# Reformulating Planning Problems: A Theoretical Point of View

**Lukáš Chrpa, Thomas Leo McCluskey and Hugh Osborne**
Knowledge Engineering and Intelligent Interfaces Research Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, t.l.mccluskey, h.r.osborne}@hud.ac.uk

## Abstract

Automated planning is a well studied research topic thanks to its wide range of real-world applications. Despite significant progress in this area many planning problems still remain hard and challenging. Some techniques such as learning macro-operators improve the planning process by reformulating the (original) planning problem. While many encouraging practical results have been derived from such reformulation methods, little attention has been paid to the theoretical properties of reformulation such as soundness, completeness, and algorithmic complexity. In this paper we build up a theoretical framework describing reformulation schemes such as action elimination or creating macro-actions. Using this framework, we show that finding entanglements (relationships useful for action elimination) is as hard as planning itself. Moreover, we design a tractable algorithm for checking under what conditions it is safe to reformulate a problem by removing primitive operators (assembled to a macro-operator).

## Introduction

AI planning (Ghallab, Nau, and Traverso 2004) deals with the problem of generating a sequence of ground actions given a set of operator schema. The actions may be generated in order to achieve a desired goal condition ("my thirst is satisfied") from some initial state, or to ground a high level task ("make me a cup of tea"). In the former area (goal achievement planning) many optimized planning engines (Hoffmann and Nebel 2001; Richter and Westphal 2008) are now available which input the planning problem in some variant of the language PDDL (Ghallab et al. 1998). These planning engines are being used as *black boxes* in applications, where the interface to the engine is the problem and domain model (referred to here as PDM) defined in the language PDDL, and the solution plan generated is the output. However, whereas specific PDMs can be solved very efficiently using this setup, it is well known that AI planning is intractable in general. Although current planning engines are very refined, there is no guarantee that they will return a solution in a reasonable length of time.



Often PDMs have their hidden specifics. For example, in the well known BlockWorld domain we have operators $UNSTACK$ and $PUTDOWN$ whose instances often appear successively in plans. Similarly, we can find out that the operator $STACK$ is used only for stacking blocks to their goal positions. One general method that has been explored to alleviate this problem is to devise a means of reformulating the input PDM by taking into account its hidden specifics, ensuring that the reformulated problem is much more efficiently solved than the original. After a solution is obtained, the output plan is transformed back to the original formulation, to retain the *black box* property of the planning engine. Creating *macro-operators* (Dawson and Siklóssy 1977) is a well known approach to reformulation which in some cases can speed up plan generation considerably (Newton et al. 2007; Botea et al. 2005). Moreover Chrpa (2010) eliminates potentially useless primitive planning operators replaced by a generated macro-operator, however it might cause that some reformulated problems become unsolvable. Another method is to eliminate actions from the planning problem altogether, such as has been done in recent work by discovering *entanglements* (Chrpa and Barták 2009). While many encouraging practical results have been derived from such reformulation methods, little attention has been paid to the theoretical properties of reformulation such as soundness, completeness, and algorithmic complexity.

In this paper we build up a theoretical framework for classical planning (deterministic, fully observable environments) on which to precisely define reformulation schemes, and investigate the combination of macro-operator creation and action elimination. Using this framework, we show that finding entanglements (relationships that are useful for guiding action elimination) is as hard as planning itself. We go on to produce conditions for a combined macro-operator creation and action elimination method that ensures the soundness and completeness of the reformulation, and show that the algorithm which operationalizes these conditions is of polynomial complexity.

## Preliminaries

PDMs can be represented in several ways. The set-theoretic representation is based on propositional logic, therefore we can define only planning problems (see definition below)

suitable for certain situations. The classical representation is based on first order (predicate) logic, therefore it is possible to define planning domains (see below) suitable for certain classes of planning problems.

In a set-theoretic representation a **planning problem** is a tuple $\Pi = \langle P, S, A, \gamma, I, G \rangle$. $P$ stands for a finite set of **atoms** (propositions), $S \subseteq 2^P$ stands for a set of **states**, $I \in S$ stands for **initial situation** (initial state), $G \subseteq P$ stands for **goal situation**. $A$ is a set of **actions**, action $a \in A$ is specified via its precondition ($pre(a) \subseteq P$), negative and positive effects ($\mathit{eff}^-(a) \subseteq P$, $\mathit{eff}^+(a) \subseteq P$). An action $a \in A$ is **applicable** in a state $s \in S$ iff $pre(a) \subseteq s$. $\gamma : S \times A \rightarrow S$ is a **transition function**, where $\gamma(s, a) = (s \backslash \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$ if $a$ is applicable in s, otherwise $\gamma(s, a)$ is undefined. $\gamma^*$ is the reflexive transitive closure of $\gamma$ and can be used for sequences of actions. A **plan** $\pi$ is a sequence of actions $\langle a_1, \ldots, a_k \rangle$. A plan $\pi$ is a **solution** of a planning problem $\Pi$ iff $G \subseteq \gamma^*(I, \pi)$.

