Operator Pruning using Lifted Mutex Groups via Compilation on Lifted Level

Daniel Fišer

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
danfis@danfis.cz

Abstract

A lifted mutex group is a schematic first-order description of sets of facts such that each set contains facts out of which at most one can hold in any reachable state. It was previously shown that lifted mutex groups can be used for pruning of operators during grounding of PDDL tasks, i.e., it is possible to prune unreachable and dead-end operators even before the grounded representation is known. Here, we show that applying such a pruning technique does not require a modification of the grounding procedure. Instead, it is possible to compile the conditions under which we can use lifted mutex groups to prune operators directly into the preconditions of lifted actions on the PDDL level. In fact, we show that such compilation captures the pruning power of lifted mutex groups perfectly.

1 Introduction

State invariants were studied in a variety of contexts (Fox and Long 1998; Gerevini and Schubert 1998; Rintanen 2000; Haslum and Geffner 2000; Bonet and Geffner 2001; Mukherji and Schubert 2005; Haslum 2009). One of the most prominent state invariants are so-called mutex groups, which are sets of facts out of which at most one can be part of any reachable state. They are necessary for a concise translation from STRIPS representation, where states are sets of facts, to finite domain representation, where states are assignments to multi-valued variables (Helmert 2009; Fišer et al. 2021).

A fact-alternating mutex group (fam-group) (Fišer and Komenda 2018) is a specific type of mutex group that behaves monotonically over state transitions. More specifically, given a reachable state s and a fam-group M, if $s \cap M = \emptyset$, then also $s' \cap M = \emptyset$ for every state s' reachable from s. This property can be used for removing actions that always lead to dead-end states (Fišer and Komenda 2018), and it was recently shown by Fišer (2020) that their lifted (first-order) descriptions (lifted fam-groups) can be used to prune such actions during grounding of PDDL (McDermott 2000) tasks to the propositional (STRIPS) level, i.e., even before the whole ground representation is known.

In this work, we follow the work of Fišer (2020) by showing that pruning using lifted fam-groups can be compiled directly into preconditions of lifted (PDDL) actions. We show that the conditions, under which a ground action can be recognized as unreachable or dead-end using a lifted fam-group, can be exactly expressed as a formula over equality predicates. The negation of such a formula can be directly integrated into the precondition of a (lifted) action so that the precondition cannot be satisfied in an unreachable state or a state leading to a dead-end (to the extent of the pruning power of lifted fam-groups). To this end, we propose a general method for turning unification between lifted atoms into a formula.

The practical implication of this contribution is that pruning during grounding does not require a modification of the grounding algorithm, but merely a compilation of the input PDDL task into another PDDL task. Moreover, we show that this compilation method can also be effortlessly used as online pruning during search in case of lifted planning.

2 PDDL and STRIPS Planning Tasks

We consider PDDL tasks without axioms, conditional effects or formulae with quantifiers.

Definition 1. A **PDDL task** is a tuple $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ where $\mathcal{B}$ is a non-empty set of **objects**, $\mathcal{T}$ is a non-empty set of **types** containing a default type $t_0 \in \mathcal{T}$. Objects and types are associated by a total function $\mathcal{D} : \mathcal{T} \mapsto 2^{\mathcal{B}}$ such that $\mathcal{D}(t_0) = \mathcal{B}$ and for every pair of types $t_i, t_j \in \mathcal{T}$ it holds that $\mathcal{D}(t_i) \subseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \supseteq \mathcal{D}(t_j)$ or $\mathcal{D}(t_i) \cap \mathcal{D}(t_j) = \emptyset$. $\mathcal{V}$ is a denumerable set of variable symbols, each $v \in \mathcal{V}$ has a type $\mathrm{t}_{\mathrm{var}}(v) \in \mathcal{T}$.

$\mathcal{P}$ is a set of **predicate symbols**, each predicate $p \in \mathcal{P}$ has **arity** $\mathrm{ar}(p) \in \mathbb{N}_0$ and, in case $\mathrm{ar}(p) \geq 1$, an associated type $\mathrm{t}_{\mathrm{pred}}(p, i) \in \mathcal{T}$ for every $i \in \{1, ..., \mathrm{ar}(p)\}$. An **atom** is of the form $p(s_1, \ldots, s_n)$, where $p \in \mathcal{P}$ is a predicate symbol, $n = \mathrm{ar}(p)$ is the arity of p, and each s_i is either an object $o \in \mathcal{D}(\mathrm{t}_{\mathrm{pred}}(p, i))$, or a variable $v \in \mathcal{V}$ with $\mathcal{D}(\mathrm{t}_{\mathrm{var}}(v)) \subseteq \mathcal{D}(\mathrm{t}_{\mathrm{pred}}(p, i))$. For a given atom $\alpha = p(s_1, \ldots, s_n)$, $\mathcal{V}[\alpha] \subset \mathcal{V}$ denotes a set of variables appearing in the atom, i.e., $\mathcal{V}[\alpha] = \{s_1, \ldots, s_n\} \cap \mathcal{V}$, and $\mathcal{P}[\alpha] = p$ denotes the predicate of α. Given a set of atoms X, we define $\mathcal{V}[X] = \bigcup_{x \in X} \mathcal{V}[x]$ and $\mathcal{P}[X] = \bigcup_{x \in X} \mathcal{P}[x]$. A **ground atom** is an atom α such that $\mathcal{V}[\alpha] = \emptyset$.

Atoms can be connected into a **formula** with connectives $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction) with their

standard meaning. A conjunction of atoms (i.e., positive literals) $a_1 \wedge \ldots \wedge a_n$ is also identified with the corresponding set of atoms $\{a_1, \ldots, a_n\}$, and we will use them interchangeably when suitable. To simplify the notation, an empty conjunction is interpreted as true ($\top$), and an empty disjunction is interpreted as false ($\bot$). Given a formula Φ, $\mathcal{V}[\Phi]$ denotes the set of all variables appearing in all atoms from Φ, and $\mathcal{P}[\Phi]$ is the set of all predicates appearing in Φ. A formula Φ such that $\mathcal{V}[\Phi] = \emptyset$ is called a ground formula.

An **action** $a \in \mathcal{A}$ is a tuple $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$ where $\mathrm{pre}(a)$ is an arbitrary formula called **precondition**, and $\mathrm{add}(a)$ and $\mathrm{del}(a)$ are conjunctions of atoms, called **add effect**, and **delete effect**, respectively. By $\mathcal{V}[a] = \mathcal{V}[\mathrm{pre}(a)] \cup \mathcal{V}[\mathrm{add}(a)] \cup \mathcal{V}[\mathrm{del}(a)]$ we denote a set of variables appearing in the action. For every pair of actions $a_i, a_j \in \mathcal{A}$, $a_i \neq a_j$, it holds that $\mathcal{V}[a_i] \cap \mathcal{V}[a_j] = \emptyset$. A **ground action** is an action a such that $\mathcal{V}[a] = \emptyset$.

ψ_I and ψ_G are sets of ground atoms (i.e., conjunctions), called **initial state** and **goal**, respectively.

We also assume to have the binary equality predicate symbol ($=$) written in the infix notation, e.g., $x = y$, with its standard meaning. Formally, we assume that $= \in \mathcal{P}$, and $(o = o) \in \psi_I$ for every $o \in \mathcal{B}$ and the equality predicate does not appear in any add or delete effect. Given a ground formula Φ s.t. $\mathcal{P}[\Phi] \subseteq \{=\}$, we say Φ is true if ψ_I entails Φ and false otherwise (assuming the standard interpretation of ψ_I: ground atoms in ψ_I are true, otherwise they are false).

An action $a \in \mathcal{A}$ is called **normalized** if $\mathrm{pre}(a)$ is a conjunction of atoms (positive literals), and a PDDL task $\mathcal{P}$ is called normalized if every action $a \in \mathcal{A}$ is normalized.

Note that every PDDL task can be normalized, but it may incur an exponential blow-up if the planning task contains disjunctions (or existential quantifiers). Most planners (even lifted ones) nowadays require a normalized PDDL task, so the normalization is usually done in the pre-processing step.

Next, we define substitutions as total functions over objects and variables that are identity on objects, and respect domains of variable types. Moreover, we define restrictions of substitutions to some set of variables so that the restriction is identity everywhere outside the given variables. We use substitutions as a basis for moving from lifted to ground representation and for a unification between atoms.

Definition 2. A total function $\sigma : \mathcal{V} \cup \mathcal{B} \mapsto \mathcal{V} \cup \mathcal{B}$ is called a **substitution** if $\sigma(o) = o$ for every object $o \in \mathcal{B}$, and for every variable $v \in \mathcal{V}$ it holds that either $\sigma(v) \in \mathcal{D}(\mathrm{t}_{\mathrm{var}}(v))$, or $\sigma(v) \in \mathcal{V}$ and $\mathcal{D}(\mathrm{t}_{\mathrm{var}}(\sigma(v))) \subseteq \mathcal{D}(\mathrm{t}_{\mathrm{var}}(v))$. We write σx to denote $\sigma(x)$.

