Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Endomorphisms of Lifted Planning Problems

Rostislav Horčík[1], **Daniel Fišer**[1,2]
[1] Czech Technical University in Prague, Faculty of Electrical Engineering, Prague, Czech Republic
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
xhorcik@fel.cvut.cz, danfis@danfis.cz

Abstract

Classical planning tasks are usually modeled in the PDDL which is a schematic language based on first-order logic. Nevertheless, most of the current planners turn this first-order representation into a propositional one via the grounding process. It is well known that the grounding process may cause an exponential blowup. Therefore it is important to detect which grounded atoms are redundant in a sense that they are not necessary for finding a plan and therefore the grounding process does not need to generate them. This is usually done by a relaxed reachability analysis, which can be improved by employing structural symmetries. Symmetries are bijective self-maps preserving the structure of the PDDL task. In this paper, we introduce a new method which is based on self-maps preserving the structure but which need not be bijective. We call these maps PDDL endomorphisms and we show that they can be used for pruning of redundant objects even if they appear in a reachable atom. We formulate the computation of endomorphisms as a constraint satisfaction problem (CSP) that can be solved by an off-the-shelf CSP solver.

Introduction

Classical planning tasks are usually modeled in the standard PDDL language based on first-order logic (McDermott 2000). Nevertheless, a vast majority of planners do not work with this first-order (lifted) representation but with a simpler one based on propositional logic (i.e., grounded representation); usually either STRIPS (Fikes and Nilsson 1971) or SAS$^+$ (Bäckström and Nebel 1995). In order to translate the PDDL representation into a propositional one, one has to go through the so-called grounding process where the first-order action schemata are translated into propositional ones by substituting all possible combinations of objects for the variables. This may result in an exponentially larger representation than the original PDDL representation.

There are, however, some approaches addressing this problem. The most commonly used one employs a relaxed reachability analysis and grounds only those atoms which possibly occur in a reachable state (Hoffmann and Nebel 2001; Helmert 2009). The reachability analysis can be further improved by the detection of unreachable actions and actions leading to dead-end states (Fišer 2020). Another approach utilizes structural symmetries (Röger, Sievers, and Katz 2018; Sievers et al. 2019) to ground the planning task only partially, because the rest of the grounded atoms and actions can be generated by the symmetries. Furthermore, Gnad et al. (2019) applied machine learning techniques to learn which actions are more likely to be a part of a plan, resulting in only partially grounded planning task (without a guarantee that the grounded part contains a plan from the original planning task).

Although the reachability analysis can greatly reduce the number of grounded atoms, its output still might be quite large. Consider for instance a planning task involving a large map consisting of some locations and their connections where all the locations are reachable but only a small part of the map is actually needed in order to construct a plan. In this case, the reachability analysis cannot prune the irrelevant part of the map. Here, we develop a method which is able to overcome this limitation to some extent.

PDDL tasks are defined over a set of objects $\mathcal{B}$ serving as values to be substituted for the variables during the grounding process. We introduce a novel notion of PDDL endomorphism which will serve as a basis for a new pruning method that, given a PDDL task $\mathcal{P}$, produces a reduced PDDL task $\mathcal{P}'$ which has a possibly smaller set of objects $\mathcal{B}' \subseteq \mathcal{B}$, but preserves at least one (optimal) plan from $\mathcal{P}$. The resulting reduced task can be solved by any planner accepting inputs formulated in PDDL. In this work, we focus mainly on the theoretical analysis that will hopefully help us better understand the underlying structures of PDDL planning tasks.

Now, we informally explain and illustrate on an example which redundant objects our method aims to find. Consider a logistic task where we have a map of cities connected by roads, a fleet of trucks and we are supposed to transport packages to their destinations. The map is usually modeled by a symmetric binary relation `road` (or its equivalent) capturing which cities are directly connected (possibly enriched by costs). Thus the map can be viewed as a graph whose vertices are cities and edges represent the roads (see Figure 1).

Assume we are supposed to move packages from the city c_0 to c_3 and all our trucks are located in c_0. To solve the task, it clearly suffices to use the path from c_0 to c_3 via c_2. Thus the city c_1 is in this sense redundant. In case of optimal planning, we have to further ensure that the path via c_2 is

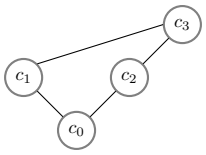

Figure 1: A simple graph representing the road connection among cities.

cheaper than the path via c_1. Whether we can really safely remove the city c_1 from our PDDL task depends on other relations and actions in the task.

In order to find out if we can prune c_1, our method utilizes the following tools. First, it leverages structure-preserving maps called homomorphisms of first-order structures (see Hodges 1997, Chapter 1). In fact, it is well known that such homomorphisms preserve all positive existential formulas (e.g. Hodges 1997). As states are sets of grounded atoms, they can be viewed as relational first-order structures having the same universe $\mathcal{B}$. Our method searches for a suitable map $\sigma\colon \mathcal{B} \to \mathcal{B}$ (we call it a PDDL endomorphism) allowing us to replace states and actions in a plan π involving the city c_1 by another states and actions not involving c_1. Thus we obtain an alternative plan π' such that σ then acts as homomorphism between the original states and the replaced ones. Moreover, the actions in the new plan π' contain only objects in the image of σ, i.e., $\sigma(\mathcal{B}) = \{\sigma(b) \mid b \in \mathcal{B}\}$. Thus all the objects in $\mathcal{B} \setminus \sigma(\mathcal{B})$ are redundant and can be pruned.

As homomorphisms preserve only positive existential formulas, they cannot properly treat delete effects in actions as delete effects involve implicitly negation. In order to overcome this problem, our method applies lifted mutex groups which are first-order invariants allowing to over-approximate the set of reachable states. There are methods how to compute them from the PDDL formulation of a task (Helmert 2009; Fišer 2020). Using the lifted mutex groups, we can identify a subset of objects on which σ has to act as the identity map. Fixing these objects ensures σ to be a homomorphism between states of π and π'.

To find a suitable map σ, we formulate a constraint satisfaction problem (CSP) which can be solved by an off-the-shelf CSP solver. More precisely, the solutions to our CSP give us possible maps σ preserving plans. Clearly, among them, we are interested in those whose image is the smallest possible. In that case, we can prune the largest amount of objects.

PDDL Planning Tasks

We consider the normalized non-numeric, non-temporal PDDL tasks without conditional effects and negative preconditions, and with all formulas being conjunctions of atoms (represented as sets of atoms). We also split effects of PDDL actions into add effects (positive literals) and delete effects (negative literals) directly in the definition below to simplify the presentation. In contrast to the normalization of PDDL tasks described by Helmert (2009), we do not support axioms (derived predicates) and we keep and utilize PDDL types. Although we disregard conditional effects in the theory for sake of simplicity, our implementation supports them.

