Can I Really Do That? Verification of Meta-Operators via Stackelberg Planning

Florian Pham[1] , **Álvaro Torralba**[2]
[1]Saarland University, Saarbrücken, Germany
[2]Aalborg University, Aalborg, Denmark
florianpham@yahoo.de, alto@cs.aau.dk

Abstract

Macro-operators are a common reformulation method in planning that adds high-level operators corresponding to a fixed sequence of primitive operators. We introduce meta-operators, which allow using different sequences of actions in each state. We show how to automatically verify whether a meta-operator is valid, i.e., the represented behavior is always doable. This can be checked at once for all instantiations of the meta-operator and all reachable states via a compilation into Stackelberg planning, a form of adversarial planning. Our results show that meta-operators learned for multiple domains can often express useful high-level behaviors very compactly, improving planners' performance.

1 Introduction

Classical planning deals with the problem of finding a sequence of actions to achieve a goal [Bylander, 1994]. Since the origins of the field, it was noted that reasoning at different levels of abstraction is key for efficient planning on complex domains. One way of introducing such new levels of abstraction is to introduce meta-actions that model effects that can be achieved by some sequence of primitive actions. This idea has been widely explored by using macro-actions [Fikes *et al.*, 1972; Korf, 1985; Botea *et al.*, 2005], meta-actions built from a fixed sequence of actions. Macro-operators go one step further, by considering sequences of parameterized operators, which are transferable skills to other instances of the same domain. Plenty of research has considered how to generate useful macro-operators, e.g., by learning them from existing plans [Chrpa, 2010; Chrpa *et al.*, 2014; Chrpa and Vallati, 2022], testing their benefits in combination with planning techniques [Coles and Smith, 2007; Gerevini *et al.*, 2009], by restricting the amount of considered instantiations and/or replacing primitive operators [Chrpa *et al.*, 2019]. They have shown that, despite increasing the amount of actions in the reformulated task, this can be beneficial due to introducing shortcuts in the state space and reducing the length of the solution, before replacing the macro-actions by their corresponding sequences or actions.

We introduce the concept of meta-operator, as an operator that is not part of the domain but whose effects can always be achieved on any state satisfying the preconditions by some, not necessarily fixed, sequence of actions. This is a lot more general than macro-operators, as completely different plans are allowed when applying the meta-operator on different states. This generality offers wide opportunities for defining useful behaviours in all kind of domains.

The main challenge is how to determine if a given meta-operator is valid, i.e., for any instantiation of the meta-operator and any reachable state satisfying the precondition, can it always be replaced by some sequence of primitive actions? Macro-operators are valid by construction, as they are derived from a fixed sequence of operators whose preconditions and effects are uniquely defined. However, we turn the problem around, defining meta-operator candidates as useful high-level behaviours and then verifying if they are indeed valid. This seems unfeasible, as it requires to find a plan for exponentially many states in the size of the planning task and exponentially many actions in the number of operator parameters. However, we show that it can be solved by compiling the problem into a single Stackelberg planning task, and leveraging Stackelberg planners [Speicher *et al.*, 2018]. Verifying meta-operators has benefits beyond their use to reformulate planning domains. We show that it generalizes the problem of testing whether actions are undoable [Daum *et al.*, 2016], i.e., whether their effects can be reversed.

After solving the reformulated task, meta-actions are replaced by a corresponding sequence of primitive actions. We propose a simple approach, based on re-planning for the missing parts of the plan. Finally, we extend techniques for deriving macro-operators in order to learn candidate meta-operators. We show that our compilation is effective at checking which candidates are valid, and that using meta-operators can pay off even when considering the time it takes to reconstruct the solution.

2 Background

A STRIPS planning task [Fikes and Nilsson, 1971; Bylander, 1994] is a tuple $\Pi = (F, A, I, G)$, where F is a finite set of facts, A is a finite set of actions, $I \subseteq F$ is the initial state and $G \subseteq F$ is the goal. Each action $a \in A$ has a precondition $\mathsf{pre}(a) \subseteq F$, and effects $\mathsf{add}(a), \mathsf{del}(a) \subseteq F$. We assume $\mathsf{add}(a) \cap \mathsf{del}(a) = \emptyset$.

```
turn_to (?s ?dn ?dp)
:pre (pointing ?s ?dp)
:add (pointing ?s ?dn)
:del (pointing ?s ?dp)

switch_on (?i ?s)
:pre (on_board ?i ?s), (power_avail ?s)
:add (power_on ?i)
:del (calibrated ?i), (power_avail ?s)

calibrate (?s ?i ?d)
:pre (on_board ?i ?s),(pointing ?s ?d),
 (calibration_target ?i ?d),(power_on ?i)
:add (calibrated ?i))
```

Figure 1: Example operators from the Satellite domain. Parameter types are indicated by the parameter name: satellite (?s), instrument (?i), and directions (?d, ?dn, ?dp).

A STRIPS planning task induces a state space over the set of states $S = 2^F$. Each state $s \in S$ is a subset of facts, which are currently true. An action a is *applicable* on a state s if $\mathsf{pre}(a) \subseteq s$. In that case, the result of applying a to s is $s[\![a]\!] = (s \cup \mathsf{add}(a)) \setminus \mathsf{del}(a)$. By $S[a]$ we denote the set of states to which a is applicable. The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are the states that satisfy the goal.