We extend σ to atoms, actions, sets of variables, sets of atoms, sets of actions, and formulae, i.e., given an atom $p(x_1, \ldots, x_n)$, $\sigma p(x_1, \ldots, x_n)$ denotes $p(\sigma x_1, \ldots, \sigma x_n)$; given an action $a \in \mathcal{A}$, $\sigma a = \langle \sigma\mathrm{pre}(a), \sigma\mathrm{add}(a), \sigma\mathrm{del}(a) \rangle$; given a set of variables or atoms or actions $X = \{x_1, \ldots, x_n\}$, $\sigma X = \{\sigma x_1, \ldots, \sigma x_n\}$; and given a formula Φ, $\sigma\Phi$ denotes the same formula where every $x \in \mathcal{V} \cup \mathcal{B}$ is replaced with σx.

We write $\sigma\sigma'$ for a composition of two substitutions, i.e., $\rho = \sigma\sigma'$ if $\rho(x) = \sigma(\sigma'(x))$ for every $x \in \mathcal{V} \cup \mathcal{B}$.

Given a substitution σ and a set of variables V, $\sigma|_V$ denotes a **restriction** of σ to V, i.e., $\sigma|_V(v) = \sigma(v)$ for every $v \in V$ and $\sigma|_V(v) = v$ for every other variable $v \notin V$. Given a substitution σ and an atom or set of atoms X, $\sigma|_X$ denotes $\sigma|_{\mathcal{V}[X]}$.

Now, we define a grounding over a given set of variables V as a more specific substitution that maps each variable $v \in V$ to an object and it is identity everywhere else (i.e., it is a restriction to V).

Definition 3. Given a set of variables $V \in \mathcal{V}$, a substitution σ is called a **grounding** over V if $\sigma(v) \in \mathcal{D}(\mathrm{t}_{\mathrm{var}}(v))$ for every variable $v \in V$ and $\sigma(v) = v$ for every $v \notin V$, i.e., σ maps every variable from V to an object, and it is identity everywhere else.

A set of all groundings over $V \in \mathcal{V}$ is denoted by $\mathcal{G}_V$. For a variable (object, atom, or action) x, $\mathcal{G}_V(x)$ denotes the set $\{\sigma x \mid \sigma \in \mathcal{G}_V\}$. For a set of variables (set of atoms, or set of actions) X, $\mathcal{G}_V(X)$ denotes the set $\bigcup_{\sigma \in \mathcal{G}_V} \sigma X$. Given an atom or set of atoms X, $\mathcal{G}_X$ is a shorthand for $\mathcal{G}_{\mathcal{V}[X]}$.

With groundings as means to move from lifted to grounded representation, we are ready to define STRIPS tasks and full groundings of PDDL tasks obtained by replacing all variables with all possible combinations of objects.

Definition 4. A **STRIPS task** Π is specified by a tuple $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$, where $\mathcal{F}$ is a set of facts, and $\mathcal{O}$ is a set of operators. A **state** $s \subseteq \mathcal{F}$ is a set of facts, $s_I \subseteq \mathcal{F}$ is an **initial state** and $s_G \subseteq \mathcal{F}$ is a **goal** specification. An **operator** o is a tuple $o = \langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o) \rangle$, where $\mathrm{pre}(o) \subseteq \mathcal{F}$ is a precondition of o, and $\mathrm{add}(o) \subseteq \mathcal{F}$ and $\mathrm{del}(o) \subseteq \mathcal{F}$ are its add and delete effect, respectively. All operators are well-formed, i.e., $\mathrm{add}(o) \cap \mathrm{del}(o) = \emptyset$ and $\mathrm{pre}(o) \cap \mathrm{add}(o) = \emptyset$. An operator o is **applicable** in a state s if $\mathrm{pre}(o) \subseteq s$. The **resulting state** of applying an applicable operator o in a state s is the state $o[s] = (s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. A state s is a **goal state** if $s_G \subseteq s$. A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that o_i is applicable in s_{i-1} and $s_i = o_i[s_{i-1}]$ for $i \in \{1, \ldots, n\}$. The resulting state of this application is $\pi[s_0] = s_n$. π is called a **plan** if $\pi[s_I] \supseteq s_G$,

A set of facts $F \subseteq \mathcal{F}$ is **reachable** if there exists an operator sequence π such that $F \subseteq \pi[s_I]$, otherwise it is called **unreachable**. A state s is a **dead-end state** if $s_G \not\subseteq s$ and there is no applicable operator sequence π such that $s_G \subseteq \pi[s]$. An operator $o \in \mathcal{O}$ is called **unreachable operator** if $\mathrm{pre}(o)$ is unreachable, and it is called **dead-end operator** if for every reachable state s such that $\mathrm{pre}(o) \subseteq s$ it holds that s is a dead-end state.

Definition 5. Given a normalized PDDL task $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$, the **full grounding** of $\mathcal{P}$ is a STRIPS task $\Pi^{\mathrm{full}}_{\mathcal{P}} = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ constructed as follows.

Let $A = \bigcup_{a \in \mathcal{A}} \mathcal{G}_{\mathcal{V}[a]}(a)$, and $X = \psi_I \cup \psi_G \cup \bigcup_{a \in A}(\mathrm{pre}(a) \cup \mathrm{add}(a) \cup \mathrm{del}(a))$. Then $\mathcal{F} := \{f_x \mid x \in X\}$ where f_x denotes a fact corresponding to the ground atom x, $s_I := \{f_x \mid x \in \psi_I\}$, $s_G := \{f_x \mid x \in \psi_G\}$, and $\mathcal{O} := \{o_a \mid a \in A\}$ with $\mathrm{pre}(o_a) = \{f_x \mid x \in \mathrm{pre}(a)\}$, $\mathrm{add}(o_a) = \{f_x \mid x \in \mathrm{add}(a)\} \setminus \mathrm{pre}(o_a)$, and $\mathrm{del}(o_a) = \{f_x \mid x \in \mathrm{del}(a)\} \setminus \{f_x \mid x \in \mathrm{add}(a)\}$.

In practice, STRIPS tasks are created from PDDL tasks using relaxed reachability and applying various other techniques like irrelevance analysis (e.g., Helmert 2009). We use the full grounding as a tool for proving certain properties of lifted structures. The basic observation here is that if an operator is unreachable or dead-end in the full grounding, then it must be unreachable or dead-end, respectively, also in a more restricted grounding.

3 Pruning with Lifted Mutex Groups

Here, we summarize previous findings related to pruning using (lifted) mutex groups relevant to our approach. Mutex groups are state invariants stating that at most one of the facts from a mutex group can be part of any reachable state. Fact-alternating mutex groups (fam-groups) are more specific mutex groups that are monotonic in the sense that the number of facts from a fam-group contained in a reachable state cannot increase when transitioning to another state.

Definition 6. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_I, s_G \rangle$ denote a STRIPS planning task. A set of facts $M \subseteq \mathcal{F}$ is called (i) a **mutex group** if $|M \cap s| \leq 1$ for every reachable state s; and (ii) a **fact-alternating mutex group** (fam-group) if $|M \cap s_I| \leq 1$ and $|M \cap \mathrm{add}(o)| \leq |M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)|$ for every operator $o \in \mathcal{O}$.

It is easy to see that if an operator has a precondition containing two facts from some mutex group, then such operator cannot be applied in any reachable state and therefore it is unreachable. Furthermore, it was shown by Fišer and Komenda (2018) that every fam-group is a mutex group, and that fam-groups can be used for detecting dead-end operators: For every fam-group M and an operator o applicable in a reachable state s it holds that $|M \cap o[s]| \leq |M \cap s|$. So, if M contains a fact from the goal and for every reachable state s, where the operator o is applicable, it holds that $M \cap s \neq \emptyset$ and $M \cap o[s] = \emptyset$, then o is a dead-end operator.

Proposition 7. *Let Π be a STRIPS planning task with operators $\mathcal{O}$ and facts $\mathcal{F}$, let $M \subseteq \mathcal{F}$ denote a set of facts, and let $o \in \mathcal{O}$ denote an operator.*

(A) If M is a mutex group and $|M \cap \mathrm{pre}(o)| \geq 2$, then o is an unreachable operator.

(B) If M is a fam-group and $|M \cap s_G| \geq 1$ and $|M \cap \mathrm{pre}(o) \cap \mathrm{del}(o)| \geq 1$ and $|M \cap \mathrm{add}(o)| = 0$, then o is a dead-end operator.

Lifted mutex groups are schematic (first-order) structures whose grounding results in the set of mutex groups on the ground (STRIPS) level. In this work, we focus on lifted mutex groups proposed by Helmert (2009) in the context of translation from PDDL to finite domain representation, and further studied by Fišer (2020) in the context of operator pruning during grounding process. In particular, what is important for us here is that Fišer showed that the lifted mutex groups proposed by Helmert are actually lifted descriptions of fam-groups with all the properties that come with it including the potential for pruning dead-end operators.