Given a set S, we denote a tuple $\langle s_1, \ldots, s_n \rangle$ of elements from S shortly by $\vec{s}$. The i-th component of $\vec{s}$ is denoted $s_i \in S$ and $|\vec{s}|$ stands for the length of $\vec{s}$. For an indexed family of sets S_i by a set of indexes I their Cartesian product is denoted by $\prod_{i \in I} S_i$.

Definition 1. Let $\mathcal{B}$ be a non-empty set of **objects**. A **type** τ is a subset of $\mathcal{B}$. A **type hierarchy** $\mathcal{T}$ over $\mathcal{B}$ is a non-empty set of types such that $\mathcal{B} \in \mathcal{T}$ and for every pair of types $\tau_i, \tau_j \in \mathcal{T}$ it holds that $\tau_i \subseteq \tau_j$ or $\tau_i \supseteq \tau_j$ or $\tau_i \cap \tau_j = \emptyset$. A type τ_i is said to be **minimal** if it has no subtype, i.e., for every type $t_j \in \mathcal{T}$ either $\tau_i \subseteq \tau_j$ or $\tau_i \cap \tau_j = \emptyset$. We further assume that the union of all minimal types equals $\mathcal{B}$.

Note that the minimal types form a partition of the set $\mathcal{B}$ since they are disjoint and cover it. Although the condition on the union of minimal types is not explicitly enforced by the PDDL standard, it is always possible to extend the type hierarchy so that it satisfies this condition: For each non-minimal type τ, take all its elements which do not belong to any of its subtypes and create a new minimal type from these elements.

Definition 2. A **normalized 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 type hierarchy over $\mathcal{B}$ and $\mathcal{V}$ is a denumerable set of variable symbols, each variable $v \in \mathcal{V}$ has a type $\tau_{var}(v) \in \mathcal{T}$.

$\mathcal{P}$ is a set of **predicate symbols**, each predicate $p \in \mathcal{P}$ has **arity** $\mathrm{ar}(p) \in \mathbb{N}$ and an associated type $\tau_{pred}(p, i) \in \mathcal{T}$ for every $i \in \{1, ..., \mathrm{ar}(p)\}$. An **atom** is of the form $p(\vec{t})$, where $\vec{t} = \langle t_1, \ldots, t_n \rangle$, $p \in \mathcal{P}$ is a predicate symbol, $n = \mathrm{ar}(p)$ is the arity of p, and each t_i is either an object $b \in \tau_{pred}(p, i)$, or a variable $v \in \mathcal{V}$ with $\tau_{var}(v) \subseteq \tau_{pred}(p, i)$. A **ground atom** is an atom without variables. For an atom $p(\vec{t})$ and a tuple of variables $\vec{x}$ we also use the notation $p(\vec{x})$ to express that variables occurring in $p(\vec{t})$ are among those in $\vec{x}$. Given an atom $p(\vec{x})$, where $\vec{x}$ is a tuple of variables, and a tuple of objects $\vec{b} \in \prod_{i=1}^{|\vec{x}|} \tau_{var}(x_i)$, $p(\vec{b})$ denotes the ground atom resulting from $p(\vec{x})$ by substituting $\vec{b}$ for $\vec{x}$.

An **action schema** $a(\vec{x}) \in \mathcal{A}$ is a tuple $a = \langle \mathrm{pre}_a(\vec{x}), \mathrm{add}_a(\vec{x}), \mathrm{del}_a(\vec{x}) \rangle$ where $\mathrm{pre}_a(\vec{x})$, $\mathrm{add}_a(\vec{x})$ and $\mathrm{del}_a(\vec{x})$ are sets of atoms, called **preconditions**, **add effects**, and **delete effects**, respectively, and $\vec{x}$ is a tuple of variables occurring in any atom in $a(\vec{x})$. If we substitute a tuple of objects $\vec{b} \in \prod_{i=1}^{|\vec{x}|} \tau_{var}(x_i)$ for $\vec{x}$, we create a **ground action** (or shortly just an action). The resulting ground action is denoted by $a(\vec{b})$.

To simplify the formulation of our results, we further assume that the action schemata do not contain objects, i.e., they contain only variables. Without any loss of generality, we can remove all objects from the action schemata: For an object $b \in \mathcal{B}$ occurring in an action schema we introduce a type $\tau_b = \{b\} \in \mathcal{T}$ and refine the type hierarchy $\mathcal{T}$ so that it satisfies all the necessary conditions from Definition 1. Further, we introduce a fresh variable v_b of type τ_b and replace all occurrences of b in the action schema by v_b. Let $a(\vec{x})$ be

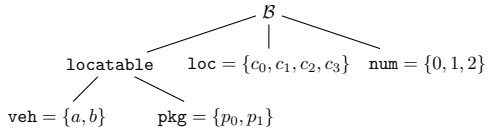

Figure 2: The type hierarchy of the transportation domain.

an action schema, $a(\vec{b})$ a corresponding ground action and $p(\vec{c}) \in \mathrm{pre}_a(\vec{b}) \cup \mathrm{add}_a(\vec{b}) \cup \mathrm{del}_a(\vec{b})$ a ground atom. The above assumption allows us to infer that all elements in $\vec{c}$ occur in $\vec{b}$.

A set of ground atoms s is called a state. The sets ψ_I and ψ_G are states, called **initial state** and **goal**, respectively. A state s is a **goal state** if $s \supseteq \psi_G$.

A ground action $a(\vec{b})$ is **applicable** in a state s if $\mathrm{pre}_a(\vec{b}) \subseteq s$. The **resulting state** of applying an applicable action $a(\vec{b})$ in a state s is the state $a(\vec{b})[\![s]\!] = (s \setminus \mathrm{del}_a(\vec{b})) \cup \mathrm{add}_a(\vec{b})$.

A sequence of ground actions $\pi = \langle a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)\rangle$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that $a_i(\vec{b}_i)$ is applicable in s_{i-1} and $s_i = a_i(\vec{b}_i)[\![s_{i-1}]\!]$ for $i \in \{1, \ldots, n\}$. The resulting state of this application is $\pi[\![s_0]\!] = s_n$. The sequence π is called a **plan** iff $\pi[\psi_I] \supseteq \psi_G$. In case of optimal planning, we assume that for each action schema $a(\vec{x})$ there is a cost function c_a assigning a cost $\mathrm{c}_a(\vec{b})$ to the ground action $a(\vec{b})$. It allows to define a cost of the plan π as $\sum_{i=1}^{n} \mathrm{c}_{a_i}(\vec{b}_i)$. A plan with a minimum cost is called **optimal**.