An action sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle$ can be applied on a state s if there exists a sequence of states $s_0, \ldots, s_n$, where $s_0 = s$, and for $i \in [1, n]$, a_i is applicable on s_{i-1} and $s_i = s_{i-1}[\![a_i]\!]$. The result is the state $s[\![\vec{a}]\!] = s_n$. We say that a state s is *reachable* if there exists a sequence of actions $\vec{a}$ such that $I[\![\vec{a}]\!] = s$. By $R(\Pi)$ we denote the set of states reachable from I. A solution for I is called a *plan* for Π and is a sequence of actions from the initial state to a goal state. Additionally, we consider a cost function $c : A \mapsto \mathbb{N}_0$. The cost of a plan is the sum of all its actions' cost. We use $c(\vec{a})$ to denote the cost of the sequence of actions, and $c^*(s, s')$ to denote the cost of the cheapest path from s to s'.

Tasks are often specified in lifted form, e.g., in the PDDL language [McDermott, 2000]. A lifted task Π^l is a tuple $(\mathcal{P}, O, C, I, G)$ where $\mathcal{P}$ is a set of (first-order) atomic *predicates*, O is a set of *operators*[1], C is a set of *object constants*, I is the *initial state*, and G is the *goal*. Predicates and action schemas have parameters. Both parameters and objects have types so that a parameter can be substituted by an object in C if their type matches. We denote individual parameters with x, y, z and sets of parameters with X, Y, Z. An operator $o[X]$ is a triple $(\mathsf{pre}(o), \mathsf{add}(o), \mathsf{del}(o))$, consisting of *preconditions*, *add list*, and *delete list*, all of which are a set of $\mathcal{P}$ instantiated with objects in C or parameters in X. I and G are subsets of $\mathcal{P}$, instantiated with objects from C.

A lifted task Π^l can be transformed into a STRIPS task $ground(\Pi^l)$, by instantiating the set of predicates and operators with the set of objects C to obtain the set of facts and actions, respectively [Helmert, 2009]. As an example, consider the Satellite domain [Long and Fox, 2003], where a set of observation tasks must be performed by several satellites, which are equipped with different instruments. Fig. 1 shows three of the action schemas, which are used to prepare instruments so that they can take images. The set of actions of an operator o, $A_{o[C]}$, contains an action for each valid substitution of the parameters. For example, the instantiations of `turn_to` could include `turn_to(satellite1, sun, earth)`, `turn_to(satellite1, earth, moon)`, etc.

[1]Operators are also often called *action schemas*. We follow the convention used in other macro-operator approaches of using "actions" to refer to the grounded level, and operators to the lifted level.

```
turn_to-switch_on-calibrate (?s ?i ?dn ?dp)
:pre (pointing ?s ?dp), (power_avail ?s),
     (calibration_target ?i ?dn),
     (on_board ?i ?s)
:add (calibrated ?i), (pointing ?s ?dn),
     (power_on ?i)
:del (pointing ?s ?dp), (power_avail ?s)
```

Figure 2: Macro-operator combining 3 Satellite operators.

2.1 Macro-operators

Given two actions $a_1, a_2 \in A$, the *macro-action*, $M(a_1, a_2)$ represents the application of a_1 followed from a_2 as follows:

- $\mathsf{pre}(M(a_1, a_2)) = \mathsf{pre}(a_1) \cup (\mathsf{pre}(a_2) \setminus \mathsf{add}(a_1))$,
- $\mathsf{add}(M(a_1, a_2)) = (\mathsf{add}(a_1) \setminus \mathsf{del}(a_2)) \cup \mathsf{add}(a_2)$, and
- $\mathsf{del}(M(a_1, a_2)) = (\mathsf{del}(a_1) \setminus \mathsf{add}(a_2)) \cup \mathsf{del}(a_2)$.

Such a macro-action is *valid* if $\mathsf{del}(a_1) \cap \mathsf{pre}(a_2) = \emptyset$. In that case, in any state s satisfying $\mathsf{pre}(M(a_1, a_2))$, we can apply a_1 followed by a_2 to reach $s[\![M(a_1, a_2)]\!]$. Similarly, we can define macro-actions from any sequence of action $\vec{a} = \langle a_1, \ldots, a_n \rangle$ as $M(\vec{a}) = M(M(M(a_1, a_2), a_3), \ldots, a_n)$. Checking whether such macro-action is valid can be done by checking every pair of actions within the recursion.

Macro-operators follow the same idea at a lifted level. We can consider any sequence of operators, along with a unification of their parameters to define a new macro-operator following the same rules as described for macro-actions. In satellite, in order to use an instrument one needs to turn the satellite towards a calibration target, switch on, and calibrate the instrument. Therefore, it may be useful to bundle these operators together into a macro-operator, as shown in Fig. 2.

2.2 Stackelberg Planning

Stackelberg planning is a framework of adversarial planning introduced by Speicher *et al.* [2018], inspired by Stackelberg security games [Tambe, 2011]. A Stackelberg planning task $\Pi^{st} = (F, A^L, A^F, I, G)$ is an extension of a STRIPS task, where there are two agents with separate sets of actions, the *leader* (A^L) and the *follower* (A^F). The follower's objective is to reach a state that satisfies the goal G, as in STRIPS. But the leader acts first with the objective of maximizing the follower's plan cost. For each state in $s \in 2^F$, we define its leader cost $L^*(s)$ as the cost of a cheapest sequence of actions in A^L starting from I and ending in s (or ∞ if no such sequence exists). The follower cost $F^*(s)$ is the optimal plan for the follower starting from s, i.e., the solution of a STRIPS task (F, A^F, s, G). A state s dominates another s' if $L^*(s) \leq$

$L^*(s')$, $F^*(s) \geq F^*(s')$ and one of the inequalities is strict. The solution of Π^{st} is the set of all leader-reachable non-dominated states $S^* \subseteq 2^F$. For our use case, it is not needed to compute multiple entries with the same leader and follower cost. Therefore, we focus on approaches that compute the Pareto front $PF(\Pi^{st})\{\langle L^*(s), F^*(s)\rangle \mid s \in S^*\}$ containing all non-dominated pairs of leader and follower cost [Torralba *et al.*, 2021]. Intuitively, a pair $\langle c^L, c^F\rangle \in PF(\Pi^{st})$ means that the leader can make the cost of the follower's optimal plan as high as c^F by spending as little as a cost of c^L.