In Definition 8, we formally introduce lifted mutex groups and fam-groups by adapting the definition of Fišer (2020) to our notation. In Proposition 9, we formally state how lifted mutex groups (fam-groups) can be used for pruning operators. Although these are not new findings, we provide a very brief proof to make it easier to connect lifted mutex groups and the conditions on operator pruning to their grounded counterparts discussed above.

Definition 8. Let $\Pi^{\mathrm{full}}_{\mathcal{P}}$ denote the full grounding of $\mathcal{P}$ with facts $\mathcal{F}$, let ϕ and ψ denote disjoint sets of variables, i.e., $\phi, \psi \subset \mathcal{V}$ and $\phi \cap \psi = \emptyset$, and let ν denote a set of atoms such that $\mathcal{V}[\nu] = \phi \cup \psi$. The set of atoms ν is called **lifted mutex group** (**lifted fam-group**) with **fixed variables** ϕ and **counted variables** ψ if for every $\sigma \in \mathcal{G}_\phi$ it holds that $\{f_x \in \mathcal{F} \mid x \in \sigma(\mathcal{G}_\psi(\nu))\}$ is a mutex group (fam-group) in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.

Proposition 9. *Let $\mathcal{P} = \langle \mathcal{B}, \mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{A}, \psi_I, \psi_G \rangle$ denote a normalized PDDL task, let ν denote a lifted fam-group with fixed variables ϕ and counted variables ψ, let a denote a ground action and o_a its counterpart in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.*

(A) If there exists a grounding $\sigma \in \mathcal{G}_\phi$ such that $|\sigma(\mathcal{G}_\psi(\nu)) \cap \mathrm{pre}(a)| \geq 2$, then o_a is unreachable in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.

(B) If there exists a grounding $\sigma \in \mathcal{G}_\phi$ such that $|\sigma(\mathcal{G}_\psi(\nu)) \cap \psi_G| \geq 1$ and $|\sigma(\mathcal{G}_\psi(\nu)) \cap \mathrm{pre}(a) \cap \mathrm{del}(a)| \geq 1$ and $|\sigma(\mathcal{G}_\psi(\nu)) \cap \mathrm{add}(a)| = 0$, then o_a is a dead-end operator in $\Pi^{\mathrm{full}}_{\mathcal{P}}$.

Proof. Existence of σ implies $\{f_x \mid x \in \sigma(\mathcal{G}_\psi(\nu))\}$ is a fam-group in $\Pi^{\mathrm{full}}_{\mathcal{P}}$. So, (A) follows directly from Proposition 7(A) and (B) follows from Proposition 7(B). □

Example 10. As an example, consider the following lifted fam-group from the Barman domain: $\nu = \{\mathrm{holding}(v, c), \mathrm{handempty}(v)\}$, where v is a fixed variable of type hand, and c is a counted variable of type container. The lifted fam-group ν expresses that each hand can hold at most one container. Suppose, we have two hands $\mathcal{D}(\mathrm{hand}) = \{\mathtt{h}_1, \mathtt{h}_2\}$, and two containers $\mathcal{D}(\mathrm{container}) = \{\mathtt{t}_1, \mathtt{t}_2\}$. Now, we have two possible groundings $\sigma_1, \sigma_2 \in \mathcal{G}_{\{v\}}$: σ_1 maps v to $\mathtt{h}_1$ and therefore $\sigma_1(\mathcal{G}_{\{c\}}(\nu)) = \{\mathrm{holding}(\mathrm{h}_1, \mathtt{t}_1), \mathrm{holding}(\mathrm{h}_1, \mathtt{t}_2), \mathrm{handempty}(\mathrm{h}_1)\}$, and σ_2 maps v to $\mathtt{h}_2$ and therefore $\sigma_2(\mathcal{G}_{\{c\}}(\nu))$ is the same as $\sigma_1(\mathcal{G}_{\{c\}}(\nu))$ except every $\mathtt{h}_1$ is replaced with $\mathtt{h}_2$, i.e., $\sigma_2(\mathcal{G}_{\{c\}}(\nu)) = \{\mathrm{holding}(\mathrm{h}_2, \mathtt{t}_1), \mathrm{holding}(\mathrm{h}_2, \mathtt{t}_2), \mathrm{handempty}(\mathrm{h}_2)\}$.

4 Unifiers as Formulae

As Proposition 9 requires a grounded action and lifted mutex group, we use the notion of unifiers as a basis for the generalization of conditions under which a lifted mutex group can be used for pruning. A unifier is a specific type of substitution σ such that, given a set of atoms X, σX is a singleton.

Definition 11. Let $X = \{a_1, \ldots, a_n\}$ denote a set of atoms. A substitution σ is called **unifier** for X if σ is a restriction to $\mathcal{V}[X]$ and $\sigma a_1 = \ldots = \sigma a_n$. A unifier σ for X is called a **most general unifier** (MGU) for X if, for every unifier τ for X, there exists a unifier ρ such that $\tau = \rho\sigma$.

To simplify the formalization, we tacitly assume than, unless explicitly specified otherwise, every unifier σ for a set of atoms X maps every variable $v \in \mathcal{V}[X]$ either to itself or to a fresh variable not used anywhere else, i.e., it never

"recycles" any variable already used in any atom or other substitution.

It is well-known, that if there exists a unifier, then there also exists a most general unifier and it is unique up to a renaming of variables (e.g., Buss 1998). Moreover, note that the grounding and unifier are complementary notions that can be combined using composition. Given a set of atoms X, a grounding τ over $\mathcal{V}[X]$ is also a unifier if τX is a singleton, and for every unifier σ for X there exists a grounding τ over $\mathcal{V}[\sigma X]$ such that $\tau\sigma X$ is a ground singleton.

Now, let us go back to Proposition 9. Let a denote an action, and let τ denote a grounding over $\mathcal{V}[a]$. The part (A) states that we can prune a ground action τa if we find a lifted mutex group ν and its grounding containing at least two different atoms from the precondition of τa. In other words, we need to find a substitution σ that unifies two different pairs of atoms at the same time, each pair having one atom from ν and one atom from $\mathrm{pre}(a)$. Then we know that, for every grounding ρ over $\mathcal{V}[\sigma a]$, $\rho\sigma a$ can be pruned. And similarly for the part (B): We need to unify an atom from a lifted famgroup ν with some atom from the precondition and delete effect of an action a, and at the same time we need to unify another atom from ν with the goal, and also make sure that the substitution achieving both unifications does not unify any atom from ν with any atom from the add effect of a. In the following Proposition 12, we show that substitutions unifying multiple sets of atoms at the same time are compositions of multiple unifiers.

Proposition 12. *Let X and Y denote sets of atoms, and let ρ denote a substitution restricted to $\mathcal{V}[X \cup Y]$. If $\rho|_X$ is a unifier for X and $\rho|_Y$ is a unifier for Y, then there exist a unifier σ for X and a unifier σ' for σY such that $\sigma'\sigma = \rho$.*

Proof. Since $\rho|_X$ is a unifier for X, we already have $\sigma = \rho|_X$. Now it is enough to show that there exist an MGU μ for σY and a substitution τ restricted to $\mathcal{V}[\mu\sigma Y]$ s.t. $(\tau\mu\sigma)|_Y = \rho|_Y$ and $(\tau\mu\sigma)|_X = \rho|_X$, and therefore $(\tau\mu)|_{\sigma X}$ is identity. So, as σ is identity on all variables $\mathcal{V}[Y] \setminus \mathcal{V}[X]$, we just need to make sure $(\tau\mu)|_{\sigma X}$ is identity which follows trivially from the fact that for every $v \in \mathcal{V}[X] \cap \mathcal{V}[Y]$ it holds that $\rho v = \rho|_{\mathcal{V}[X]\cap\mathcal{V}[Y]}(v) = \rho|_X(v) = \rho|_Y(v)$. □

Determining substitutions is not enough for our purpose, because we want to compile the conditions from Proposition 9 into preconditions of actions so that the preconditions are satisfied only when the actions are not unreachable or dead-end. Therefore, we need to express these conditions as formulae. Definition 13 shows how to transform a substitution into a formula, and after that we prove that this formula captures the substitution over a certain set of variables exactly. Note, that the definition requires a substitution and a set of variables, but makes no assumption about the relation between them. This is intentional, because we need to find (unifying) substitutions over variables from both lifted mutex groups and actions, but we want to have formulae only over the action variables.