A state s is **reachable** if there exists an action sequence π such that $\pi[\![\psi_I]\!] = s$.

Note that a state s can be viewed as a first-order structure whose universe is $\mathcal{B}$ and every predicate symbol $p \in \mathcal{P}$ is interpreted by the relation

$$R(p, s) = \{\vec{b} \in \prod_{i=1}^{\mathrm{ar}(p)} \tau_{pred}(p, i) \mid p(\vec{b}) \in s\}.$$

There are two kinds of predicate symbols. The **static** ones and **dynamic** ones. A static predicate p is interpreted in all reachable states by the same relation (i.e., it occurs neither in add effects nor delete effects of any action). Thus its fixed interpretation is given by the initial state ψ_I. The remaining predicates are dynamic, i.e., those whose interpretation may change by the application of an action.

Example 3. Now we introduce a running example allowing us to illustrate defined concepts throughout the paper. Consider a task in the logistics domain whose type hierarchy is depicted in Figure 2. The task has two static binary predicates: $\texttt{road}(x\colon \texttt{loc}, y\colon \texttt{loc})$ represents the connectivity between locations and $\texttt{pred}(x\colon \texttt{num}, y\colon \texttt{num})$ represents ordering of vehicles' capacities. The interpretation of road is depicted in Figure 1, i.e., it is $\{\langle c_0, c_1\rangle, \langle c_1, c_0\rangle, \langle c_0, c_2\rangle, \langle c_2, c_0\rangle, \ldots\}$. The interpretation of pred is $\{\langle 0, 1\rangle, \langle 1, 2\rangle\}$. Furthermore, the task has three dynamic predicates: $\texttt{at}(v\colon \texttt{locatable}, x\colon \texttt{loc})$ describes the location of a vehicle or package, $\texttt{in}(p\colon \texttt{pkg}, v\colon \texttt{veh})$ is true if the package p is loaded in the vehicle v, and $\texttt{cap}(v\colon \texttt{veh}, n\colon \texttt{num})$ represents the current free capacity of the vehicle v. Apart from the interpretations of static predicates, the initial state ψ_I contains also atoms $\texttt{at}(a, c_0)$, $\texttt{at}(b, c_3)$, $\texttt{at}(p_0, c_0)$, $\texttt{at}(p_1, c_3)$, $\texttt{cap}(a, 2)$, $\texttt{cap}(b, 2)$. The goal is defined by $\psi_G = \{\texttt{at}(p_0, c_3), \texttt{at}(p_1, c_0)\}$. So the package p_0 is in c_0 and p_1 in c_3. The goal is to interchange their locations.

There are three action schemata $\texttt{drive}(\vec{u})$, $\texttt{pick}(\vec{v})$ and $\texttt{drop}(\vec{v})$ defined as follows:

$\vec{u} = \langle v\colon \texttt{veh}, x, y\colon \texttt{loc}\rangle$
$\mathrm{pre}_{\texttt{drive}}(\vec{u}) = \{\texttt{at}(v, x), \texttt{road}(x, y)\}$,
$\mathrm{add}_{\texttt{drive}}(\vec{u}) = \{\texttt{at}(v, y)\}$,
$\mathrm{del}_{\texttt{drive}}(\vec{u}) = \{\texttt{at}(v, x)\}$,
$\vec{v} = \langle v\colon \texttt{veh}, x\colon \texttt{loc}, p\colon \texttt{pkg}, n_1, n_2\colon \texttt{num}\rangle$
$\mathrm{pre}_{\texttt{pick}}(\vec{v}) = \{\texttt{at}(v, x), \texttt{at}(p, x), \texttt{pred}(n_1, n_2), \texttt{cap}(v, n_2)\}$,
$\mathrm{add}_{\texttt{pick}}(\vec{v}) = \{\texttt{in}(p, v), \texttt{cap}(v, n_1)\}$,
$\mathrm{del}_{\texttt{pick}}(\vec{v}) = \{\texttt{at}(p, x), \texttt{cap}(v, n_2)\}$,
$\mathrm{pre}_{\texttt{drop}}(\vec{v}) = \{\texttt{at}(v, x), \texttt{in}(p, v), \texttt{pred}(n_1, n_2), \texttt{cap}(v, n_1)\}$,
$\mathrm{add}_{\texttt{drop}}(\vec{v}) = \{\texttt{at}(p, x), \texttt{cap}(v, n_2)\}$,
$\mathrm{del}_{\texttt{drop}}(\vec{v}) = \{\texttt{in}(p, v), \texttt{cap}(v, n_1)\}$.

We can overapproximate which combinations of ground atoms might appear in a reachable state by employing lifted mutex groups (Helmert 2009; Fišer 2020). We extend our notational conventions. Given two tuples of variables $\vec{x}$ and $\vec{y}$, we denote their concatenation $\vec{x}, \vec{y}$. Then we use the notation $p(\vec{x}, \vec{y})$ for an atom with predicate p whose variables are among $\vec{x}, \vec{y}$.

Definition 4. A set of atoms $\mu(\vec{x}, \vec{y}) = \{p_1(\vec{x}, \vec{y}), \ldots, p_n(\vec{x}, \vec{y})\}$ with two tuples of distinguished variables, **fixed** variables $\vec{x}$ and **counted** variables $\vec{y}$, is said to be a **lifted mutex group** if for each reachable state s and each tuple $\vec{b} \in \prod_{i=1}^{|\vec{x}|} \tau_{var}(x_i)$ there is at most one $j \in \{1, \ldots, n\}$ and at most one tuple $\vec{c} \in \prod_{i=1}^{|\vec{y}|} \tau_{var}(y_i)$ such that $p_j(\vec{b}, \vec{c}) \in s$.

Example 5. In our running example, there are three lifted mutex groups $\mathcal{M} = \{\mu_1, \mu_2, \mu_3\}$:

$$\mu_1(v\colon \texttt{veh}, x\colon \texttt{loc}) = \{\texttt{at}(v, x)\}$$
$$\mu_2(v\colon \texttt{veh}, n\colon \texttt{num}) = \{\texttt{cap}(v, n)\}$$
$$\mu_3(p\colon \texttt{pkg}, v\colon \texttt{veh}, x\colon \texttt{loc}) = \{\texttt{at}(p, x), \texttt{in}(p, v)\},$$

where the first variable in each μ_i is fixed and the remaining ones are counted. So, it follows from μ_1 that for each (fixed) vehicle v there is at most one location x so that $\texttt{at}(v, x)$ holds in a reachable state. And similarly from μ_2, it follows that each vehicle has assigned at most one capacity. Lastly, μ_3 says that for each (fixed) package p there is at most one object $z \in \texttt{loc} \cup \texttt{veh}$ such that either $\texttt{at}(p, z)$, or $\texttt{in}(p, z)$ holds.

