, 2005.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

Peter Jonsson, Patrik Haslum and Christer Bäckström. Towards Efficient Universal Planning – A Randomized Approach. *Artificial Intelligence*, 117(1):1–29, 2000.

Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Who said we need to relax *all* variables? In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pages 126–134, 2013.

Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

Shlomi Maliah, Ronen I. Brafman, Erez Karpas, and Guy Shani. Partially observable online contingent planning using landmark heuristics. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 2014.

Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 2322–2329, 2014.

Mark Peot and David E. Smith. Conditional non-linear planning. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems (AIPS'92)*, pages 189–197, 1992.

Jussi Rintanen. An iterative algorithm for synthesizing invariants. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00)*, pages 806–811, 2000.

Ulrich Scholz. Extracting state constraints from PDDL-like planning domains. In *Proc. AIPS 2000 Workshop on Analyzing and Exploiting Domain Knowledge for Efficient Planning*, pages 43–48, 2000.

Guy Shani and Ronen I. Brafman. Replanning in domains with partial information and sensing actions. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2021–2026.

Ingo Weber, Hiroshi Wada, Alan Fekete, Anna Liu, and Len Bass. Automatic undo for cloud management via AI planning. In *Proceedings of the 8th Workshop on Hot Topics in System Dependability (HotDep'12)*, 2012.

Ingo Weber, Hiroshi Wada, Alan Fekete, Anna Liu, and Len Bass. Supporting undoability in systems operations. In *Proceedings of the 27th Large Installation System Administration Conference (LISA'13)*, pages 75–88, 2013.

Brian C. Williams and P. P. Nayak. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1178–1185, 1997.