Definition 13. Given a substitution σ and a set of variables $V \subset \mathcal{V}$, we define the formulae $\Phi^{\mathrm{var}}_{\sigma,V}$, $\Phi^{\text{var-obj}}_{\sigma,V}$, $\Phi^{\mathrm{subtype}}_{\sigma,V}$, and $\Phi^{\mathrm{unifier}}_{\sigma,V}$ as follows:

$$\Phi^{\mathrm{var}}_{\sigma,V} = \bigwedge_{\{v,w\}\in X} (v = w), \tag{1}$$

where $X = \{\{v, w\} \subseteq V \mid v \neq w, \sigma v = \sigma w\}$;

$$\Phi^{\text{var-obj}}_{\sigma,V} = \bigwedge_{v\in Y} (v = \sigma v), \tag{2}$$

where $Y = \{v \in V \mid \sigma v \in \mathcal{B}\}$;

$$\Phi^{\mathrm{subtype}}_{\sigma,V} = \bigwedge_{v\in Z} \Big(\bigvee_{o\in\mathcal{D}(\mathrm{t_{var}}(\sigma v))} (v = o)\Big), \tag{3}$$

where $Z = \{v \in V \mid \sigma v \notin \mathcal{B}, \mathrm{t_{var}}(\sigma v) \neq \mathrm{t_{var}}(v)\}$[1]; and

$$\Phi^{\mathrm{unifier}}_{\sigma,V} = \Phi^{\mathrm{var}}_{\sigma,V} \wedge \Phi^{\text{var-obj}}_{\sigma,V} \wedge \Phi^{\mathrm{subtype}}_{\sigma,V}. \tag{4}$$

Moreover, given an atom or set of atoms X, $\Phi^{\mathrm{var}}_{\sigma,X}$, $\Phi^{\text{var-obj}}_{\sigma,X}$, $\Phi^{\mathrm{subtype}}_{\sigma,X}$, and $\Phi^{\mathrm{unifier}}_{\sigma,X}$ are shorthands for $\Phi^{\mathrm{var}}_{\sigma,\mathcal{V}[X]}$, $\Phi^{\text{var-obj}}_{\sigma,\mathcal{V}[X]}$, $\Phi^{\mathrm{subtype}}_{\sigma,\mathcal{V}[X]}$, and $\Phi^{\mathrm{unifier}}_{\sigma,\mathcal{V}[X]}$, respectively.

Eventually, we want to show that, given an action a and a grounding τ over $\mathcal{V}[a]$, $\tau(\Phi^{\mathrm{unifier}}_{\sigma,V})$ is true if and only if σ unifies some atom from a lifted mutex group with an atom q from the action such that $V = \mathcal{V}[q]$. This way, we will be able to express unifiers (and their compositions) as formulae and integrate them (or their negation) into preconditions of actions. The formula $\Phi^{\mathrm{unifier}}_{\sigma,V}$ is constructed so that it captures all properties of the given substitution σ relevant to our purpose. First, $\Phi^{\mathrm{var}}_{\sigma,V}$ makes sure that if the given substitution σ unifies two variables v and w, then the grounding τ has to assign the same object to both v and w. Second, $\Phi^{\text{var-obj}}_{\sigma,V}$ deals with variables v that σ maps to a specific object, i.e., in this case τ needs to map v to σv. Third, σ may map a variable v to another variable with a different type (in which case $\mathcal{D}(\mathrm{t_{var}}(\sigma v)) \subseteq \mathcal{D}(\mathrm{t_{var}}(v))$). So, $\Phi^{\mathrm{subtype}}_{\sigma,V}$ is constructed in such a way that $\tau(\Phi^{\mathrm{subtype}}_{\sigma,V})$ is true only if τ maps every such variable v to an object from $\mathcal{D}(\mathrm{t_{var}}(\sigma v))$.

Before we get to the main result of this section formulated in Theorem 15, we show that the formula $\Phi^{\mathrm{unifier}}_{\sigma,Y}$ is sufficient to capture unification over the set of atoms Y whenever σ is a unifier over some superset of Y.

Proposition 14. *Let X denote a set of atoms, let $Y \subseteq X$, let σ denote a unifier for X, and let $\tau \in \mathcal{G}_Y$ denote a grounding over $\mathcal{V}[Y]$. If $\tau(\Phi^{\mathrm{unifier}}_{\sigma,Y})$ is true, then τ is a unifier for Y.*

Proof. Let $Y = \{a_1, \ldots, a_n\}$ and since σ is a unifier for Y, we can also write $a_i = p(x_{i,1}, \ldots, x_{i,m})$ for every $i \in \{1, \ldots, n\}$ where $m = \mathrm{ar}(p)$. To prove the claim by contradiction, we assume $\tau(\Phi^{\mathrm{unifier}}_{\sigma,Y})$ is true and τ is not a unifier for Y, and therefore there exist $i, j \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, m\}$ such that $i \neq j$ and $\tau x_{i,k} \neq \tau x_{j,k}$ and therefore also $x_{i,k} \neq x_{j,k}$. Moreover, since σ is a unifier, we have that $\sigma x_{i,k} = \sigma x_{j,k}$, and therefore we need to investigate two cases. (i) (w.l.o.g.) $x_{i,k} \in \mathcal{B}$ and $x_{j,k} \in \mathcal{V}[Y]$ and

[1] Recall that for every variable v and substitution σ it holds that $\mathcal{D}(\mathrm{t_{var}}(\sigma v)) \subseteq \mathcal{D}(\mathrm{t_{var}}(v))$ (Definition 2).

$\tau x_{j,k} \neq x_{i,k}$: Since $\tau x_{j,k} \neq x_{i,k}$ and $\sigma x_{i,k} = \sigma x_{j,k}$, it follows that $\sigma x_{j,k} = x_{i,k}$. Therefore from $\tau\Phi^{\text{var-obj}}_{\sigma,Y}$, we have that $\tau x_{j,k} = \sigma x_{j,k} = x_{i,k}$ which is a contradiction.

(ii) $x_{i,k}, x_{j,k} \in \mathcal{V}[Y]$ and $x_{i,k} \neq x_{j,k}$ and $\tau x_{i,k} \neq \tau x_{j,k}$: Since $\sigma x_{i,k} = \sigma x_{j,k}$, we have from $\tau\Phi^{\text{var}}_{\sigma,Y}$ that $\tau x_{i,k} = \tau x_{j,k}$ which is a contradiction. □

Let σ denote a unifier for some set of atoms X, and let $Y \subseteq X$. In the following theorem, we show that if $\tau(\Phi^{\text{unifier}}_{\sigma,Y})$ is true for some grounding τ over Y, then τ can be extended over the variables from X so that such an extension is a unifier for the whole set X. And conversely, if a grounding τ over $\mathcal{V}[X]$ (s.t. $\tau = \rho\sigma$ for some substitution ρ) is a unifier for X, then $\tau(\Phi^{\text{unifier}}_{\sigma,Y})$ is true. In this sense, $\Phi^{\text{unifier}}_{\sigma,Y}$ captures the unifier σ perfectly. In particular, whenever σ is an MGU, because every unifier for X can be expressed using an MGU that is unique up to a renaming of variables.

Theorem 15. *Let X denote a set of atoms, let $Y \subseteq X$, let σ denote a unifier for X, and let $\tau \in \mathcal{G}_Y$ denote a grounding over $\mathcal{V}[Y]$. Then $\tau(\Phi^{\text{unifier}}_{\sigma,Y})$ is true if and only if there exists a unifier ρ for σX such that $(\rho\sigma)|_Y = \tau$ (i.e., $\rho\sigma$ is a unifier for X, and for every $v \in \mathcal{V}[Y]$ it holds that $\rho\sigma v = \tau v$).*

Proof. "⇒": It follows from Proposition 14 that τ is a unifier for Y. Let ρ denote a substitution such that $\rho\sigma v = \tau v$ for every $v \in \mathcal{V}[Y]$ and it is identity everywhere else (i.e., $(\rho\sigma)|_{\mathcal{V}[X]\setminus\mathcal{V}[Y]} = \sigma|_{\mathcal{V}[X]\setminus\mathcal{V}[Y]}$). Now we show that ρ is well-defined. First, we need to show that ρ is a function, i.e., we need to show that for every $v, v' \in \mathcal{V}[Y]$ s.t. $v \neq v'$ and $\sigma v = \sigma v'$ it holds that $\tau v = \tau v'$ (because then we can set $\rho(\sigma v)$ to τv for every $v \in \mathcal{V}[Y]$). And, indeed, it follows from $\tau(\Phi^{\text{var}}_{\sigma,Y})$ being true that $\tau v = \tau v'$ for every $v, v' \in \mathcal{V}[Y]$ s.t. $v \neq v'$ and $\sigma v = \sigma v'$.

Second, we need to show that ρ is a substitution, i.e., for every $w \in \mathcal{V}[\sigma Y]$ it holds that $\rho w \subseteq \mathcal{D}(\mathrm{t}_{\text{var}}(w))$. This trivially holds for $v \in \mathcal{V}[Y]$ s.t. $\mathrm{t}_{\text{var}}(v) = \mathrm{t}_{\text{var}}(\sigma v)$ as τ is a unifier for Y. For $v \in \mathcal{V}[Y]$ s.t. $\mathrm{t}_{\text{var}}(v) \neq \mathrm{t}_{\text{var}}(\sigma v)$, it follows from $\tau(\Phi^{\text{subtype}}_{\sigma,Y})$ being true that $\tau v \in \mathcal{D}(\mathrm{t}_{\text{var}}(\sigma v))$. Therefore ρ is, indeed, a substitution, and therefore $\rho\sigma$ is a unifier for X because σX is a singleton.