Unlike the set-theoretic representation where atoms and actions refer to certain objects (e.g. on-a-b or stack-a-b), the classical representation (Ghallab, Nau, and Traverso 2004) defines atoms as predicates (e.g. on(X,Y)) and planning operators (e.g stack(X,Y)) instead of actions. Variable symbols (e.g. X,Y) can be substituted by constants referring to certain objects (e.g. a,b). Hence atoms and planning operators do not depend on a particular planning problem. Formally, a **planning operator** is a 4-tuple $o = (name(o), pre(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$, where $name(o)$, the **name** of the operator $o$, is an expression of the form $name(x_1, \ldots, x_k)$ where *name* is called an **operator symbol**, $x_1, \ldots, x_k$ are all of the variable symbols that appear in the operator, and *name* is unique. $pre(o)$, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are generalizations of the preconditions, negative and positive effects of the set-theoretic action (instead of being sets of propositions, they are sets of predicates).

In the classical representation **a planning domain** consists of a set of predicates and a set of planning operators. A planning problem is then specified by objects, initial and goal situation. Obtaining the set-theoretic representation from the classical representation is done by grounding of all the predicates and planning operators.

## Reformulation Scheme

There are many ways in which PDMs can be encoded. Some of the encodings might be hard for planners while others may not. For instance, Hoffmann (2011) shows how problem analysis of PDMs can shed light on how hard particular encodings of benchmark domains are. This leads to the question of whether the encodings can be reformulated in order to make the problem easier to solve.

**Definition 1.** *Let PROBS be a set of planning problems. Let PLANS* $= \{\pi \mid \pi \; is \; a \; solution \; of \; any \; \Pi \in$ *PROBS*$\}$ *be a set of plans. A* ***reformulation scheme*** *for planning is a pair of functions* $(probref, planref)$ *defined in the following way:*

- *probref* : *PROBS* $\rightarrow$ *PROBS is a* ***problem reformulation function***
- *planref* : *PLANS* $\rightarrow$ *PLANS is a* ***plan reformulation function***

*The reformulation scheme* $(probref, planref)$ *is* ***sound*** *if* $\pi' \in$ *PLANS is a solution of probref*$(\Pi) \in$ *PROBS, then planref*$(\pi')$ *is a solution of* $\Pi \in$ *PROBS.*

*The reformulation scheme* $(probref, planref)$ *is* ***complete*** *if it is sound and if* $\Pi \in$ *PROBS is solvable (i.e., there is a solution of* $\Pi$*), then probref*$(\Pi) \in$ *PROBS is also solvable.* ■

Composition of reformulation schemes do not affect soundness or completeness (Chrpa 2011). The most common and well studied kinds of reformulation scheme are adding macro-actions or eliminating some (unnecessary) actions (see below).

### Action Eliminating Scheme

Reformulating a PDM to eliminate unnecessary actions is an obvious way to try to reduce branching when searching for plans.

**Definition 2.** *Let* $(elim, id)$ *be a reformulation scheme for planning. Let id be a plan reformulation function such that for every plan* $\pi$ *id*$(\pi) = \pi$ *(identity function). Let elim be a problem reformulation function such that for every planning problem* $\Pi = \langle P, S, A, \gamma, I, G \rangle$ *and* $\Pi' = \langle P, S, A', \gamma', I, G \rangle$ *such that elim*$(\Pi) = \Pi'$*,* $A' \subseteq A$ *and* $\gamma'$ *is defined on* $S \times A'$ *and* $\gamma' \subseteq \gamma$*. Then, we say that* $(elim, id)$ *represents an* ***action eliminating scheme****.* ■

Action eliminating schemes are sound but incomplete (Chrpa 2011). Obviously, if actions are eliminated the problem may become unsolvable. If we eliminate unreachable actions (i.e., actions whose preconditions cannot be satisfied in any step of the planning process) then completeness is not affected. However, checking whether an action is unreachable in general is PSPACE-complete (we have to solve a planning problem where the action precondition is a goal situation).