One can define lifted Stackelberg planning tasks in an analog manner as for STRIPS tasks [Sauer *et al.*, 2023].

3 Meta-Operators

We start by defining meta-actions, as behaviours that can be achieved by arbitrary sequences of actions.

Definition 1 (Meta-action)**.** *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task. An action $a^{\mathcal{M}}$ is a valid meta-action for Π if and only if for all reachable states $s \in R(\Pi)$ where $\mathsf{pre}(a^{\mathcal{M}}) \subseteq s$, there exists an action sequence $\vec{a} = \langle a_1, \ldots, a_n\rangle \in A$ such that $s[\![\vec{a}]\!] = s[\![a^{\mathcal{M}}]\!]$.*

We remark that this property can refer to any action, and in fact this property is interesting only when $a^{\mathcal{M}} \notin A$. The notion can be naturally extended to the lifted case.

Definition 2 (Meta-operator)**.** *An operator $o^{\mathcal{M}}$ is a valid meta-operator for a lifted planning task Π^l if and only if all instantiations $a^{\mathcal{M}} \in A_{o^{\mathcal{M}}[C]}$ are valid meta-actions for $ground(\Pi^l)$.*

Figure 3 shows an example of a meta-operator. Meta-switch-on-calibrate achieves the same useful effects as the macro-operator in Figure 2, without the side-effect changing the direction the satellite is pointing to. This is beneficial for having more focused meta-actions, that can achieve the desired effect with fewer preconditions and/or side effects.

Meta-operators generalize the standard notion of macro-operator. The key difference is that in the definition of macro-action, the sequence $\overrightarrow{a}$ is the same for all states. Similarly, macro-operators require that the sequence $\overrightarrow{a}$ can be lifted, i.e., for all instantiations of o the sequence $\overrightarrow{o}$ uses the same action schemas and parameter mappings.

Considering the special case of macro-operators has several advantages. First of all, it makes trivial the process of identifying if the meta-operator is valid, as it suffices to check if the precondition of one of the actions in the sequence is contradicted by the effect of a previous action. Second, it is always straightforward to translate a plan that depends on macro-actions to one that only uses primitive operator: as we know in advance for each macro-action exactly what sequence of actions should it be replaced by.

Meta-actions, on the other hand, are hard to validate. Even when considering a meta-action that is applicable only on a single state, if the effect of the meta-action consists of achieving the goal, then validating whether it is valid is as hard as deciding solvability of planning, i.e., PSPACE-hard.

But, the generality of meta-operators has strong advantages too, as it opens widely the space of possible reformulations

```
meta-switch-on-calibrate (?s ?i)
 :pre (on_board ?i ?s) (power_avail ?s)
 :add (calibrated ?i) (power_on ?i)
 :del (power_avail ?s)
```

Figure 3: Meta-operator to switch on and calibrate an instrument

that can be applied to the planning task. For example, macro-operators have a hard time modelling scenarios where an object needs to traverse multiple locations to reach a final destination. In those cases, to ensure that there is a fixed sequence of actions for all possible instantiations, it is necessary to introduce parameters that model all intermediate locations visited along the way. This is not only cumbersome, but utterly impractical, as the number of instantiations grows exponentially. With meta-operators, we can instead "assume" that the target will be reachable by some sequence of actions. Ensuring validity of the meta-operator will necessarily involve checking this for all pairs of source and destination. Similarly, in our example, we can get rid of certain parameters such as the direction the satellite is pointing to, making the meta-operator more applicable with less instantiations.

4 Validation via Stackelberg Planning

We compile the problem of testing whether a given action schema is a valid meta-action for a given task as a Stackelberg planning task. The idea of the compilation is that the leader can apply any sequence of actions to choose any reachable state $s \in R(\Pi)$ where some instantiation of the operator is applicable, and choose one such action. Then, the goal of the follower will be to go from the resulting state, to a state where the effects of the action have been accomplished.

Definition 3. *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task, $A^{\mathcal{M}}$ a set of actions. The meta-action task of Π and $A^{\mathcal{M}}$ is the Stackelberg planning task $\Pi^{\mathcal{M}}(\Pi, A^{\mathcal{M}}) = (F^{st}, A^L, A^F, I^{st}, G^{st})$ where:*

- $F^{st} = F \cup \{g^p, ok^p \mid p \in F\} \cup \{turn^L\}$
- $I^{st} = I \cup \{turn^L\} \cup \{ok^p \mid p \in F\} \cup \{g^p \mid p \in I\}$
- $G^{st} = \{ok^p \mid p \in F\}$
- $A^L = \{\alpha^L \mid \alpha \in A\} \cup \{\mathtt{passTurn}^L[a^{\mathcal{M}}] \mid a^{\mathcal{M}} \in A^{\mathcal{M}}\}$
- $A^F = \{\alpha^F \mid \alpha \in A\} \cup \{\mathtt{achieveGoal}^F\}$

where the actions are defined as shown in Table 1.

For each fact $p \in F$, our compilation includes three different copies: p represents the current value; g^p represents whether the follower's goal is to make p true or false; and ok^p represents whether the current value of p is the desired one. In other words, the value of ok^p can be derived from p and g^p, as there is always an invariant such that $ok^p = (p \leftrightarrow g^p)$. Finally, there is an additional fact $turn^L$ indicating when the leader passes the turn to the follower by choosing which action $a^{\mathcal{M}}$ should the follower verify from the current state.