"⇐": Suppose $\tau(\Phi^{\text{unifier}}_{\sigma,Y}) = \rho\sigma(\Phi^{\text{unifier}}_{\sigma,Y})$ is false. If $\rho\sigma(\Phi^{\text{var}}_{\sigma,Y})$ is false, then there are variables $v, w \in \mathcal{V}[Y]$ such that $v \neq w$ and $\sigma v = \sigma w$, but also $\rho\sigma v \neq \rho\sigma w$ which is a contradiction. If $\rho\sigma(\Phi^{\text{var-obj}}_{\sigma,Y})$ is false, then there is a variable $v \in \mathcal{V}[Y]$ such that $\sigma v \in \mathcal{B}$ and $\rho\sigma v \neq \sigma v$, but ρ is identity on objects. If $\rho\sigma(\Phi^{\text{subtype}}_{\sigma,Y})$ is false, then there is a variable $v \in \mathcal{V}[Y]$ such that σv is a variable and $\rho\sigma v \notin \mathcal{D}(\mathrm{t}_{\text{var}}(\sigma v))$, which is a contradiction with Definition 2. □

Corollary 16. *Let X, Y, and τ be as before, and let σ denote an MGU for X. Then $\tau(\Phi^{\text{unifier}}_{\sigma,Y})$ is true if and only if there exists a unifier ρ for X such that $\rho|_Y = \tau$.*

Proof. Since every unifier ρ can be expressed as $\rho'\sigma$ for some substitution ρ', it follows from Theorem 15. □

5 Pruning as Compilation

In this section, we show how to compile conditions from Proposition 9 (or rather their negation) into preconditions of actions so that their groundings are not unreachable or dead-end. But before that consider the following situation.

Suppose we have the lifted fam-group ν from Example 10, and two atoms $q = \mathrm{holding}(\mathtt{h}_1, \mathtt{t}_1)$ and $q' = \mathrm{holding}(\mathtt{h}_1, \mathtt{t}_2)$. Moreover, suppose we want to find a substitution ρ that unifies $p = \mathrm{holding}(v, c)$ from ν with both q and q' to prove that there is no reachable state containing both q and q' (i.e., no reachable state where a hand holds two containers at the same time). This is clearly not possible, because ρ can map c to only one of $\mathtt{t}_1$ and $\mathtt{t}_2$. We can, however, resolve this problem by creating a copy p' of p with the same fixed variable v, but rename the counted variable c to another variable c' with the same type. Then we can prove there is no state containing both q and q' using Proposition 9(A) if we find a substitution unifying $\{p, q\}$ and $\{p', q'\}$, because $\mathcal{G}_{\{c\}}(p) = \mathcal{G}_{\{c'\}}(p')$. To resolve this (rather technical) issue with counted variables, we introduce the following notion of renaming for lifted mutex groups.

Definition 17. Given a lifted mutex group ν with fixed variables ϕ and counted variables $\psi = \{c_1, \ldots, c_n\}$, ${}^{\mathrm{r}}\nu$ denotes a copy of ν where fixed variables remain the same, but counted variables ψ are renamed to a fresh set of variables ${}^{\mathrm{r}}\psi = \{c'_1, \ldots, c'_n\}$ not appearing in ϕ or ψ, i.e., $\mathrm{t}_{\text{var}}(c'_i) = \mathrm{t}_{\text{var}}(c_i)$ for every $i \in [n]$ and every c_i is replaced with c'_i.

Proposition 18. *Let ν denote a lifted mutex group with fixed variables ϕ and counted variables ψ. Then (i) $\nu \cup {}^{\mathrm{r}}\nu$ is a lifted mutex group, and (ii) for every $\sigma \in \mathcal{G}_\phi$ it holds that $\sigma(\mathcal{G}_\psi(\nu)) = \sigma(\mathcal{G}_{\psi \cup {}^{\mathrm{r}}\psi}(\nu \cup {}^{\mathrm{r}}\nu))$.*

Proof. (i) follows directly from (ii) and (ii) follows from a simple observation that $\mathcal{G}_\psi(\nu) = \mathcal{G}_{{}^{\mathrm{r}}\psi}({}^{\mathrm{r}}\nu)$ by definition. □

Now, we are ready to put everything together. The construction of formula identifying unreachable operators using a given lifted mutex group ν is described in Algorithm 1, and we prove in Theorem 19 that the formula captures the pruning power of ν exactly, i.e., the formula is true if and only if the corresponding ground action can be identified as unreachable using ν. The idea is quite simple. We simply iterate over all possible pairs of atoms from the precondition of the given action a, and over all pairs of atoms from the lifted mutex group ν, where the atoms from ν do not share counted variables (lines 2 and 3). Then we try to find MGUs unifying atoms from $\mathrm{pre}(a)$ and ν (lines 4 and 5). If we find them, then we construct the corresponding formula ($\Phi^{\text{unifier}}_{\sigma,q} \wedge \Phi^{\text{unifier}}_{\sigma'\sigma,q'}$ on line 7). Finally, on lines 8 to 10, we add the formula ensuring that whenever we unify two atoms from $\mathrm{pre}(a)$ with two atoms from ν, these atoms differ and therefore the whole formula $\tau(\Phi^{\text{unifier}}_{\sigma,q} \wedge \Phi^{\text{unifier}}_{\sigma'\sigma,q'} \wedge \neg\Phi^{\text{unifier}}_{\sigma'',\{q,q'\}})$ is true for some grounding $\tau \in \mathcal{G}_{\mathcal{V}[a]}$ only if $\tau(a)$ is unreachable.

Theorem 19. *Let $a \in \mathcal{A}$ denote an action, let $\tau \in \mathcal{G}_{\mathcal{V}[a]}$ denote a grounding of a, let ν denote a lifted mutex group with fixed variables ϕ and counted variables ψ, and let Φ*

Algorithm 1: Formula identifying unreachable operators

Input: A normalized action a, a lifted mutex group ν
Output: A formula Φ

1 $\Phi \leftarrow \bot$;
2 **for each** $q, q' \in \mathrm{pre}(a)$ *s.t.* $q \neq q'$ **do**
3 **for each** $p \in \nu$, $p' \in {}^{\mathrm{r}}\nu$ *s.t.* $p \neq p'$ **do**
4 $\sigma \leftarrow$ find a MGU for $\{p, q\}$;
5 $\sigma' \leftarrow$ find a MGU for $\{\sigma p', \sigma q'\}$;
6 **if** *both* σ *and* σ' *exist* **then**
7 $\Phi' \leftarrow \Phi^{\mathrm{unifier}}_{\sigma,q} \wedge \Phi^{\mathrm{unifier}}_{\sigma'\sigma,q'}$;
8 $\sigma'' \leftarrow$ find a MGU for $\{q, q'\}$;
9 **if** σ'' *exists* **then**
10 $\Phi' \leftarrow \Phi' \wedge \neg\Phi^{\mathrm{unifier}}_{\sigma'',\{q,q'\}}$;
11 $\Phi \leftarrow \Phi \vee \Phi'$;

Algorithm 2: Formula identifying dead-end operators

Input: A normalized action a, a lifted fam-group ν
Output: A formula Φ

1 $\Phi \leftarrow \bot$;
2 **for each** $q \in \psi_G$, $r \in \mathrm{pre}(a)$, $s \in \mathrm{del}(a)$ **do**
3 **for each** $p \in \nu$, $p' \in {}^{\mathrm{r}}\nu$ **do**
4 $\sigma \leftarrow$ find a MGU for $\{p, q\}$;
5 $\sigma' \leftarrow$ find a MGU for $\{\sigma p', r, s\}$;
6 **if** *both* σ *and* σ' *exist* **then**
7 $\Phi' \leftarrow \Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{r,s\}}$;
8 **for each** $p'' \in {}^{\mathrm{r}}\nu$, $t \in \mathrm{add}(a)$ **do**
9 $\sigma'' \leftarrow$ find a MGU for $\{\sigma'\sigma p'', \sigma'\sigma t\}$;
10 **if** σ'' *exists* **then**
11 $\Phi' \leftarrow \Phi' \wedge \neg\Phi^{\mathrm{unifier}}_{\sigma''\sigma'\sigma,t}$;
12 $\Phi \leftarrow \Phi \vee \Phi'$;

denote a formula returned by Algorithm 1 for a and ν. Then $\tau\Phi$ is true if and only if there exists a grounding $\tau' \in \mathcal{G}_\phi$ such that $|\tau'(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{pre}(a)| \geq 2$.

Proof. It follows from Proposition 18, that we can safely replace ν with $\nu \cup {}^{\mathrm{r}}\nu$, and $\mathcal{G}_\psi(\nu)$ with $\mathcal{G}_{\psi \cup {}^{\mathrm{r}}\psi}(\nu \cup {}^{\mathrm{r}}\nu)$.