Investigating whether an action can be removed from a planning problem without affecting its solvability is also PSPACE-complete. To prove this we can use the landmark theory (Hoffmann, Porteous, and Sebastia 2004). Landmarks are atoms which must be true at some point of the planning process. Similarly action landmarks are actions which must be presented in every solution. Deciding whether an atom is a landmark has been proved to be PSPACE-complete (Hoffmann, Porteous, and Sebastia 2004). We can use this to prove PSPACE-completeness of deciding an action landmark.

**Proposition 1.** *Deciding an action landmark is PSPACE-complete.*

*Proof.* The problem of deciding an action landmark can be reduced into the problem of deciding a landmark, which is known to be PSPACE-complete. Let $\{a_1, \ldots, a_n\}$ be a set of actions such that for every $1 \leq i \leq n$ $p \in \mathit{eff}^+(a_i)$. Without loss of generality we assume that $\{a_1, \ldots, a_n\}$ and $p$ are defined in some problem $\Pi$ and no other action has $p$ in its positive effects. We modify the actions $\{a_1, \ldots, a_n\}$ such that we replace $p$ in the positive effects by $p'$ (without loss of generality we assume that $p'$ is not defined in $\Pi$). Then we introduce a new action $a'$, where $pre(a') = \mathit{eff}^-(a') = \{p'\}$

and $\mathit{eff}^+(a') = \{p\}$. $a'$ is an action landmark in the modified problem if and only if $p$ is a landmark in $\Pi$. □

This result shows us that deciding whether an action can be removed (in general) is also PSPACE-complete (we cannot remove an action landmark).

### Macro-action Scheme

We say that an action $a_{1,\ldots,k}$ is a **macro-action** over the sequence of actions $\langle a_1, \ldots, a_k \rangle$ if for every $s \in S$ (in a given problem) it holds that $\gamma^*(s, a_{1,\ldots,k}) = \gamma(s, \langle a_1, \ldots, a_k \rangle)$ or both are undefined. Hence, macro-actions represent 'shortcuts' in the state space.

**Definition 3.** *Let $(\mathit{macro}, \mathit{unfold})$ be a reformulation scheme for planning. Let macro be a problem reformulation function such that for every planning problem $\Pi = \langle P, S, A, \gamma, I, G \rangle$ and $\Pi' = \langle P, S, A \cup A_m, \gamma', I, G \rangle$ ($A_m$ is a set of macro-actions) such that $\mathit{macro}(\Pi) = \Pi'$, for every $a_{1,\ldots,k} \in A_m$ such that $a_{1,\ldots,k}$ is a macro-action over the sequence of actions $\langle a_1, \ldots, a_k \rangle$ ($a_1, \ldots a_k \in A$) and $\gamma' \supseteq \gamma$ (defined on $S \times (A \cup A_m)$). Let unfold be a plan reformulation function such that for every $\pi'$ and $\pi$ such that $\mathit{unfold}(\pi') = \pi$ it holds that every macro-action (from $A_m$) in $\pi'$ is replaced by the corresponding sequence of primitive actions (from $A$) in $\pi$. Then, we say that $(\mathit{macro}, \mathit{unfold})$ represents a **macro-action scheme**.* ■

Unsurprisingly, macro-action schemes are sound and complete (Chrpa 2011). Macro-actions can be generalized in the same way as (normal) actions, i.e. macro-operators are a general form of macro-actions. Even though the reformulation scheme is defined using the set-theoretic representation there is a straightforward relation to the classical representation (e.g. if a macro-operator is added into the (classical) planning domain then in fact all its instances (macro-actions) are added into the (set-theoretic) planning problem).

## Eliminating Actions by Entanglements

Chrpa and Barták (2009) introduced entanglements as a tool for eliminating potential unnecessary actions. Entanglements are relations between planning operators and initial or goal atoms. For example, if operator *UNSTACK* is entangled with predicate $on$ (for deeper insight about BlockWorld domain, see (Slaney and Thiébaux 2001)), then the *UNSTACK* operator is used only for unstacking blocks from their initial positions (i.e., only the corresponding instances can be provided). The formal definition is provided below.