The leader has two types of actions, α^L, and $\mathtt{passTurn}^L[a^{\mathcal{M}}]$. α^L is a copy of the original actions, and allows the leader to navigate through the state space to

Action	Precondition	Effect	Cost
α^L	$\mathsf{pre}(\alpha) \cup \{turn^L\}$	$\{p, g^p \mid p \in \mathsf{add}(\alpha))\} \cup \{\neg p, \neg g^p \in \mathsf{del}(\alpha))\}$	0
$\texttt{passTurn}^L[a^{\mathcal{M}}]$	$\mathsf{pre}(a^{\mathcal{M}}) \cup \{turn^L\}$	$\{\neg turn^L\} \cup \{g^p, (\neg p \rightarrow \neg ok^p) \mid p \in \mathsf{add}(a^{\mathcal{M}}))\} \cup$ $\{\neg g^p, (p \rightarrow \neg ok^p) \mid p \in \mathsf{del}(a^{\mathcal{M}}))\}$	0
α^F	$\mathsf{pre}(\alpha)$	$\{\ p, (g^p \rightarrow ok^p), (\neg g^p \rightarrow \neg ok^p) \mid p \in \mathsf{add}(\alpha)\} \cup$ $\{\neg p, (g^p \rightarrow \neg ok^p), (\neg g^p \rightarrow ok^p) \mid p \in \mathsf{del}(\alpha)\}$	$c(\alpha)$
$\texttt{achieveGoal}^F$	$\{turn^L\}$	G^{st}	0

Table 1: Action definition for all actions in the Stackelberg planning task to verify if d is a valid meta-action. In the effect definition, positive literals means that they are add effects, whereas negative literals represent delete effects. As syntactic sugar, we use conditional effects of the form $(cond \rightarrow eff)$, where the effect eff is applied if and only if the $cond$ holds in the state where the action is applied.

reach any reachable state. Through this process, we also update g^p so that $g^p = p$. Then, the $\texttt{passTurn}^L$ actions allow the leader to pick one action in $A^{\mathcal{M}}$, if the preconditions are satisfied. Note that, due to the fact $turn^L$, only one such action may be applied, and it must be the last action applied by the leader. The action $\texttt{passTurn}^L[a^{\mathcal{M}}]$ sets the goal for the follower, by applying the effects of $a^{\mathcal{M}}$ on the g^p facts. Also, it keeps the $ok^p = (p \leftrightarrow g^p)$ invariant. Note that, after applying $a^{\mathcal{M}}$, the value of g^p facts corresponds to the state that would be obtained by applying $a^{\mathcal{M}}$ on the current state (so that all facts in $\mathsf{add}(a^{\mathcal{M}})$ are true, all facts in $\mathsf{del}(a^{\mathcal{M}})$ are false, and all remaining facts should retain the same value).

Then, the follower will apply actions to reach the state that results from applying $a^{\mathcal{M}}$ in the current state. This is so, because it must achieve all ok^p facts, whose value satisfies the invariant $ok^p = (p \leftrightarrow g^p)$. As actions α^F do not have any effect over g^p, the only way is to change the values of the current facts p until they have the same value as g^p. Finally, the action $\texttt{achieveGoal}^F$ simply forces the leader to end their plan with a $\texttt{passTurn}^L$ action, as otherwise it is trivial for the follower agent to reach the goal with a cost of 0.

The formulation of Definition 3 uses some syntactic sugar to define the effects of the α^F actions, for the sake of simplifying the presentation. Namely, it uses conditional effects, some of which have negated literals ($\neg g^p$) in the conditions. This can be compiled away into the standard formalism of Stackelberg planning introduced in Section 2.2 by:

- Eliminating negative conditions, e.g., using an additional fact per $p \in F$ that always has the opposite value than g^p. In practice, this is not needed as the Stackelberg planner we use supports negative preconditions.
- Compiling conditional effects away [Nebel, 2000]. In our implementation, we create multiple copies of each action, one per combination of conditional effects [Gazen and Knoblock, 1997]. This is feasible because there are not too many conditional effects, and we can exclude any combination with mutually exclusive conditions.

Action costs are not strictly needed to test validity. However, using costs is convenient because the leader just tests reachability and the distance from the initial state is not relevant. Moreover, as Thm 1 shows, this allows us to obtain what is the maximum cost required for applying the meta-operator.

Theorem 1. *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task and $A^{\mathcal{M}}$ a set of actions. Then, the solution of $\Pi^{\mathcal{M}}(\Pi, A^{\mathcal{M}})$, $PF(\Pi^{\mathcal{M}}(\Pi, A^{\mathcal{M}}))$, contains a single entry $(0, c^{\mathcal{M}})$, and*

$$c^{\mathcal{M}} = \max_{a^{\mathcal{M}} \in A^{\mathcal{M}}} \max_{s \in R(\Pi)} c^*(s, s[\![a^{\mathcal{M}}]\!]) \text{ if } s \in S[a^{\mathcal{M}}] \text{ else } 0$$

Proof. $PF(\Pi^{\mathcal{M}}(\Pi, A^{\mathcal{M}}))$ contains a single entry because all leader actions have a cost of 0. By definition of Stackelberg planning, the follower cost of such entry is $\max_{\pi^L} \min_{\pi^F} c(\pi^F)$, where π^L and π^F range over all possible leader and follower plans, respectively. We need to show that this is equal to $c^{\mathcal{M}}$.