There are methods for the inference of lifted mutex groups from the PDDL formulation (see Helmert 2009; Fišer 2020).

We denote a set of inferred lifted mutex groups by $\mathcal{M}$. Similarly to action schemata, we assume that no object occurs in any atom from any lifted mutex group in $\mathcal{M}$.

Definition 6. A substitution $\xi\colon \mathcal{V} \to \mathcal{V}$ is a map respecting types, i.e., $\tau_{var}(\xi(v)) \subseteq \tau_{var}(v)$ for any $v \in \mathcal{V}$.

Clearly, lifted mutex groups are closed under substitutions, i.e., if $\mu(\vec{u}, \vec{v})$ is a lifted mutex group then $\mu(\xi(\vec{u}), \xi(\vec{v}))$ is a lifted mutex group as well.

PDDL Endomorphisms

In this section, we introduce a novel notion of PDDL endomorphism, and we show how to use PDDL endomorphisms to infer redundant objects, i.e., objects that are not necessary for solving the planning task.

Definition 7. A set of objects $Z \subseteq \mathcal{B}$ is called **redundant** if there is an (optimal) plan $\pi = \langle a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)\rangle$ such that none of the elements from Z occurs in π.

Given a map $\sigma\colon \mathcal{B} \to \mathcal{B}$, we will extend it elementwise to tuples, i.e., if $\vec{b} = \langle b_1, \ldots, b_n\rangle$ then $\sigma(\vec{b}) = \langle\sigma(b_1), \ldots, \sigma(b_n)\rangle$. In order to decrease the amount of parenthesis in mathematical expressions, we adopt the common convention of removing parenthesis in $\sigma(\vec{b})$, i.e., writing $\sigma\vec{b}$ instead. Further, we extend σ on states by $\sigma(s) = \{p(\sigma\vec{b}) \mid p(\vec{b}) \in s\}$. Similarly, we extend σ on sets of objects, in particular $\sigma(\mathcal{B}) = \{\sigma(b) \mid b \in \mathcal{B}\}$ denotes the image of σ on all objects.

Before we define PDDL endomorphisms, we define homomorphisms between states. Since states are in fact first-order structures, we define homomorphisms between states as the usual homomorphisms of first-order structures (Hodges 1997, Chapter 1). More precisely, a map $\sigma\colon \mathcal{B} \to \mathcal{B}$ is a **state homomorphism** from a state s to t if $\sigma(s) \subseteq t$ (i.e., $p(\vec{b}) \in s$ implies $p(\sigma\vec{b}) \in t$). We denote this fact by $\sigma\colon s \to t$. A state homomorphism $\sigma\colon s \to t$ is called **state endomorphism** on s if $s = t$. Moreover, the state endomorphism σ is called **state automorphism** on s if $\sigma(s) = s$, (i.e., it can only permute atoms in s).

Definition 8. A map $\sigma\colon \mathcal{B} \to \mathcal{B}$ is called a **PDDL endomorphism** if the following conditions are satisfied:

(P1) σ preserves types, i.e., $\sigma(\tau) \subseteq \tau$ for all types $\tau \in \mathcal{T}$,

(P2) σ is a state endomorphism on ψ_I,

(P3) σ is a state automorphism on ψ_G,

(P4) for all reachable states s, t and each ground action $a(\vec{b})$ if $\sigma\colon s \to t$ and $s' = a(\vec{b})[\![s]\!]$ then $\sigma\colon s' \to t'$ for $t' = a(\sigma\vec{b})[\![t]\!]$.

(P5) for the optimal planning, we further assume that $\mathrm{c}_a(\sigma\vec{b}) \leq \mathrm{c}_a(\vec{b})$ for all ground actions $a(\vec{b})$.

The condition (P4) states that whenever we have reachable states s, t such that σ is a state homomorphism from s to t and a ground action $a(\vec{b})$ is applicable in s, then the ground action $a(\sigma\vec{b})$ is applicable in t and σ remains a state homomorphism after applications of $a(\vec{b})$ and $a(\sigma\vec{b})$ as depicted in Figure 3 (a).

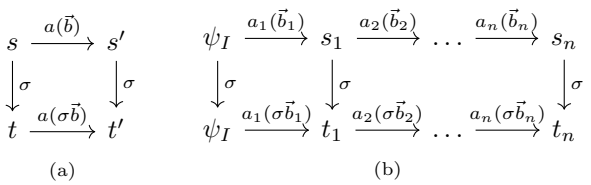

Figure 3: (a) the condition (P4), (b) the plan preservation

Now we show that PDDL endomorphisms preserve (optimal) plans.

Theorem 9. *Let σ be a PDDL endomorphism. If $\pi = \langle a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)\rangle$ is an (optimal) plan, then $\pi' = \langle a_1(\sigma\vec{b}_1), \ldots, a_n(\sigma\vec{b}_n)\rangle$ is an (optimal) plan as well.*

Proof. Let $\pi = \langle a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)\rangle$ be an (optimal) plan and σ a PDDL endomorphism. Consider the diagram depicted at Figure 3 (b). In the upper row there is the plan π starting in the initial state ψ_I leading to a goal state $s_n \supseteq \psi_G$. By (P2) $\sigma\colon \psi_I \to \psi_I$ is a state endomorphism. Thus we can apply (P4) to show that $\sigma\colon s_1 \to t_1$ is a state homomorphism, where $t_1 = a(\sigma\vec{b}_1)[\![\psi_I]\!]$. Applying (P4) repeatedly, we finally get that $\sigma\colon s_n \to t_n$. As σ is a state automorphism on ψ_G, we have $\psi_G = \sigma(\psi_G) \subseteq \sigma(s_n) \subseteq t_n$. Consequently, π' is a plan. In case π is optimal, the condition (P5) ensures that the cost of π' is less than or equal to the cost of π. As π is optimal, the cost of π' has to be equal to the cost of π. □

Informally, Theorem 9 shows that PDDL endomorphisms embed plans into the same PDDL planning task. So, it easily follows that the objects that do not belong to the image of σ are redundant, i.e., we can safely remove them from the planning task.