**Definition 4.** *Let $P$ be a planning problem, where $I$ is an initial situation and $G$ is a goal situation. Let $o$ be a planning operator and $p$ be a predicate ($o$ and $p$ are defined in a planning domain which is related to $P$). Operator $o$ is **entangled by init (or goal)** with predicate $p$ in planning problem $P$ if and only if $p \in \mathit{pre}(o)$ (or $p \in \mathit{eff}^+(o)$) and there is a plan $\pi$ that solves $P$ and for every action $a \in \pi$ which is an instance of $o$ and for every grounded instance $p_{gnd}$ of the predicate $p$ it holds: $p_{gnd} \in \mathit{pre}(a) \Rightarrow p_{gnd} \in I$ (resp. $p_{gnd} \in \mathit{eff}^+(a) \Rightarrow p_{gnd} \in G$).* ■

The definition of entanglement can be extended to full entanglement which applies not just to one problem only but to a whole class of problems having the same planning domain.

We say that an action $a$ (instance of an operator $o$) **violates** the entanglement by init (resp. goal) with a predicate $p$ if an instance of $p \in \mathit{pre}(a)$ is not in $I$ (resp. an instance of $p \in \mathit{eff}^+(a)$ is not in $G$).

In the language of reformulation schemes we deal with a special case of an action eliminating scheme. Formally, $(\mathit{ent\_elim}, \mathit{id})$ is **an entanglement reformulation scheme** if $(\mathit{ent\_elim}, \mathit{id})$ is an action eliminating scheme and for every planning problem $\Pi$ $\mathit{ent\_elim}(\Pi)$ is constructed in such a way that if $A$ is a set of actions in $\Pi$, then $A \setminus \{a | a \text{ violates any entangement}\}$ is a set of actions in $\mathit{ent\_elim}(\Pi)$.

The entanglement reformulation scheme is sound because it is a 'special case' of an action eliminating scheme. Completeness of the entanglement reformulation scheme is given directly from the definition of entanglement (see definition 4). However, the detection of entanglement is (in general) PSPACE-complete which is proved in the following theorem.

**Theorem 1.** *Deciding an entanglement is PSPACE-complete.*

*Proof.* The proof of PSPACE-completeness of deciding entanglements is done by reducing it to the problem of deciding an action landmark, which is also PSPACE-complete (see Proposition 1).

**entanglement by init** — Let $\Pi$ be a planning problem and $a$ be an action defined in $\Pi$. Let $o$ be a planning operator which is a general form of $a$ (i.e., $a$ is an instance of $o$). Let $p$ be a predicate which has the same variable symbols (arguments) as the operator $o$ (without loss of generality we assume that $p$ is not defined in $\Pi$). We extend $o$ in such a way that we add $p$ into $\mathit{pre}(o)$. We create a planning operator $o'$ such that $o'$ has the same variable symbols (arguments) as the operator $o$, $\mathit{pre}(o') = \mathit{eff}^-(o') = \emptyset$ and $\mathit{eff}^+(o') = \{p\}$. We also extend the initial state by adding all the instances of $p$ but one that corresponds with the action $a$ (i.e., if $\Theta$ is a substitution such that $o\Theta = a$ then $p\Theta$ is not added to the initial state). It can be seen that if the extended operator $o$ is entangled by init with $p$ in the extended $\Pi$ then $a$ is not an action landmark and vice versa (in fact if this entanglement does not hold than $a$ must appear in every solution). The extended problem is solvable even though $a$ must be present in the solution because in that case the corresponding instance of the extended $o$ must be preceded by a corresponding instance of $o'$ which gives the instance of $p$ missing in the initial state.

**entanglement by goal** — Can be addressed similarly as in the entanglement by init case. In this case the operator $o$ is extended in such a way that we add $p$ into $\mathit{eff}^+(o)$. We put all instances but one (corresponding with $a$) into the initial state as well as into the goal situation. In this case $o'$ is not necessary because $p$ is not in the precondition of $o$ thus the absence of an instance of $p$ in the initial state does not affect applicability of the corresponding instance

of $o$. It can be also seen that if the extended operator $o$ is entangled by goal with $p$ in the extended $\Pi$ then $a$ is not an action landmark and vice versa.

□

The previous theorem in fact says that deciding entanglement is as hard as planning itself. On the other hand in some domains such as BlockWorld we believe that some entanglements (e.g. *UNSTACK* with *on*) can be found in a polynomial time. Nevertheless, entanglements can be 'detected' by an approximation such as that in Chrpa and Barták (2009) which despite incompleteness produced very promising results.