The leader plan π^L maximizing the expression necessarily corresponds to a path $\langle \alpha^L_1, \ldots, \alpha^L_m, \texttt{passTurn}^L[a^{\mathcal{M}}] \rangle$. The only exception is if no such path is applicable. In that case, the meta-operator is trivially valid due to its preconditions not being met in any reachable state, and $c^{\mathcal{M}} = 0$. Otherwise, as the leader attempts to maximize $c^{\mathcal{M}}$, $\texttt{passTurn}^L[a^{\mathcal{M}}]$ should be applied (otherwise the follower can reach the goal with cost 0 using $\texttt{achieveGoal}^F$). And due to $turn^L$, it is necessarily the last action in π^L. As there is a direct correspondence between each $\pi^L = \langle \alpha^L_1, \ldots, \alpha^L_m, \texttt{passTurn}^L[a^{\mathcal{M}}] \rangle$, and the set of reachable states $R(\Pi)$ where $a^{\mathcal{M}}$ is applicable, it suffices to show that given such a path $c^*(s, s[\![a^{\mathcal{M}}]\!]) = \min_{\pi^F} c(a^F)$, where $s = I[\pi^L]$.

Let π^F be the optimal follower plan. Again, there is one such plan for every sequence of actions applicable from s, as the preconditions and effects of each α^F over facts p are exactly the same as those of the original actions α. It remains to show that such a path satisfies the follower's goal G^{st} iff $s[\pi^F] = s[a^{\mathcal{M}}]$. G^{st} requires to have all ok^p facts, which as explained above keep the invariant $ok^p = (p \leftrightarrow g^p)$. As $\texttt{passTurn}^L[a^{\mathcal{M}}]$ sets the g^p facts to exactly the set of facts true in $s[a^{\mathcal{M}}]$ this is equivalent to reach a state in which $p \leftrightarrow g^p$. □

However, for testing whether a meta-operator is valid, it is sufficient to run the compilation with unit costs. Indeed, a meta-operator is valid as long as the Pareto front does not contain an entry where the follower cost is ∞.

Corollary 1. *Let $\Pi = (F, A, I, G)$ be a STRIPS planning task and $a^{\mathcal{M}}$ an action. Then, $a^{\mathcal{M}}$ is a meta-action if and only if $PF(\Pi^{\mathcal{M}}(\Pi, A^{\mathcal{M}}))$ does not contain an entry (x, ∞).*

To test if a candidate operator o is a valid meta-operator on a task Π, we can simply set $A^{\mathcal{M}}$ to be all valid instantiations of o on Π. Note that then the actions $\texttt{passTurn}^L[a^{\mathcal{M}}]$ can also be written as an operator (action schema) in PDDL.

5 Relation to Undoability

The problem of testing whether a meta-operator is valid is related to the notion of *undoability* [Eiter *et al.*, 2007; Eiter *et al.*, 2008; Daum *et al.*, 2016; Morak *et al.*, 2020], i.e., testing whether the effects of a given action/operator can always be undone. This is the case, if for every reachable state s where a is applicable, $c^*(s[\![a]\!], s) < \infty$. Daum *et al.* [2016] showed that the problem of testing whether an action is undoable can be compiled into a contingent planning task. This can also be done with our compilation. For any planning action, we can always define a reverse action, which has the opposite effects.[2] Then, testing whether the action is undoable, is equivalent to testing whether the reverse action is a valid meta-action.

Our Stackelberg compilation is inspired by their compilation to Contingent planning, but also has several advantages. We do not need to approximate the set of reachable states by using state invariants, as our encoding automatically lets the leader enumerate exactly the set of reachable states. Furthermore, in order to test whether an operator is undoable, the contingent planning compilation requires to solve a separate contingent planning task for each action instantiation. Our compilation instead is able to test validity of a meta-operator by solving a single Stackelberg planning task.

6 Learning and Using Meta-Operators

Following previous work on macro-operators, our approach is based on learning a set of meta-operators on a set of training instances, for which we already have found a plan. Then, to solve a new planning task, we reformulate the domain PDDL file by introducing our meta-operators. Finally, after finding a plan on the reformulated task we replace any meta-action in the plan by a sequence of actions achieving the same effects.

6.1 Generate Meta-Operator Candidates

We base our approach to generate candidates on previous macro-operator approaches. Specifically, we start from a set of selected macro-operators and generate more compact meta-operators that can get similar effects, but reducing the amount of parameters and preconditions. This increases the applicability of the meta-actions and reduces the amount of instantiations that are needed. We generate candidates by applying three types of modifications to the macro-operators.

Remove precondition parameters (C_{pre}) Our first method, removes parameters that are absent in the effect of the macro. This aims at obtaining meta-operators that are improved versions of the original macro-operators, with the same useful effects of macros while reducing the number of parameters. When a parameter is removed, all predicates containing this parameter are removed from the action precondition as well. If this causes a parameter to not be mentioned anymore in the precondition and/or effect of the macro, we remove it as well. Given a macro-operator, we attempt to remove the parameters one by one. Note that removing preconditions of an invalid candidate can never cause it to become a valid meta-operator, as this only increases the amount of instantiations and reachable states where they are applicable. Therefore, we only consider removing additional parameters over successful candidates.

Remove effect parameters (C_{eff}) A limitation of the C_{pre} method, is that in many cases there is no parameter that is not mentioned at all in the effects. In our running example, `turn_to-switch_on-calibrate`, all parameters are mentioned in the effect, so C_{pre} is not applicable. However, often only some effects are useful, whereas others are side effects. In our running example, the effects modifying the direction the satellite is pointing to are not essential.

Our second method, C_{eff}, eliminates a parameter appearing in the effects, removing any precondition and/or effect that depends on it. This may cause other parameters to be removed as well. In contrast to C_{pre}, we could in principle remove additional parameters even if intermediate results were not valid candidates. However, this leads to too many candidates so we stick with removing a single parameter.