"$\Rightarrow$": Since $\tau\Phi = \tau(\Phi_1 \vee \ldots \vee \Phi_m)$ is true, we select an arbitrary formula Φ_i such that $\tau\Phi_i$ is true (created in one of the inner cycles of Algorithm 1). So we have $q, q' \in \mathrm{pre}(a)$ s.t. $q \neq q'$, $p \in \nu$ and $p' \in {}^{\mathrm{r}}\nu$ s.t. $p \neq p'$, an MGU σ for $\{p, q\}$, an MGU σ' for $\{\sigma p', \sigma q'\}$, and an MGU σ'' for $\{q, q'\}$ if such σ'' exists.

First, if σ'' exists, then $\tau(\Phi^{\mathrm{unifier}}_{\sigma'',\mathcal{V}[\{q,q'\}]})$ is false, so it follows from Corollary 16 that $\tau q \neq \tau q'$. If σ'' does not exist, then $\tau q \neq \tau q'$ follows trivially.

Second, from Theorem 15 and $\tau(\Phi^{\mathrm{unifier}}_{\sigma,\mathcal{V}[q]})$ being true it follows that we have a unifier ρ for $\{\sigma p, \sigma q\}$ s.t. $(\rho\sigma)|_q = \tau|_q$. And since τq is a ground atom it follows that $\rho|_{\sigma p}$ is a grounding over $\mathcal{V}[\sigma p]$. Similarly, from Theorem 15 and $\tau(\Phi^{\mathrm{unifier}}_{\sigma'\sigma,\mathcal{V}[q']})$ being true, we have a unifier ρ' for $\{\sigma'\sigma p', \sigma'\sigma q'\}$ s.t. $(\rho'\sigma'\sigma)|_{q'} = \tau|_{q'}$ and $\rho'|_{\sigma'\sigma p'}$ is a grounding over $\mathcal{V}[\sigma'\sigma p']$. Therefore $\rho'\sigma'|_{\sigma p'}$ is a grounding over $\mathcal{V}[\sigma p']$. Therefore ρ and $\rho'\sigma'$ agree on all variables $\mathcal{V}[\sigma p] \cap \mathcal{V}[\sigma p']$. And since $\mathcal{V}[p] \cap \mathcal{V}[p'] \subseteq \phi$ it clearly follows that there exists $\tau' \in \mathcal{G}_\phi$ that agrees with $\rho\sigma$ and $\rho'\sigma'\sigma$ on all variables $\mathcal{V}[p] \cap \mathcal{V}[p']$. So it follows that $\rho\sigma p, \rho'\sigma'\sigma p' \in \tau'(\mathcal{G}_\psi(\nu))$ and $\rho\sigma p = \tau q \in \tau\mathrm{pre}(a)$ and $\rho'\sigma'\sigma p' = \tau q' \in \tau\mathrm{pre}(a)$ and $\tau p \neq \tau p'$.

"$\Leftarrow$": Since $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{pre}(a)| \geq 2$, there exist ground atoms g and g' such that $g, g' \in \tau\mathrm{pre}(a)$, and $g, g' \in \tau'(\mathcal{G}_\psi(\nu))$, and $g \neq g'$. And therefore there exist atoms $p \in \nu$, $p' \in {}^{\mathrm{r}}\nu$, and $q, q' \in \mathrm{pre}(a)$ such that $p \neq p'$, $q \neq q'$, $\mathcal{V}[p] \cap \mathcal{V}[p'] \subseteq \phi$, and there exists a grounding τ'' s.t. $\tau''p = \tau q = g \neq g' = \tau''p' = \tau q'$. Therefore there exists a grounding $\rho \in \mathcal{G}_{\{p,p',q,q'\}}$ such that $\rho|_{\{p,q\}}$ is a unifier for $\{p, q\}$ and $\rho|_{\{p',q'\}}$ is a unifier for $\{p', q'\}$ and $\rho|_{\{q,q'\}} = \tau|_{\{q,q'\}}$. So, it follows from Proposition 12 that there exist (a) a unifier for X and therefore also an MGU σ for X; (b) a unifier for σY and therefore also an MGU σ' for σY; and (c) a grounding τ'' over $\mathcal{V}[\sigma'\sigma\{p, q, p', q'\}]$ s.t. $\tau''\sigma'\sigma = \rho$ and therefore $(\tau''\sigma'\sigma)|_{\{q,q'\}} = \tau|_{\{q,q'\}}$. Therefore, it follows from Theorem 15 that both $\tau(\Phi^{\mathrm{unifier}}_{\sigma,\{p,q\}})$ and $\tau(\Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{p',q'\}})$ are true. Finally, from $\tau q \neq \tau q'$ and Theorem 15 it follows that either there does not exist an MGU σ'' for $\{q, q'\}$ or if it exists, then $\tau(\Phi^{\mathrm{unifier}}_{\sigma'',\{q,q'\}})$ is false. □

Example 20. Consider the lifted fam-group ν from Example 10, and the action fill-shot that has a precondition containing atoms $\mathrm{holding}(x_h, x_c)$ and $\mathrm{handempty}(y_h)$, where x_h, x_c, and y_h are the action's variables (x_h and y_h are of type hand, and x_c has a type container). In this case, Algorithm 1 will be able to find an MGU σ unifying $\mathrm{holding}(x_h, x_c)$ and $\mathrm{holding}(v, c)$, and another MGU σ' unifying $\sigma(\mathrm{handempty}(y_h))$ and $\mathrm{handempty}(v)$. So, as $\mathrm{holding}(x_h, x_c)$ and $\mathrm{handempty}(y_h)$ cannot be unified, we get the resulting formula $x_h = y_h$ (see Equation (1)).

Algorithm 2 encapsulates the construction of a formula identifying dead-end operators given a lifted fam-group ν, which we formally prove in Theorem 21. The idea is essentially the same as before. We try to unify an atom from ν with some goal ground atom, and then another atom from ν with one atom from the action's precondition and one atom from its delete effect (lines 4 and 5). Note that since q is a ground atom, then $\sigma r = r$ and $\sigma s = s$. If we find such unifiers, we construct the formula $\Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{r,s\}}$ (line 7), which is equal to $\Phi^{\mathrm{unifier}}_{\sigma,q} \wedge \Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{r,s\}}$ because q is a ground atom and therefore $\Phi^{\mathrm{unifier}}_{\sigma,q} = \Phi^{\mathrm{unifier}}_{\sigma,\emptyset} = \top$. Finally, the cycle on lines 8 to 11 constructs the formula excluding cases where the action adds another atom unifiable with some atom from ν. Note that ${}^{\mathrm{r}}\nu$ on line 3 and on line 8 are different as each one denotes a fresh renaming of the lifted fam-group ν.

Theorem 21. *Let $a \in \mathcal{A}$ denote an action, let $\tau \in \mathcal{G}_{\mathcal{V}[a]}$ denote a grounding of a, let ν denote a lifted mutex with fixed variables ϕ and counted variables ψ, and let Φ denote a formula returned by Algorithm 2 for a and ν. Then $\tau\Phi$ is true if and only if there exists a grounding $\rho \in \mathcal{G}_\phi$ such that $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{pre}(a) \cap \tau\mathrm{del}(a)| \geq 1$ and $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{add}(a)| = 0$ and $|\rho(\mathcal{G}_\psi(\nu)) \cap \psi_G| \geq 1$.*

Proof Sketch. Note that every atom from goal $q \in \psi_G$ is a ground atom, i.e., $\mathcal{V}[q] = \emptyset$, and $\Phi^{\mathrm{unifier}}_{\sigma,\emptyset}$ is true for any

Algorithm 3: Pruning via compilation

Input: A normalized action a, a set of lifted fam-groups M
Output: A modified action a

1 $\Phi \leftarrow \bot$;
2 **for each** $\nu \in M$ **do**
3 $\quad \Phi' \leftarrow$ Algorithm 1 for a and ν;
4 $\quad \Phi'' \leftarrow$ Algorithm 2 for a and ν;
5 $\quad \Phi \leftarrow \Phi \vee \Phi' \vee \Phi''$;
6 $\mathrm{pre}(a) \leftarrow \mathrm{pre}(a) \wedge \neg\Phi$;

substitution σ. Furthermore note that, for the same reason, the unifier σ from line 4 is identity over all variables $\mathcal{V}[a]$ and therefore $\sigma r = r \in \mathrm{pre}(a)$ and $\sigma s = s \in \mathrm{del}(a)$. The rest can be proved analogously to Theorem 19: In Algorithm 2, we go over every possible combination of atoms $p \in \psi_G$, $r \in \mathrm{pre}(a)$, $s \in \mathrm{del}(a)$, find an atom from ν that can be unified with the goal (the MGU σ), and under such unification we find another atom that can be unified with both precondition and delete effect (the MGU σ'). Then we have from Theorem 15 that (i) if $\tau(\Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{r,s\}})$ is true, then there exists a corresponding groundings $\rho \in \mathcal{G}_\phi$ such that $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{pre}(a) \cap \tau\mathrm{del}(a)| \geq 1$ and $|\rho(\mathcal{G}_\psi(\nu)) \cap \psi_G| \geq 1$; and (ii) we have that if $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{pre}(a) \cap \tau\mathrm{del}(a)| \geq 1$ and $|\rho(\mathcal{G}_\psi(\nu)) \cap \psi_G| \geq 1$ hold, then the corresponding MGUs σ and σ' exist and $\tau(\Phi^{\mathrm{unifier}}_{\sigma'\sigma,\{r,s\}})$ is true.