Corollary 10. *Let σ be a PDDL endomorphism. If there is an (optimal) plan π then there is an (optimal) plan whose actions are grounded by tuples of objects from $\sigma(\mathcal{B})$. In particular, if $\sigma(\mathcal{B}) \subsetneq \mathcal{B}$ then objects not belonging to $\sigma(\mathcal{B})$ are redundant.*

Inference of PDDL Endomorphisms

The condition (P4) is not particularly useful for the inference of PDDL endomorphisms in practice, because in order to use it, we would need to know the whole state space in the first place. In this section, we replace the condition (P4) with another condition that can be evaluated directly on the PDDL representation if we also utilize additional information about reachable states provided by lifted mutex groups.

Before we state the new condition, note that the condition (P4) requires that $a(\sigma\vec{b})$ is applicable in t given $a(\vec{b})$ is applicable in s and $\sigma\colon s \to t$ (see Figure 3 (b) again). This follows immediately since the homomorphisms preserve positive existential formulas, in particular conjunctions of atoms. The preconditions $\mathrm{pre}_a(\vec{b}) \subseteq s$ can be viewed as a conjunction of atoms, therefore we get $\mathrm{pre}_a(\sigma\vec{b}) \subseteq t$ as shown in the following lemma.

Lemma 11. *Let $\sigma\colon s \to t$ be a state homomorphism, $a(\vec{x})$ an action schema and $\vec{b}$ a tuple of objects such that the ground action $a(\vec{b})$ is applicable in s. Then $a(\sigma\vec{b})$ is applicable in t.*

Proof. Let $\mathrm{pre}_a(\vec{x}) = \{p_1(\vec{x}), \ldots, p_n(\vec{x})\}$. Note that $\sigma(\mathrm{pre}_a(\vec{b})) = \mathrm{pre}_a(\sigma\vec{b})$. Indeed, $\sigma(\mathrm{pre}_a(\vec{b})) = \sigma(\{p_1(\vec{b}), \ldots, p_n(\vec{b})\}) = \{p_1(\sigma\vec{b}), \ldots, p_n(\sigma\vec{b})\} = \mathrm{pre}_a(\sigma\vec{b})$. Since $\mathrm{pre}_a(\vec{b}) \subseteq s$ and $\sigma\colon s \to t$, we obtain $\mathrm{pre}_a(\sigma\vec{b}) = \sigma(\mathrm{pre}_a(\vec{b})) \subseteq \sigma(s) \subseteq t$. □

The second part of the condition (P4) is more delicate. It states that σ remains homomorphism after the parallel application of $a(\vec{b})$ and $a(\sigma\vec{b})$, i.e., σ maps $s' = a(\vec{b})[\![s]\!]$ to $t' = a(\sigma\vec{b})[\![t]\!]$. It is easy to see that the homomorphism is preserved on monotonic tasks, and what makes planning tasks non-monotonic are non-empty delete effects. More precisely, to preserve the homomorphism σ, we need to make sure that, if $a(\vec{b})$ preserves an atom $p(\vec{b})$ from s to s', then $a(\sigma\vec{b})$ preserves its homomorphic image $p(\sigma\vec{b})$ from t to t'.

Now we are ready to state the new condition (P4') that can replace (P4) and after that, we show how to use lifted mutex groups to efficiently evaluate this condition on the PDDL task:

(P4') For each ground action $a(\vec{b})$ and each reachable state s, if $a(\vec{b})$ is applicable in s and there exist $p(\vec{d}) \in \mathrm{del}_a(\vec{b})$ and $p(\vec{c}) \in s$ such that $\vec{c} \neq \vec{d}$, then $\sigma\vec{c} \neq \sigma\vec{d}$.

Lemma 12. *Let $\sigma\colon \mathcal{B} \to \mathcal{B}$ be a map on objects satisfying (P1–P3). If σ satisfies (P4'), then σ satisfies (P4).*

Proof. Suppose that s, t are reachable states such that $\sigma\colon s \to t$ and $a(\vec{b})$ is applicable in s. Let $s' = a(\vec{b})[\![s]\!]$. By Lemma 11 we know that $a(\sigma\vec{b})$ is applicable in t. So it remains to prove that σ is a state homomorphism from s' to $t' = a(\sigma\vec{b})[\![t]\!]$, i.e., we have to check that $p(\vec{c}) \in s'$ implies $p(\sigma\vec{c}) \in t'$.

Let $p(\vec{c})$ be a ground atom belonging to s'. We distinguish two cases. First, $p(\vec{c}) \in \mathrm{add}_a(\vec{b})$. It follows that $p(\sigma\vec{c}) \in \mathrm{add}_a(\sigma\vec{b})$. Consequently, $p(\sigma\vec{c}) \in t'$.

Second, suppose $p(\vec{c}) \notin \mathrm{add}_a(\vec{b})$. In this case, we must have $p(\vec{c}) \in s$ and $p(\vec{c}) \notin \mathrm{del}_a(\vec{b})$. As $\sigma\colon s \to t$, it follows that $p(\sigma\vec{c}) \in t$. To finish the proof, we have to show that $p(\sigma\vec{c}) \notin \mathrm{del}_a(\sigma\vec{b})$ because it implies $p(\sigma\vec{c}) \in t'$. Suppose for the sake of contradiction $p(\sigma\vec{c}) \in \mathrm{del}_a(\sigma\vec{b})$. It follows that there has to be a tuple $\vec{d}$ such that $p(\vec{d}) \in \mathrm{del}_a(\vec{b})$ and $\sigma\vec{d} = \sigma\vec{c}$. Moreover, $\vec{d} \neq \vec{c}$ because $p(\vec{c}) \notin \mathrm{del}_a(\vec{b})$. Thus by (P4') we get $\sigma\vec{d} \neq \sigma\vec{c}$ (a contradiction). □

In order to find a computable condition implying (P4'), we identify a set of objects that have to be fixed by the mapping σ in a sense that $\sigma(b) = b$ for every such object $b \in \mathcal{B}$. We fix those objects for which we cannot ensure that (P4') holds using lifted mutex groups. The reason we need lifted mutex groups is that they can tell us which atoms cannot appear together in any reachable state. Consequently, this allows us to say which objects do not need to be fixed.

Definition 13. Let $\mu(\vec{u}, \vec{v})$ be a lifted mutex group with fixed and counted variables $\vec{u}$ and $\vec{v}$, respectively. An atom $p(\vec{u}, \vec{v}) \in \mu(\vec{u}, \vec{v})$ **covers** another atom $p(\vec{x}, \vec{y})$ if there is a substitution ξ such that $\xi(\vec{u}) = \vec{x}$ and $\xi(\vec{v}) = \vec{y}$. The variables $\vec{x}$ are then called fixed and $\vec{y}$ counted.