## Removing Primitive Operators Replaced by Macro-operators

Chrpa (2010) introduced the idea of removing primitive operators which are replaced by a created macro-operator. Using the definitions above means that the reformulation scheme is composed from the macro-action scheme and action eliminating scheme. However, we know that the action eliminating scheme is incomplete and checking whether a problem loses completeness or not after being reformulated is in general PSPACE-complete.

Macro-actions are in fact 'shortcuts' in the state-space and removing the primitive actions may be that the intermediate state might become unreachable or the goal might become unreachable from the intermediate state. For illustration let $s_0, s_1$ and $s_2$ be states and $a_1$ and $a_2$ be actions such that $\gamma(s_0, a_1) = s_1$ and $\gamma(s_1, a_2) = s_2$. Introducing a macro-action $a_{1,2}$ (over the sequence $\langle a_1, a_2 \rangle$) gives a direct connection from $s_0$ to $s_2$. The question is in what conditions we can remove actions $a_1$ and $a_2$ without breaking completeness. If there are reversible actions (inverse actions) for $a_1$ and $a_2$ (condition 1 of Proposition 2), then $s_1$ remains reachable, and the completeness is not affected. If there are not we must somehow 'bypass' $s_1$ (condition 2 of Proposition 2). We must not start or end in $s_1$ (i.e., $s_1$ must not be an initial state, and if neither $s_0$ nor $s_2$ is a goal state, then nor can $s_1$ be one). If there is another action $a$ applicable in $s_1$, then to ensure completeness there must be an alternative action $a'$ resulting in the same state as the application of $a$ in $s_1$ which is applicable in $s_0$ (see (a,b) in Figure 1) or $s_2$ (see (c,d) in Figure 1). Similarly we must find an alternative actions for actions which leads to $s_1$ (for illustration assume reverse directions for $a$ and $a'$ in Figure 1).

**Proposition 2.** *Let $\Pi = \langle P, S, A, \gamma, I, G \rangle$ be a planning problem. Without loss of generality we assume that $a_1, a_2 \in A$ and $a_{1,2} \notin A$. Let $a_{1,2}$ be a macro-action over the sequence $\langle a_1, a_2 \rangle$. Let $A^- = \{a_1, a_2\}$ be a set of actions. We assume that one the following conditions holds for every triple of states $s_0, s_1, s_2 \in S$ such that $\gamma(s_0, a_1) = s_1$ and $\gamma(s_1, a_2) = s_2$.*

1. *There are actions $a'_1, a'_2 \in A$ such that $\{a'_1, a'_2\} \cap A^- = \emptyset$, $\gamma(s_2, a'_2) = s_1$ and $\gamma(s_1, a'_1) = s_0$.*
2. *$s_1 \not\subseteq I \wedge (G \subseteq s_1 \rightarrow (G \subseteq s_0 \vee G \subseteq s_2))$ and for every $a \in A \setminus A^-$ and $s \in S \setminus \{s_1, s_2\}$ such that $\gamma(s_1, a) = s$ (resp. $\gamma(s, a) = s_1$, $s \neq s_0$) it holds that $s = s_0$ or there is an action $a' \in A \setminus A^-$ such that $\gamma(s_0, a') = s$ or $\gamma(s_2, a') = s$ (resp. $\gamma(s, a') = s_0$, $s \neq s_0$ or $\gamma(s, a') = s_2$).*

*Let $\Pi' = \langle P, S, (A \cup \{a_{1,2}\}) \setminus A^-, \gamma', I, G \rangle$ be a planning problem ($\gamma'$ is defined according to the definitions of action eliminating and macro-action scheme). If $\Pi$ has a solution, then $\Pi'$ has also a solution.*

See (Chrpa 2011) for a proof of proposition 2

In this paper we exploit Proposition 2 in order to create a tractable algorithm for finding *complete* reformulations involving macro-operators and action elimination. Proposition 2 is the basis for Algorithm 1 which tests whether it is safe to remove primitive operators replaced by a macro-operator. If the algorithm returns true then it is safe (i.e., if the reformulated problem is unsolvable then the original one is unsolvable too). If the algorithm returns false then the reformulated problem may no longer be complete.

The algorithm first checks if the removed operators have

**Algorithm 1** Algorithm for detecting completeness of removal of the primitive operators replaced by a new macro-operator.