For the condition on the add effect, we can continue with the same line of argument. Under the mapping $\sigma\sigma'$, we look for the unification of the lifted fam-group ν and add effect $\mathrm{add}(a)$ and whenever we find such unifier σ'' for some $t \in \mathrm{add}(a)$, $\tau(\Phi^{\mathrm{unifier}}_{\sigma''\sigma'\sigma,\mathcal{V}[t]})$ is false if and only if $|\rho(\mathcal{G}_\psi(\nu)) \cap \tau\mathrm{add}(a)| = 0$ holds. □

Example 22. Consider the following lifted fam-group from the Barman domain: $\{\mathrm{contains}(v_1, v_2), \mathrm{clean}(v_1), \mathrm{used}(v_1, c)\}$, where v_1 and v_2 are fixed variables of type shot and cocktail, respectively, and c is a counted variable of type beverage (which is a parent type of cocktail). Consider an action empty-shot having the atom $\mathrm{contains}(x_s, x_b)$ in its precondition and delete effect, and only the atom $\mathrm{empty}(x_s)$ in its add effect (x_s is of type shot, and x_b of type beverage). And suppose the goal includes $\mathrm{contains}(\mathtt{s}, \mathtt{k})$, where s is an object of type shot and k is an object of type cocktail. Now, we can find an MGU σ unifying $\mathrm{contains}(v_1, v_2)$ with $\mathrm{contains}(\mathtt{s}, \mathtt{k})$, and then unify $\sigma(\mathrm{contains}(v_1, v_2)) = \mathrm{contains}(\mathtt{s}, \mathtt{k})$ and $\mathrm{contains}(x_s, x_b)$. Finally, since $\mathrm{empty}(x_s)$ is not unifiable with any atom from ν, we have the resulting formula $(x_s = \mathtt{s}) \wedge (x_b = \mathtt{k})$ (see Equation (2)). That is, the action empty-shot leads to a dead end whenever $x_s = \mathtt{s}$ and $x_b = \mathtt{k}$, because such action deletes the atom $\mathrm{contains}(\mathtt{s}, \mathtt{k})$ required by the goal, but the action does not replace it with any other atom allowing to recover this atom again in any subsequent state.

Algorithm 3 shows how to compile the pruning with lifted mutex groups directly into preconditions of actions by combining Algorithm 1 and Algorithm 2. We simply find the formulae corresponding to conditions where actions are unreachable or dead-end, and then add negations of those formulae into the preconditions.

Both Algorithm 1 and Algorithm 2, and therefore also Algorithm 3, are polynomial in the size of the input normalized PDDL task, because unification between two atoms is linear and all for-cycles iterate over a polynomially bounded number of elements. However, these algorithms can generate complicated formulae. So, if negations of the generated formulae contain disjunctions and they are added to actions' preconditions, then the normalization of the resulting PDDL task can cause an exponential blow-up of the number of normalized actions.

6 Experimental Evaluation

The proposed method was implemented[2] in C and evaluated on a cluster with Intel E5-2660 processors and 30 minutes time and 4 GB memory limit for each process. Since we proved in Theorem 19 and 21 that the pruning power of our method is exactly the same as the one proposed by Fišer (2020), we decided to focus our experimental evaluation on different aspects here. So, besides evaluating the compilation and grounding process, we focus on the application of our method in lifted planning. We use the so-called hard-to-ground (HTG) domains (Masoumi, Antoniazzi, and Soutchanski 2015; Gnad et al. 2019; Corrêa et al. 2020; Lauer et al. 2021), and domains from optimal and satisficing tracks of International Planning Competitions (IPCs) from 1998 to 2018. We removed domains with conditional effects, fully grounded domains, and duplicates, leaving 56 domains and 3 464 tasks out of which 33 domains and 1 703 tasks were affected by the compilation (29 domains and 1 485 tasks were affected by Algorithm 1; 10 domains and 544 tasks were affected by Algorithm 2).

We compare baseline (`base`), i.e., no compilation, with three variants of Algorithm 3: `un` denotes the variant where only unreachable operators are pruned (i.e., Algorithm 3 skips line 4); `de` denotes the variant pruning only dead-end operators; and `un-de` denote full Algorithm 3. Two translators from PDDL to finite domain representation are evaluated: Our translator written in C (`dl`) and the translator from Fast Downward (Helmert 2006) written in Python, both implementing datalog-based grounding introduced by Helmert (2009). Lastly, we test our lifted planner implementing ideas proposed by Corrêa et al. (2020) and using a successor generator based on the same code as the grounder in `dl`: We use A* with the blind heuristic (`blind`), A* with the $h^{\max}$ heuristic (`hmax`), greedy best-first-search with h^{add} (`hadd`) (Bonet and Geffner 2001), and lazy greedy best-first search with h^{add} (`lz-hadd`).

The compilation time, i.e., the time needed for parsing the input PDDL files, normalization, application of Algorithm 3, and subsequent second normalization, was almost always negligible: the median over all tasks was less than one millisecond, and mean was 565 milliseconds. It took more than one second only in the domains Organic-synthesis, Barman, and Nomystery, where the median (mean) time was

[2] https://gitlab.com/danfis/cpddl, branch icaps23-pruning-lmg

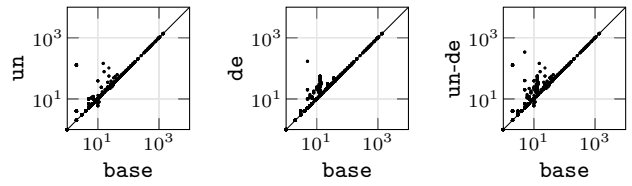

Figure 1: Number of normalized actions.

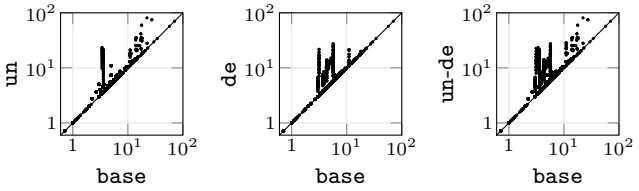

Figure 2: Average number of atoms in preconditions of normalized actions.

2.8 (124), 23.2 (29.1), and 2.4 (2.8) seconds, respectively.

As the compilation can generate disjunctions and therefore the number of normalized actions can grow exponentially, we compare the number of normalized actions with and without the compilation in Figure 1. Note that the number of actions can also decrease, because it may happen that the whole lifted action is unreachable or dead-end (i.e., whenever Algorithm 1 or 2 return $\top$). This happened in 16 Organic-synthesis, and 6 Airport tasks. The number of actions more than doubled only in 20 tasks from Organic-synthesis, the whole Woodworking domain, 22 tasks from Trucks, and 6 tasks from Scanalyzer. The biggest increase was due to the unreachability conditions in the tasks from the Scanalyzer domain (from 2 to 128) and one task from Organic-synthesis (from 5 to 340).

After the compilation, preconditions of actions can get more complicated. To quantify how much and how often, we compared the average number of atoms in preconditions of normalized actions before and after the compilation (Figure 2). Note that preconditions of normalized actions are conjunctions of atoms, so the compared numbers correspond to average sizes of preconditions. The average number of atoms in actions' preconditions increases at most 7.2 times (for `un-de`), it increases more than five times in only 27 tasks for `un-de`, ten tasks for `de`, and 17 tasks for `un`. Furthermore, it increases more than two times in 297, 200, and 82 tasks for `un-de`, `de`, and `un`, respectively. So, considering the number of affected tasks is 1 703 (1 485 by `un`, and 544 by `de`), it does not happen very often that the size of preconditions grows significantly, but it tends to grow more with the dead-end pruning (`de`) than with the unreachability pruning (`un`).

On one hand, the increased number of lifted actions can never result in a higher number of ground actions. On the other hand, more actions and more complicated preconditions can result in a slower grounding (even with pruning) or a bigger memory footprint of the grounding process.

`dl-un-de` was able to ground four more tasks in Blocksworld, and one more task in Caldera and Childsnack, and one less task in Tpp and Visitall (all HTG domains).

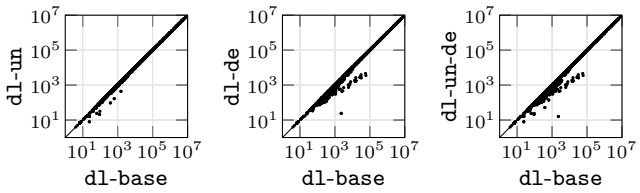

Figure 3: Number of STRIPS operators after grounding.