Remove additional effects (C_{inv}) Removing effect parameters might cause the resulting candidates to be automatically invalid, whenever they do not comply with the state-invariants. For example, removing `?d_new` from `turn_to-switch_on-calibrate` results in a meta-operator that deletes `(pointing ?s ?d_prev)` without adding a new direction. This meta-operator is invalid, as it contradicts the state-invariant that a satellite always points in a direction. In principle, it is possible to automatically extract lifted mutex groups [Helmert, 2009; Fišer, 2020]. Here, we just approximate this by observing that many lifted invariants refer to the same predicate. Therefore, we take the C_{eff} candidates, and remove other preconditions and effects with the same predicate as the ones removed by the C_{eff} method. In our running example, we remove preconditions and effects with `calibration_target` and `pointing`. The result is the meta-operator from Fig. 3, which does not violate any invariant and is in fact valid.

6.2 Solving New Instances

Once we have collected a set of meta-operator candidates, we use our compilation from Section 4 on the set of training instances. We then extend the domain with a set of meta-operators that were valid on all training instances, so that any classical planner can be called on the reformulated domain. When a new instance arrives, we do not validate the meta-operators again, as that would be prohibitive. Instead, we rely on the training instances being representative of the entire domain, which is a common assumption in learning and/or generalized planning approaches. Note that the behaviour described by valid meta-operators was doable for all instantiations of the operator on all states of the training task, so a few training instances suffice to obtain a strong evidence that this behaviour is always doable in the domain.

After solving the instance on the reformulated domain, we obtain a plan which may contain one or more meta-actions. The next step is to reconstruct a plan composed only of primitive actions. This was easy for macro-actions as they correspond to fixed action sequences. For meta-actions, how-

[2] This may require considering multiple actions, e.g. if some added facts could already be true.

ever, one needs to find a specific plan for the state in which they are applied. In principle, the validation process may give hints about the structure of such plans. However, transferring this from the training instances to the new instance is an open problem. Instead, we follow a very simple approach: calling a classical planner for each meta-action on the original domain to find a plan from the current state to the exact state that results from applying the meta-action.

At first glance, this solution reconstruction procedure may look completely unfeasible: solving multiple classical planning problems which could be as hard as the original one. Indeed, in some domains the overhead is noticeable. However, there are several reasons why this can still pay off. First, the goal is fully specified, which tends to make heuristics more informed. Second, if a meta-operator could be validated, then it was possible to find plans that work for all states in the state-space of the training instance. Thus, one can expect these plans to not be too long, nor particularly difficult to find.

7 Experiments

We evaluate meta-operators on a standard set of IPC benchmarks. All experiments were conducted on a cluster of Intel Xeon CPU E5-2660 with 2.20GHz using Downward Lab [Seipp *et al.*, 2017]. Each run had a time limit of 1800 seconds and a memory limit of 4 GB. The code and data are publicly available [Pham and Torralba, 2023].

7.1 Learning and Verifying Meta-Operators

First, we analyze the set of meta-operators that can be learned by our approach. For this, we use a small set of 5 instances per domain generated with the publicly available random generators of these IPC domains. As a seed to our method we use macro-operators generated with the Planning Task Transformer (PTT) tool, which implements multiple generation methods: exploiting action dependencies in plans [Chrpa, 2010], MUM [Chrpa *et al.*, 2014], and using inner entanglements [Chrpa *et al.*, 2019]. We learn macros with all three configurations and take the union (which we denote PTT) as a basis for generating meta-operator candidates. Henceforth, we focus on domains for which the PTT set is not empty.

As a sanity check, we ran our validation tool on the set of PTT macro-operators. Interestingly, despite being valid by construction, our tool reported a few cases invalid. As the tool provides the leader plan, this can be used to analyze in what kind of states and instantiations lies the issue. In this case, two parameters were being instantiated by the same object, so that the resulting action had the same fact on the add and delete effects. We fix this by introducing an additional precondition requiring both parameters to be different. This shows that our validation tool can have uses beyond learning meta-actions, like finding bugs during domain modelling.

We validate the candidates by using our meta-operator compilation into a single Stackelberg planning tasks. As Stackelberg planner, we use the symbolic leader search algorithm [Torralba *et al.*, 2021]. We use two configurations that differ on whether the plans computed for the follower are optimal or not. The `SLS-opt` configuration, used by Torralba *et al.* [2021], is an optimal symbolic search configuration [Torralba *et al.*, 2017]. The `SLS-sat` configuration,

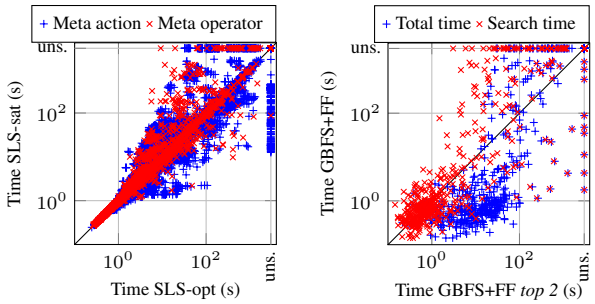

Figure 4: Left: Runtime of optimal and satisficing Stackelberg planners for validating meta-actions/operators. Right: Total and search (excluding plan reconstruction) time with/without meta-operators.

on the other hand, uses Greedy Best-First Search with the FF heuristic [Hoffmann and Nebel, 2001]. In the `SLS-sat` configuration, we also modify the symbolic leader search algorithm by initializing the initial follower cost to a large constant (10^5). This indicates that the exact follower cost is irrelevant, as we are mainly interested in whether the meta-operator is valid (has a cost lower than ∞) or not. We consider both the problem of validating the entire meta-operator, as well as validating a single meta-action (i.e., a specific instantiation of the operator). For the latter, we sampled 10 instantiations for each instance and meta-operator candidate.