**Require:** Planning domain $\Sigma$, macro-operator $o_{1,2}$, primitive operators $o_1, o_2$
**if** There are operators $o'_1$ and $o'_2$ such that they are inverse to $o_1$ and $o_2$ **then**
  **return** true
**end if**
create templates of $S_0, S_1, S_2$ {see the text for details}
**if** $S_1$ can be instantiated to an initial state **then**
  **return** false
**end if**
**if** $S_1$ can be instantiated to a goal state while $S_0$ or $S_2$ cannot **then**
  **return** false
**end if**
**for all** $o \neq o_2$ applicable in $S_1$ **do**
  **if** application of $o$ in $S_1$ leads towards $S_0$ **then**
    continue
  **end if**
  find $o' \neq o_1$ applicable in $S_0$ or $S_2$ such that application of $o'$ in $S_0$ or $S_2$ has the same result that application $o$ in $S_1$ {see the text for details}
  **if** $o'$ does not exist **then**
    **return** false
  **end if**
**end for**
**for all** $o \neq o_1$ applicable in some state template $S$ such that application of $o$ leads towards $S_1$ **do**
  find $o' \neq o_2$ applicable in $S$ such that application of $o'$ leads towards $S_1$ {see the text for details}
  **if** $o'$ does not exist **then**
    **return** false
  **end if**
**end for**
**return** true

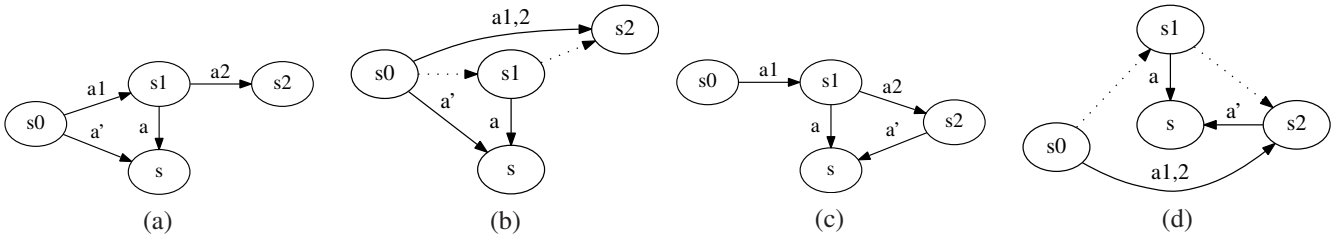


Figure 1: Replacing primitive actions (a,c) by a macro-action (b,d). Removed primitive actions are visualized by a dotted line in (b,d).

inverses (Line 1). Informally, an operator is an inverse of another if it reverses the changes of the other operator. Formally, $o$ is **inverse** to $o'$ if $\mathit{eff}^-(o) = \mathit{eff}^+(o')$, $\mathit{eff}^+(o) = \mathit{eff}^-(o')$ and $\mathit{pre}(o') \subseteq \mathit{eff}^+(o) \cup (\mathit{pre}(o) \setminus \mathit{eff}^-(o))$ ($\mathit{pre}(o)$ is determined analogously). Finding inverse operators to $o_1$ and $o_2$ (Line 1) obviously covers all situations where instances of $o_1$ and $o_2$ are involved which satisfies condition 1 of Proposition 2. Creating templates of states (see Line 4) is done as follows. **State templates** are of the form $S = \langle S^+, S^- \rangle$ where $S^+$ contains atoms that must be included in the state and $S^-$ contains atoms that must not be included in the state. For $S_0$ it must hold that $S_0 \supseteq \mathit{pre}(o_{1,2})$, therefore $S_0^+ = \mathit{pre}(o_{1,2})$ and $S_0^- = \emptyset$. For $S_1$ it must hold that:

$$S_1 \supseteq \mathit{eff}^+(o_1) \quad (1)$$

$$S_1 \cap \mathit{eff}^-(o_1) = \emptyset \quad (2)$$

$$S_1 \supseteq \mathit{pre}(o_1) \setminus \mathit{eff}^-(o_1) \quad (3)$$

$$S_1 \supseteq \mathit{pre}(o_2) \quad (4)$$

Then $S_1^+ = \mathit{eff}^+(o_1) \cup \mathit{pre}(o_2) \cup (\mathit{pre}(o_1) \setminus \mathit{eff}^-(o_1))$ and $S_1^- = \mathit{eff}^-(o_1)$. $S_2$ is created according to (1)-(3) (analogously to $S_1$ except (4)) but instead of $o_1$ we use $o_{1,2}$. A state template $S_1$ can be instantiated to an initial state $I$ if there is a ground substitution $\Theta$ such that $S_1^+\Theta \subseteq I$ and $S_1^-\Theta \cap I = \emptyset$. If $S_1$ can be instantiated to $I$ (Line 5), then obviously condition 2 of Proposition 2 is broken. A state template $S_1$ can be instantiated to a goal state ($G$ stands for a goal situation) if there is a substitution $\Theta$ such that $S_1^+\Theta \cap G \neq \emptyset$ and $S_1^-\Theta \cap G = \emptyset$. Hence, an instance of $S_1^+$ must provide at least one goal atom (obviously atom instances from $S_1^-$ must be disjoint to goal atoms) to be considered as a potential goal state. One might suggest that even if the instance of $S_1^+$ does not provide any goal atoms, then there might be an instance of $S_1$ which is the goal state. However, in this case we can simply find out that an instance of $S_0$ must also be a goal state because all the atoms which might not be available in $S_0$ are added by an instance of $o_1$ which are listed in $S_1^+$. For state templates $S_0$ and $S_2$ we have to add an additional condition, i.e., $S_0^+\Theta \supseteq S_1^+\Theta \cap G$ (analogously for $S_2$). This is because we must be sure that if the instance of $S_1$ is the goal state, then the instance of $S_0$ or $S_2$ is the goal state as well. We can also see that the additional condition means that atoms in $S_1^+$ contributing to the goal state have already been in $S_0^+$ or are not removed from $S_2^+$. Hence, if the test (Line 8) fails, then condition 2 of Proposition 2 is broken.

Applicability of an operator in a state template is checked in a similar way to checking if it can be instantiated to the goal state but in this case the substitution is unground (i.e., does not provide constants). If there is an unground substitution $\Theta$ such that $S_1^+\Theta \cap \mathit{pre}(o) \neq \emptyset$ and $S_1^-\Theta \cap \mathit{pre}(o)\Theta = \emptyset$, then $o$ is applicable in $S_1$ (these substitutions will henceforth be assumed when we compare unground sets of atoms). If $o \neq o_2$ can be applied in $S_1$ (Line 11), then the result of the application $S$ is constructed according to (1)-(3) and we also have to move all unchanged atoms (atoms which do not appeared in effects of $o$) from $S_1$. If $S^+ \subseteq S_0^+$ and $S^- \supseteq S_0^-$, then the application of $o$ in $S_1$ leads towards $S_0$ (Line 12). If so, then according to Proposition 2 (condition 2) the existence of $o$ does not affect completeness. Otherwise we have to find an operator $o' \neq o_1$ applicable in $S_0$ or $S_2$ that leads towards $S$. To be sure that $o'$ is really applicable in the state templates $S_0$ and $S_2$ we must strengthen the condition of operator applicability in such a way that $S_0^+ \supseteq \mathit{pre}(o')$ (analogously for $S_2$). Constructing $S'$ as a result of application of $o'$ in $S_0$ or $S_2$ is done analogously to the construction of $S$. If $S'^+ \supseteq S$ and $S'^- \subseteq S^-$, then we can say that the application of $o'$ in $S_0$ or $S_2$ has the same result as application of $o$ in $S_1$. If such an operator $o'$ does not exist, then condition 2 of Proposition 2 might be broken.

Checking whether an application of an operator $o \neq o_1$ in a state template $S$ leads towards the state template $S_1$ (Line 20) is done as follows. It must hold that $S^+ \supseteq \mathit{pre}(o)$, $S^+ \supseteq S_1^+ \setminus \mathit{eff}^+(o)$ and $S^- \supseteq S_1^- \setminus \mathit{eff}^-(o)$. Let $S'$ be a state template created according to (1)-(3) ($o$ is applied in $S$). If $S_1^+ \cup S'^- = \emptyset$, $S_1^- \cup S'^+ = \emptyset$ and $S_1^+ \cup S'^+ \neq \emptyset$, then the application of $o$ leads towards $S_1$. We have to find an operator $o' \neq o_2$ applicable in $S$ (i.e., $\mathit{pre}(o') \subseteq S^+$) such that it leads towards $S_0$ or $S_2$ (Line 21). Let $S''$ be a state template created according to (1)-(3) ($o'$ is applied in $S$). To be sure that the application of $o'$ really leads towards $S_0$ or $S_2$ it must hold that $S_0^+ \subseteq S''^+$ (analogously for $S_2$). If such an operator $o'$ does not exist, then condition 2 of Proposition 2 might be broken.