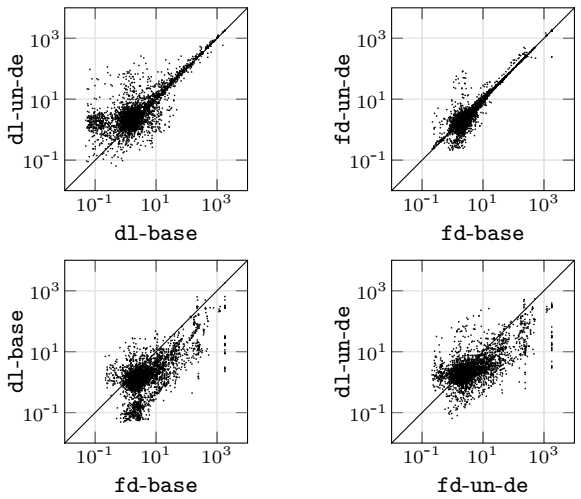

Figure 4: Translation time in seconds.

`fd-un-de` grounds one more task in Visitall and two fewer in Logistics (both HTG). Also note that `dl-un-de` can ground 280 more HTG tasks than `fd-un-de`, which is not surprising given `fd-un-de` is implemented in Python. We also show in Figure 3 the number of STRIPS operators after grounding, but since the pruning power of our method is exactly the same as that of Fišer (2020), a more detailed analysis of the number of pruned operators can be found there.

The compilation can slow down the translation because of more complicated preconditions, but it can also speed up the translation because pruning can make the state space of the relaxed task smaller. Figure 4 shows the comparison of running times of the whole translation process. The mean runtime of `dl-base` and `dl-un-de` was 32.4 and 35.9 seconds, respectively, and the median was 1.9 and 2.7 seconds, respectively. So, our compilation leads more often to a slowdown than to a speedup. In case of `fd-base` and `fd-un-de`, the mean runtime was about 51 seconds for both, and the median was 3.3 seconds for `fd-base`, and 3.1 seconds `fd-un-de`. This means that the runtime of `fd-base` was influenced by the compilation to a lesser extent than `dl-base`.

Applying pruning of unreachable operators (`un`) in lifted planners is more often detrimental than beneficial: `blind-un` solves ten fewer tasks than `blind`, `hmax-un` five fewer than `hmax`, `hadd-un` 17 fewer than `hadd`, and `lz-hadd-un` 25 fewer than `lz-hadd`. The reason is that pruning of unreachable operators cannot decrease the branching factor as these operators are never considered in the forward

domain	blind		hmax		hadd		lz-hadd	
	base	de	base	de	base	de	base	de
airport (50)	16	**17**	16	16	22	22	21	21
barman (74)	4	4	4	4	4	4	**6**	5
floortile (80)	2	**10**	2	2	14	14	13	**16**
parcprinter (70)	14	**18**	20	20	49	**57**	49	**57**
trucks (30)	2	**5**	3	**4**	8	**9**	9	**11**
woodworking (98)	12	**13**	0	0	0	0	0	0
others (142)	46	46	42	42	111	111	115	115
Σ (544)	96	113	87	88	208	217	213	225

Table 1: Number of solved tasks.

search. However, it can increase the time needed by the successor generator because of the more complicated preconditions of actions, and it could increase informativeness of (delete-relaxed) heuristics because the pruning may remove unreachable operators that are reachable in the relaxed planning task. Unfortunately, we did not observe any change of the heuristic value for initial states, but we sometimes observed a slowdown of the successor generator resulting in the aforementioned lower coverage. For this reason, we focused on the pruning of dead-end operators (`de`) because this kind of pruning can decrease the branching factor.

Table 1 summarizes the number of solved tasks by the considered lifted planners over 10 domains and 544 tasks affected by the `de` compilation. Executing Algorithm 2 on the remaining 1 159 tasks, that were not modified by the `de` compilation, was almost never detrimental. The only exceptions was one task in the Pipesworld-tankage domain and one task in the Organic-synthesis domain solved by `hmax` but not by `hmax-de`, and one task in the Logistics domain solved by `lz-hadd` but not by `lz-hadd-de`. Table 1 shows that dead-end pruning compiled into actions' preconditions can be sometimes beneficial. However, it seems to be less beneficial when the search is guided by heuristics which is not surprising. The comparison of the number of expanded states on Figure 5 shows that, indeed, the pruning has a positive effect on the search. And the comparison of the number of states reported by the heuristic as dead-end states (Figure 6) is also often higher with pruning. Overall, we see only a small improvement, but for a small cost. That is, it does not happen very often that applying pruning through compilation has a detrimental effect, but sometimes it can be beneficial. We think this might change in future, when we see a more diverse set of hard-to-ground domains, and other techniques for lifted planning.

7 Conclusion

Following on the previous work of Fišer (2020) focusing on pruning of unreachable and dead-end actions using lifted mutex groups, we have shown that such pruning can be left to standard algorithms for grounding of PDDL tasks or successor generation, because it can be compiled directly into preconditions of actions. Furthermore, we have shown that the proposed compilation preserves the pruning power of the input lifted mutex groups perfectly. The experimental evaluation shows that the compilation can be beneficial for both grounding and lifted planning and it is rarely detrimental.

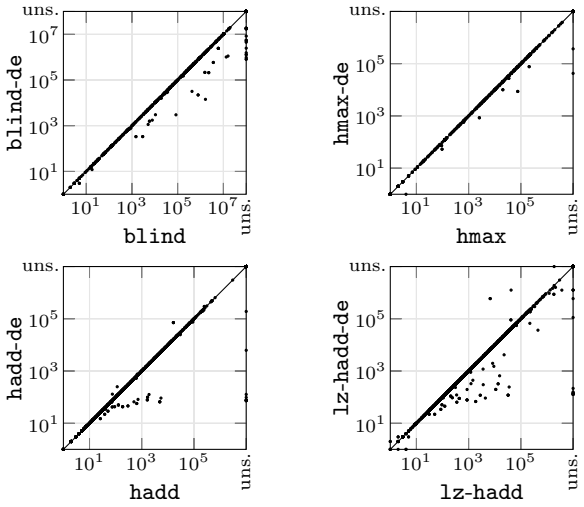

Figure 5: Number of expanded states (before the last f-layer for `blind` and `hmax`; all for `hadd` and `lz-hadd`).

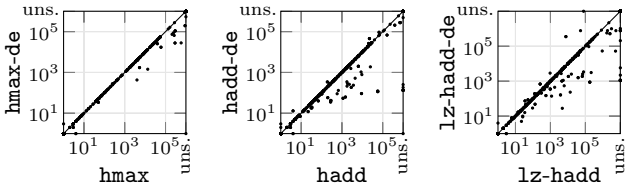

Figure 6: Number of dead-end states.

Since the proposed method provides a general framework for expressing unification of multiple atoms as formulae, we think it would be interesting to see whether a similar approach can utilize also different types of lifted invariants (e.g., Rintanen 2000). Another question we left for future research is how the compilation affects symmetries (Röger, Sievers, and Katz 2018), or whether the additional information provided by the compilation cannot be utilized in finding better join orders in grounders and (lifted) successor generators (Helmert 2009; Corrêa et al. 2020).

Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—project number 389792660—TRR 248 (see https://perspicuous-computing.science).

References

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1–2): 5–33.

Buss, S. R. 1998. Chapter I - An Introduction to Proof Theory. In Buss, S. R., ed., *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, 1–78. Elsevier.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS'20)*, 80–89.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, 9835–9842.

Fišer, D.; and Komenda, A. 2018. Fact-Alternating Mutex Groups for Classical Planning. *Journal of Artificial Intelligence Research*, 61: 475–521.

Fišer, D.; Gnad, D.; Katz, M.; and Hoffmann, J. 2021. Custom-Design of FDR Encodings: The Case of Red-Black Planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*, 4054–4061.

Fox, M.; and Long, D. 1998. The Automatic Inference of State Invariants in TIM. *Journal of Artificial Intelligence Research*, 9: 367–421.

Gerevini, A.; and Schubert, L. 1998. Inferring State-Constraints for Domain Independent Planning. In *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence (AAAI'98)*, 905–912.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan - Partial Grounding in Classical Planning. In *Proceedings of the 33nd AAAI Conference on Artificial Intelligence (AAAI'19)*, 7602–7609.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation from $h^{\max}$ to h^m. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 354–357.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, 140–149.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Lauer, P.; Torralba, A.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*, 4119–4126.

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling Organic Chemistry and Planning Organic Synthesis. In *Global Conference on Artificial Intelligence (GCAI'15)*, volume 36 of *EPiC Series in Computing*, 176–195.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *The AI Magazine*, 21(2): 35–55.

Mukherji, P.; and Schubert, L. K. 2005. Discovering Planning Invariants As Anomalies In State Descriptions. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, 223–230.

Rintanen, J. 2000. An Iterative Algorithm for Synthesizing Invariants. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI'00)*, 806–811.

Röger, G.; Sievers, S.; and Katz, M. 2018. Symmetry-Based Task Reduction for Relaxed Reachability Analysis. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS'18)*, 208–217.