Let $\mathcal{M}$ denote a set of lifted mutex groups. We say that $\mathcal{M}$ **covers** an atom $p(\vec{x}, \vec{y})$ if there exists a mutex group $\mu(\vec{u}, \vec{v}) \in \mathcal{M}$ and an atom $p(\vec{u}, \vec{v}) \in \mu(\vec{u}, \vec{v})$ that covers $p(\vec{x}, \vec{y})$.

Note that if $\mathcal{M}$ covers an atom $p(\vec{x})$, the covering atom from a mutex group in $\mathcal{M}$ does not need to be unique. Only when a covering atom is selected, one can split variables $\vec{x}$ into fixed ones and counted ones since different covering atoms can induce different splittings for $\vec{x}$.

Example 14. Continuing with our running example, note that all delete effects of the action schemata `drive`, `pick`, and `drop` are covered by exactly one atom from the mutex groups μ_1, μ_2, μ_3. The delete effect $\mathtt{at}(v, x)$ of the action schema `drive` is covered by the atom from μ_1 with the identity substitution. Thus v is a fixed variable and x is counted.

Consider the delete effects of the action schema `pick`, i.e., $\mathrm{del}_{\mathtt{pick}}(\vec{v}) = \{\mathtt{at}(p, x), \mathtt{cap}(v, n_2)\}$. The first delete effect is covered by the first atom in μ_3 with the identity substitution. The second one is covered by the atom from μ_2 with the substitution $v \mapsto v$ and $n \mapsto n_2$. So p, v are fixed variables and x, n_2 are counted. Covering of the delete effects of `drop` is analogous.

Observe that for a covered atom $p(\vec{x}, \vec{y})$ with fixed variables $\vec{x}$ and counted variables $\vec{y}$ it holds that whenever atoms $p(\vec{b}, \vec{c})$ and $p(\vec{b}, \vec{d})$ for $\vec{b} \in \prod_{i=1}^{|\vec{x}|} \tau_{var}(x_i)$ and $\vec{c}, \vec{d} \in \prod_{i=1}^{|\vec{y}|} \tau_{var}(y_i)$ belong to a reachable state, then $\vec{c} = \vec{d}$. In other words, we cannot have two different atoms $p(\vec{b}, \vec{c})$ and $p(\vec{b}, \vec{d})$ simultaneously in a reachable state. This property of lifted mutex groups is necessary for identifying which objects do not need to be fixed. The remaining objects specified in the following definition are fixed.

Definition 15. Given a set of lifted mutex groups $\mathcal{M}$ and an action schema $a(\vec{x}) \in \mathcal{A}$, let:

- X_a^{del} denote the set of variables appearing in all $p(\vec{x}) \in \mathrm{del}_a(\vec{x}) \setminus \mathrm{pre}_a(\vec{x})$; it contains variables from delete effects that do not have their matching atoms in the precondition. In other words, these are delete effects for which we cannot be sure that they actually delete an atom;
- $X_{a,\mathcal{M}}^{\mathrm{ncov}}$ denote the set of variables appearing in all $p(\vec{x}) \in \mathrm{del}_a(\vec{x})$ *not covered* by $\mathcal{M}$;
- $X_{a,\mathcal{M}}^{\mathrm{cov}}$ denote the set of *fixed variables* appearing in all $p(\vec{x}) \in \mathrm{del}_a(\vec{x})$ *covered* by $\mathcal{M}$. We assume here that for each such $p(\vec{x})$ a covering atom was selected so that the tuple of fixed variables occurring in $\vec{x}$ is uniquely determined. In our experiments, we implemented this selection naively without striving to find the best possible choice.

$X_{\mathcal{M}} = \bigcup_{a' \in \mathcal{A}} (X_{a'}^{\mathrm{del}} \cup X_{a',\mathcal{M}}^{\mathrm{ncov}} \cup X_{a',\mathcal{M}}^{\mathrm{cov}})$ denotes a set of **identity variables** for the set of lifted mutex groups $\mathcal{M}$, and

$\mathcal{F}_{\mathcal{M}} = \bigcup_{x \in X_{\mathcal{M}}} \tau_{var}(x)$ denotes a set of **identity objects** for the set of lifted mutex groups $\mathcal{M}$.

Fixing objects in $\mathcal{F}_{\mathcal{M}}$ helps us to ensure that the following situation cannot occur. Let s be a state containing an atom $p(\vec{c})$ and let $a(\vec{b})$ be an action applicable in s preserving $p(\vec{c})$ and deleting an atom $p(\vec{d})$ (thus $\vec{c} \neq \vec{d}$) not necessarily contained in s. If $\sigma\colon s \to t$ maps both $\vec{c}, \vec{d}$ to the same tuple $\sigma\vec{c} = \sigma\vec{d}$, then $p(\sigma\vec{c}) = p(\sigma\vec{d})$ is not preserved by $a(\sigma\vec{b})$ because $p(\sigma\vec{d})$ is a delete effect of $a(\sigma\vec{b})$. Consequently, σ would not be a state homomorphism from $a(\vec{b})[\![s]\!]$ to $a(\sigma\vec{b})[\![t]\!]$ because $p(\vec{c}) \in a(\vec{b})[\![s]\!]$ and $p(\sigma\vec{c}) \notin a(\sigma\vec{b})[\![t]\!]$.

To ensure that this situation cannot happen, we either enforce $\sigma\vec{c} \neq \sigma\vec{d}$ by fixing suitable objects or apply the mutex groups $\mathcal{M}$ to infer that $p(\vec{c})$ and $p(\vec{d})$ cannot occur simultaneously in s. To exclude the cases when the delete effect $p(\vec{d}) \notin s$, the variables X_a^{del} are included among the identity variables. Further, to apply the mutex groups, we need the delete effect $p(\vec{d})$ to be covered. Therefore the variables $X_{a,\mathcal{M}}^{\mathrm{ncov}}$ belong to the set of identity variables. Finally, $\vec{c}$ and $\vec{d}$ may differ only on counted variables for the inference via $\mathcal{M}$ to be applicable. That is why the set $X_{a,\mathcal{M}}^{\mathrm{cov}}$ is introduced.