Because Proposition 2 requires reachability of $S_2$, then we must be sure that if $S_1$ is reachable then $S_2$ is reachable as well. If $\mathit{pre}(o_2) \subseteq \mathit{eff}^+(o_1)$ then if $o_1$ is applicable (leads towards $S_1$) then $o_2$ is applicable as well (leads towards $S_2$). If $S_1$ is not reachable then neither $o_1$ nor $o_2$ is applicable,

therefore removing them cannot break completeness.

It can be seen that the complexity of the Algorithm 1 depends quadratically on the number of planning operators. If $o_1$ is not about to be removed, then we can modify the algorithm by omitting Lines 8-19 (i.e., the goal check and applicability of another operator on $S_1$ check), because $S_1$ remains reachable from $S_0$. If $o_2$ is not about to be removed, then we can modify the algorithm by omitting Lines 5-7 and 20-25 (i.e., the initial state check and checking if any operator (except $o_1$) leads towards $S_1$), because $S_2$ remains reachable from $S_1$.

## Example

As an example how Algorithm 1 works we choose the well known BlockWorld domain. We have four operators *UNSTACK* (unstacking the (clear) block from another block and leave it hanging on the gripper), *STACK* (stack the block currently hanging on the gripper onto another (clear) block), *PICKUP* (picks the (clear) block from the table and leave it hanging on the gripper) and *PUTDOWN* (puts the block currently hanging on the gripper on the table). If we create a macro-operator *UNSTACK-PUTDOWN* and remove the primitive operators *UNSTACK* and *PUTDOWN*, then we can see that there still remain $PICKUP$ and *STACK*, inverse operators to the removed ones. Therefore, the test on Line 1 is successful. If we then create a macro-operator *PICKUP-STACK* and remove the primitive operators *PICKUP* and *STACK*, then the first test (Line 1) fails (no inverse operators exist). Passing the tests (Lines 5 and 8) is conditioned by the fact that no block must be hanging on the gripper in the initial or goal situation (we can see that there is only one atom in $S_1^+$ which refers to a situation in which a block is hanging on the gripper, this atom is not presented in $S_0^+$ or $S_2^+$). Similarly, we can see that there is no operator applicable in $S_1$ or leading towards $S_1$. Therefore, the completeness is not broken (if no block is required to be hanging on the gripper in the initial or goal state).

## Conclusion

In this paper we have formally defined the idea of *reformulation schemes* and their properties of soundness and completeness. Reformulation schemes are a general way of defining a reformulation on a PDM and setting up its "recovery" via a function that transforms the output plan into the original PDM formulation. We have defined action elimination and macro composition as two important types of *sound* reformulation scheme, and have shown that selecting particular action elimination choices using the relationship of entanglement is at least as hard as planning itself. Further, we used the theoretical framework to investigate a more complex reformulation scheme based on a combination of both macro-creation and action elimination. We introduced such a *complete* reformulation scheme via Proposition 2, and rigorously derived an operational form of the proposition in the form of an algorithm with low order polynomial time complexity.

Our future work will involve implementing and empirically evaluating Algorithm 1 using benchmark domains, and using any insights gained to discover further general reformulation schemes with guaranteed completeness, in order to improve the efficiency of AI planning engines. Another interesting topic for future work is inspired by the example (applying the Algorithm 1 in BlockWorld domain). We found out that we can remove primitive operators if and only if no block is hanging on the gripper in the initial or goal position. It requires some sort of reasoning which can tell us in which cases and why removing primitive operators causes loss of completeness. Such an information might be helpful for development more sophisticated reformulating schemes (for instance combination of macro-operators and entanglements).

## References


Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

Chrpa, L., and Barták, R. 2009. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA 2009*, 50–57.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review* 25(3):281–297.

Chrpa, L. 2011. Theoretical aspects of using learning techniques for problem reformulation in classical planning. In *Proceedings of PlanSIG*, 23–30.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI 1977*, 465–471.

Ghallab, M.; Nationale, E.; Aeronautiques, C.; Isi, C. K.; Penberthy, S.; Smith, D. E.; Sun, Y.; and Weld, D. 1998. Pddl - the planning domain definition language. Technical report.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

Hoffmann, J. 2011. Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h+. *The Journal of Artificial Intelligence Research (JAIR)* 41:155–229.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS 2007*, 256–263.

Richter, S., and Westphal, M. 2008. The lama planner using landmark counting in heuristic search. In *Proceedings of the sixth IPC*.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.