Fig. 4 (left) compares both configurations in terms of runtime. Overall, `SLS-sat` solves more cases, which is not surprising given that it is a satisficing configuration. But the plot shows that in many cases, we can find the maximum cost required to emulate such behaviour on a specific planning task. Table 2 gives an overview of the number of candidates created by our three methods and how many of them were validated. A meta-operator is validated if either `SLS-opt` and/or `SLS-sat` is able to prove that is valid on all 5 training instances. We also considered validated two candidates in the transport domain, where the meta-operator compilation ran out of memory but all sampled instantiations were validated by the meta-action compilation. For most of the rejected candidates, they were shown to not be valid in at least one of the instances. The set of learned meta-operators contains all validated candidates except those that were subsumed either by another candidate, an operator of the domain, or a macro-operator in the PTT set. Here, we consider a meta-operator to be subsumed if it has the same effect and the same or more preconditions, up to renaming of the parameters.

In most domains, we learned meta-operators that describe the effects of the macro-operators in a more succinct way. Our three criteria are very complementary, finding interesting meta-operators in different domains. While many meta-operators have a low cost, meaning that they can be replaced by short sequences of primitive operators, there are also a significant number of cases where the behaviour described by the meta-operator is non-trivial, i.e., at least in some cases it requires more than a handful of primitive actions.

7.2 Planning with Meta-Operators

Our next experiment evaluates whether meta-operators can enhance satisficing planners. We compare the performance

Domain	PTT	# candidates C_{pre}	C_{eff}	C_{inv}	$\sum$	# validated C_{pre}	C_{eff}	C_{inv}	$\sum$	$c^{\mathcal{M}}$
Barman	4	12	18	15	45	12	2	2	16	10
Blocks	4	0	7	4	11	0	0	0	0	–
Childsnack	4	2	19	2	23	1	1	2	4	6
Depots	4	2	20	14	36	2	4	0	6	6
Driverlog	3	1	13	9	23	1	1	0	2	10
Freecell	2	7	12	12	31	0	0	0	0	–
Grid	4	0	12	6	18	0	1	2	3	13
Gripper	1	0	4	2	6	0	1	0	1	7
Hiking	1	0	5	2	7	0	0	0	0	–
Logistics	6	3	24	12	39	0	2	0	2	5
Miconic	3	0	9	6	15	0	0	6	6	5
Mprime	2	0	16	16	32	0	0	0	0	–
Mystery	2	1	13	11	25	0	1	1	2	2
Nomystery	3	4	20	17	41	0	1	0	1	2
Parking	1	0	5	4	9	0	0	0	0	–
Rovers	3	8	14	5	27	8	1	3	12	7
Satellite	3	1	13	5	19	1	2	4	7	13
TPP	2	9	11	8	28	9	0	0	9	7
Transport	2	0	15	13	28	0	2	0	2	8
Zenotravel	1	0	6	5	11	0	3	1	4	8

Table 2: Number of macro-operators used as seed (PTT), candidate meta-operators, and number of validated macros. $c^{\mathcal{M}}$ shows the maximum plan length required to replace a meta-operator by primitive actions across all training instances and valid candidates.

Domain	#	GBFS + FF –	PTT	all	rel	tp2	LAMA –	PTT	all	rel	tp2
Barman11	20	14	3	**19**	**19**	**19**	**20**	**20**	**20**	**20**	**20**
Barman14	20	5	0	17	**19**	18	**20**	**20**	**20**	**20**	**20**
Childsnack	20	1	4	**8**	6	**8**	6	**18**	0	**18**	**18**
Depot	22	16	**19**	14	13	**19**	20	21	21	21	**22**
Driverlog	20	**18**	**18**	16	16	16	**20**	**20**	**20**	**20**	**20**
Grid	5	4	4	4	4	**5**	**5**	**5**	4	4	**5**
Logistics98	35	28	26	**34**	**34**	**34**	**35**	32	34	34	34
Mystery	30	**17**	16	16	16	16	**19**	18	16	18	18
Nomystery	20	9	4	**12**	**12**	**12**	11	6	**12**	**12**	**12**
Rovers	40	**26**	20	21	25	**26**	**40**	**40**	32	36	**40**
Satellite	36	27	**30**	27	27	28	**36**	31	27	27	**36**
TPP	30	23	21	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
Transport08	30	**17**	13	13	13	13	**30**	14	13	13	**30**
Transport11	20	**0**	**0**	**0**	**0**	**0**	**18**	0	0	13	13
Transport14	20	**0**	**0**	**0**	**0**	**0**	**16**	0	0	0	5
$\sum$	586	423	396	449	452	**462**	**544**	493	467	504	536

Table 3: Coverage of standard planners with the original domain (–), macro-operators learned by PTT, and three sets of meta-operators.

of two planners, implemented in the Fast Downward planning system [Helmert, 2006]: GBFS with the FF heuristic, which is a standard baseline for satisficing planning; and LAMA [Richter and Westphal, 2008], which is a state-of-the-art satisficing planner. We use the same configuration for finding the plan on the reformulated domain, as well as for reconstructing the plan with only primitive operators. We evaluate on the instances from the IPC, which were not used for learning or validating the meta-operator candidates.

We used three sets of meta-operators. The first set includes all meta-operators validated in the previous phase. Then, we identified the most useful meta-operators by solving a separate set of validation instances, distinct both from the training and the evaluation set. We solve those with multiple domain files: one with all meta-operators, as well as variants with exactly one meta-operator added. Our second set, *rel*, selects any meta-operator used by any of the plans found. This filters any irrelevant meta-operator that will not be used in plans anyway. Finally, to focus on useful meta-operators, we select only the *top 2* (tp2) according to the performance of configurations including exactly one meta-operator on the validation instances. We pick the two that lead to highest coverage, and among those, the ones with lowest total runtime, including the time to find a plan as well as the repair time.