Example 16. Consider our running example again. All its actions satisfy $\mathrm{del}_a(\vec{x}) \subseteq \mathrm{pre}_a(\vec{x})$. Hence $X_a^{\mathrm{del}} = \emptyset$ for each of them. As all delete effects are covered by mutex groups $\mathcal{M} = \{\mu_1, \mu_2, \mu_3\}$, we have $X_{a,\mathcal{M}}^{\mathrm{ncov}} = \emptyset$ as well. Finally, the delete effect $\mathtt{at}(v, x)$ of the action schema `drive` is covered by the atom from μ_1 with v fixed and x counted. Thus $X_{\mathtt{drive},\mathcal{M}}^{\mathrm{cov}} = \{v\}$. Similarly, the fixed variables occurring in the delete effects of `pick` for the covering described in Example 14 are p and v. Hence $X_{\mathtt{pick},\mathcal{M}}^{\mathrm{cov}} = \{p, v\}$. Analogously, we have $X_{\mathtt{drop},\mathcal{M}}^{\mathrm{cov}} = \{p, v\}$. Consequently, the set of identity variables $X_{\mathcal{M}} = \{p, v\}$ and $\mathcal{F}_{\mathcal{M}} = \mathtt{pkg} \cup \mathtt{veh}$. Based on the delete effects, we look for a map $\sigma\colon \mathcal{B} \to \mathcal{B}$ which behaves on $\mathtt{pkg} \cup \mathtt{veh}$ as identity. Thus we can detect redundant objects only in $\mathtt{loc} \cup \mathtt{num}$.

Before we finally get to the main result of this section stated in Theorem 19, where we show how to use lifted mutex groups to verify that a given mapping σ is, in fact, a PDDL endomorphism, we need to show that we can separate tuples of objects using minimal types.

Definition 17. Given two different objects $b, c \in \mathcal{B}$, we say that b and c are **separable by types** if there exist disjoint types $\tau, \tau' \in \mathcal{T}$ such that $b \in \tau$ and $c \in \tau'$.

Given two tuples of objects $\vec{b}, \vec{c} \in \mathcal{B}^k$ of the same size k, we say that $\vec{b}$ and $\vec{c}$ are **separable by types** if there exists $i \in \{1, \ldots, k\}$ such that b_i and c_i are separable by types.

Lemma 18. *Let $\sigma\colon \mathcal{B} \to \mathcal{B}$ be a PDDL endomorphisms and let $\vec{b}, \vec{c} \in \mathcal{B}^k$ be tuples of objects such that $\vec{b} \neq \vec{c}$. Then the following hold:*

1. *If $\vec{b}, \vec{c}$ are separable by types, then $\sigma\vec{b} \neq \sigma\vec{c}$.*
2. *If $\vec{b}, \vec{c}$ are not separable by types, then for every $i \in \{1, \ldots, k\}$ there exists a minimal type $\tau_i \in \mathcal{T}$ such that $b_i, c_i \in \tau_i$.*

Proof. 1. If $\vec{b}, \vec{c}$ are separable by types, then there is i such that b_i, c_i are separable by types, i.e., there are disjoint types τ, τ' such that $b_i \in \tau$ and $c_i \in \tau'$. Since σ preserves all types (P1), it follows that $\sigma(b_i) \neq \sigma(c_i)$ and hence $\sigma\vec{b} \neq \sigma\vec{c}$.

2. By the assumption, we have that b_i, c_i are not separable by types for every i. As minimal types cover all the objects, b_i belongs to a minimal type τ_i. Similarly, $c_i \in \tau_i'$ for a minimal type τ_i'. If $\tau_i \neq \tau_i'$ then $\tau_i \cap \tau_i' = \emptyset$ and b_i, c_i would be separable by types. Thus we must have $\tau_i = \tau_i'$. □

Now we are ready to state the main result.

Theorem 19. *Let $\mathcal{M}$ denote a set of lifted mutex groups, and let $\sigma\colon \mathcal{B} \to \mathcal{B}$ denote a map such that $\sigma(b) = b$ for all identity objects $b \in \mathcal{F}_{\mathcal{M}}$. If σ satisfies (P1–P3), then σ satisfy (P4).*

Proof. It follows from Lemma 12 that it is enough to prove that σ satisfies (P4'). Assume that $a(\vec{b})$ is applicable in a reachable state s and $p(\vec{d}) \in \mathrm{del}_a(\vec{b})$. This means that there has to be an action schema $a(\vec{x})$ such that $\vec{b} \in \prod_{i=1}^{|\vec{x}|} \tau_{var}(x_i)$. Moreover, there must be a tuple of variables $\vec{y}$ such that $p(\vec{y}) \in \mathrm{del}_a(\vec{x})$ and $\vec{d} \in \prod_{i=1}^{|\vec{y}|} \tau_{var}(y_i)$. Further, suppose that $p(\vec{c}) \in s$ for some $\vec{c}$ and $\vec{c} \neq \vec{d}$. Now we need to show that $\sigma\vec{c} \neq \sigma\vec{d}$.

By Lemma 18, if $\vec{c}, \vec{d}$ are separable by types, then $\sigma\vec{c} \neq \sigma\vec{d}$ and we are done. Next suppose that $\vec{c}, \vec{d}$ are not separable by types. By Lemma 18, for all $i \in \{1, \ldots, |\vec{y}|\}$, c_i, d_i belong to the same minimal type. Thus $\vec{c} \in \prod_{i=1}^{|\vec{y}|} \tau_{var}(y_i)$. There are several cases. If $p(\vec{y}) \notin \mathrm{pre}_a(\vec{x})$ then all variables in $\vec{y}$ belongs to $X_a^{\mathrm{del}} \subseteq X_{\mathcal{M}}$. Similarly, if $p(\vec{y})$ is not covered by $\mathcal{M}$ then all variables from $\vec{y}$ belong to $X_{a,\mathcal{M}}^{\mathrm{ncov}} \subseteq X_{\mathcal{M}}$. Consequently, we have $\sigma\vec{d} = \vec{d} \neq \vec{c} = \sigma\vec{c}$. Thus we can suppose now that $p(\vec{y})$ is covered by $\mathcal{M}$ and $p(\vec{y}) \in \mathrm{pre}_a(\vec{x})$. Hence the variables $\vec{y}$ can be split into fixed variables and counted variables.

Since $\vec{c} \neq \vec{d}$, there must be i such that $c_i \neq d_i$. If at least one such i corresponds to a fixed variable $y_i \in X_{\mathcal{M}}$ then $\sigma(c_i) = c_i \neq d_i = \sigma(d_i)$. Thus $\sigma\vec{c} \neq \sigma\vec{d}$.