Fig. 4 (right) shows the runtime on IPC instances of the *top 2* configuration. Meta-operators are not always helpful, specially on easy instances solved in less than 100s by the baseline. But despite the overhead, there are a number of instances that are solved up to two orders of magnitude faster with meta-operators. This is also reflected in the coverage results of Table 3. Meta-operators improve the coverage on the baseline and, even though they are outperformed when using LAMA, coverage still improves in specific domains like depots or nomystery. This shows that meta-operators have a huge potential to reduce search effort on hard planning tasks.

Regarding the overhead, this is partially due to adding too many additional actions, which explains why using *all* meta-operators may be detrimental. We alleviated this by using only the *top 2* meta-operators, but there is still margin of improvement, e.g. using known enhancements on macro-operators such as outer-entanglements [Chrpa *et al.*, 2018]. On the other hand, the overhead of plan reconstruction can be observed by comparing search and total time in Fig. 4. This is a bottleneck on easy instances, where meta-operators were not too helpful. But when meta-operators significantly reduce search effort on hard instances, it still pays off in total time.

8 Conclusions

Combining multiple primitive operations into macro-operators is a well-known method to reformulate planning problems and plan at a more abstract level. We introduce meta-operators, which define behaviours that can always be accomplished in any state satisfying their precondition, possibly with very different action sequences for different states. This allows for more compact high-level actions with fewer parameters and more focused on useful effects.

We show how to validate whether any invented, or user-provided, meta-operator is valid or not. This requires to check whether there exists any reachable state and instantiation of the meta-operator in which the preconditions are met but achieving the effects is not possible. This can be done efficiently by leveraging Stackelberg planners. This is a powerful tool, that opens many opportunities such as new ways of domain debugging, analyzing properties like undoability, or finding new reformulations for the planning task.

Our experimental results show the potential for using meta-operators as a reformulation technique. Using meta-operators can pay off on hard instances, even when considering the non-trivial plan reconstruction process.

References

[Botea *et al.*, 2005] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[Chrpa and Vallati, 2022] Lukás Chrpa and Mauro Vallati. Planning with critical section macros: Theory and practice. *Journal of Artificial Intelligence Research*, 74:691–732, 2022.

[Chrpa *et al.*, 2014] Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey. MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do, editors, *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014.

[Chrpa *et al.*, 2018] Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey. Outer entanglements: a general heuristic technique for improving the efficiency of planning algorithms. *J. Exp. Theor. Artif. Intell.*, 30(6):831–856, 2018.

[Chrpa *et al.*, 2019] Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey. Inner entanglements: Narrowing the search in classical planning by problem reformulation. *Computational Intelligence*, 35(2):395–429, 2019.

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

[Coles and Smith, 2007] Andrew Coles and Amanda Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156, 2007.

[Daum *et al.*, 2016] Jeanette Daum, Álvaro Torralba, Jörg Hoffmann, Patrik Haslum, and Ingo Weber. Practical undoability checking via contingent planning. In Amanda Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner, editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 106–114. AAAI Press, 2016.

[Eiter *et al.*, 2007] Thomas Eiter, Esra Erdem, and Wolfgang Faber. On reversing actions: Algorithms and complexity. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 336–341, 2007.

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

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] Richard Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(1-3):251–288, 1972.

[Fišer, 2020] Daniel Fišer. Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9835–9842. AAAI Press, 2020.

[Gazen and Knoblock, 1997] B. Cenk Gazen and Craig A. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning. 4th European Conference on Planning (ECP 1997)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, pages 221–233. Springer-Verlag, 1997.

[Gerevini *et al.*, 2009] Alfonso E. Gerevini, Alessandro Saetti, and Mauro Vallati. An automatically configurable portfolio-based planner with macro-actions: PbP. In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 350–353. AAAI Press, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

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

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

[Korf, 1985] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Long and Fox, 2003] Derek Long and Maria Fox. The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[McDermott, 2000] Drew McDermott. The 1998 AI Planning Systems competition. *AI Magazine*, 21(2):35–55, 2000.

[Morak *et al.*, 2020] Michael Morak, Lukas Chrpa, Wolfgang Faber, and Daniel Fišer. On the reversibility of actions in planning. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 652–661, 9 2020.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[Pham and Torralba, 2023] Florian Pham and Álvaro Torralba. Code, benchmarks, and experimental data for the

IJCAI 2023 paper "can i really do that? verification of meta-operators via stackelberg planning". https://doi.org/10.5281/zenodo.7942439, 2023.

[Richter and Westphal, 2008] Silvia Richter and Matthias Westphal. The LAMA planner — Using landmark counting in heuristic search. IPC 2008 short papers, http://ipc.informatik.uni-freiburg.de/Planners, 2008.

[Sauer *et al.*, 2023] Philipp Sauer, Marcel Steinmetz, Robert Künnemann, and Jörg Hoffmann. Lifted stackelberg planning. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling, ICAPS 2023*, 2023.

[Seipp *et al.*, 2017] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[Speicher *et al.*, 2018] Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. Stackelberg planning: Towards effective leader-follower state space search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 6286–6293. AAAI Press, 2018.

[Tambe, 2011] Milind Tambe. Security and game theory: Algorithms, deployed systems, lessons learned. *Cambridge University Press*, 2011.

[Torralba *et al.*, 2017] Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence*, 242:52–79, 2017.

[Torralba *et al.*, 2021] Álvaro Torralba, Patrick Speicher, Robert Künnemann, Marcel Steinmetz, and Jörg Hoffmann. Faster stackelberg planning via symbolic search and information sharing. In Kevin Leyton-Brown and Mausam, editors, *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 11998–12006. AAAI Press, 2021.