Finally assume that $\vec{c}$ and $\vec{d}$ differ only in components corresponding to counted variables. Since $p(\vec{y}) \in \mathrm{pre}_a(\vec{x})$, we have $p(\vec{d}) \in \mathrm{pre}_a(\vec{b}) \subseteq s$. Thus both $p(\vec{c})$ and $p(\vec{d})$ belong to s. However, this contradicts the fact that $p(\vec{y})$ is covered by an atom from a lifted mutex group. □

Example 20. Using Theorem 19, we can finish our running example. We know from Example 16 that our endomorphism $\sigma\colon \mathcal{B} \to \mathcal{B}$ has to fix all the objects in $\mathtt{pkg} \cup \mathtt{veh}$. Since σ has to preserve types, we have $\sigma(\mathtt{loc}) \subseteq \mathtt{loc}$ and analogously for `num`. There is a non-identity endomorphism on the graph represented by the relation `road` which maps c_1 to c_2. On the other hand, there is no non-identity endomorphism on the three-element chain represented by the interpretation of `pred`.[1] Altogether, consider a map $\sigma\colon \mathcal{B} \to \mathcal{B}$

[1] Note that if we change, for instance, the interpretation of `pred` to $\{\langle 0, 1\rangle, \langle 0, 2\rangle\}$, then there would be an endomorphism mapping 1 to 2 and fixing $0, 2$.

such that $\sigma(c_1) = c_2$ and $\sigma(x) = x$ for $x \neq c_1$. This map clearly satisfies (P1). It satisfies (P2) because σ preserves the interpretation of `road`. Further, it satisfies (P3) as c_1 do not occur in ψ_G. Finally, it fixes elements from $\mathcal{F}_\mathcal{M}$. Hence it is a PDDL endomorphism by Theorem 19. So, c_1 is a redundant object, and can be safely removed from the planning task.

CSP Formulation

In this section, we will formulate a constraint satisfaction problem whose solutions will be endomorphisms of PDDL tasks. First, we recall the **Constraint Satisfaction Problem** (CSP) (for details see, e.g., Russell and Norvig 2010, Chapter 6). Suppose we are given a set of variables $\mathcal{X}$, a finite domain $\mathcal{D}$ where the variables can take values, and a set of constraints $\mathcal{C}$. The problem is whether there exists an assignment $h\colon \mathcal{X} \to \mathcal{D}$ of values $\mathcal{D}$ to variables $\mathcal{X}$ that satisfies all the constraints. Each constraint $C \in \mathcal{C}$ is specified as a pair $C = \langle \vec{X}, R\rangle$ where $\vec{X}$ is a tuple of variables from $\mathcal{X}$ of length n, and R is an n-ary relation on $\mathcal{D}$. The assignment h satisfies the constraint $C = \langle \vec{X}, R\rangle$ if $h(\vec{X}) \in R$. CSPs are often formulated in such a way that each variable $X \in \mathcal{X}$ has its domain of allowed values $\mathcal{D}_X \subseteq \mathcal{D}$. Note that each such restriction can be equivalently expressed as a unary constraint $\langle X, \mathcal{D}_X\rangle$.

Let $\mathcal{M}$ denote an inferred set of lifted mutex groups, $X_\mathcal{M}$ the corresponding set of identity variables, and $\mathcal{F}_\mathcal{M}$ the set of identity objects. For each object $b \in \mathcal{B}$ introduce a variable X_b. The domain of X_b is $\mathcal{B}$. Given a tuple $\vec{b}$ of objects, we denote the tuple of corresponding variables $\vec{X}_{\vec{b}}$. Further, we introduce the following constraints corresponding to Theorem 19:

(C1) for each type $\tau \in \mathcal{T}$ and each object $b \in \tau$ the constraint $C_{b,\tau} = \langle X_b, \tau\rangle$,

(C2) for each grounded atom $p(\vec{b})$ in ψ_I the constraint $C_{p(\vec{b})} = \langle \vec{X}_{\vec{b}}, \{\vec{b}' \mid p(\vec{b}') \in \psi_I\}\rangle$,

(C3) for each $b \in \mathcal{B}$ occurring in ψ_G the constraint $G_b = \langle X_b, \{b\}\rangle$,

(C4) for each $b \in \mathcal{F}_\mathcal{M}$ the constraint $F_b = \langle X_b, \{b\}\rangle$,

A solution to the above CSP is a mapping of variables to values in $\mathcal{B}$ satisfying all the constraints (C1–C4). As there is a 1-1 correspondence between the CSP variables and objects, the solution is, in fact, a self-map σ on $\mathcal{B}$. Moreover, the constraints (C1–C3) ensure that (P1–P3) holds. Finally, $\sigma(b) = b$ for any $b \in \mathcal{F}_\mathcal{M}$ is ensured by (C4).

For the optimal planning, we further need constraints for the costs of actions forcing σ to satisfy (P5). Consider an action schema $a(\vec{x})$. In our experiments on the IPC domains, we had to deal with two cases. First, the cost of all instances of $a(\vec{x})$ have the same cost, i.e., $\mathrm{c}_a(\vec{b}) = \mathrm{c}_a(\sigma\vec{b})$ holds for any $\sigma\colon \mathcal{B} \to \mathcal{B}$. Thus there is no need to introduce a constraint in this case. Second, the actual cost of a ground action $a(\vec{b})$ is encoded in the initial state as a "valued" static k-ary predicate, i.e., a set of k-ary ground atoms together with their costs is provided in the initial state. For instance, there is a valued binary predicate $\mathrm{road\text{-}length}$ in the transportation domain encoding a distance between connected locations. Let ψ_I^{cost} be the set of ground atoms of valued predicates in the initial state. We call the elements of ψ_I^{cost} "valued" ground atoms. For two valued ground atoms $q(\vec{b}), q(\vec{c}) \in \psi_I^{\mathrm{cost}}$ with the same predicate symbol q we write $q(\vec{b}) \leq q(\vec{c})$ if the cost associated with $q(\vec{b})$ is less than or equal to the cost of $q(\vec{c})$. In order to satisfy (P5), we have to preserve the valued predicates in the initial state which can be done by the following constraints:

(C5) for each valued ground atom $q(\vec{b}) \in \psi_I^{\mathrm{cost}}$ we introduce the constraint

$$C_{q(\vec{b})} = \langle \vec{X}_{\vec{b}}, \{\vec{b}' \mid q(\vec{b}') \leq q(\vec{b}),\, q(\vec{b}') \in \psi_I^{\mathrm{cost}}\}\rangle.$$

Related Work

PDDL endomorphisms naturally induce endomorphisms of facts and operators on the ground level through the grounding process. For example in our running example, a PDDL edomorphism mapping the location c_1 to c_2 induces the ground endomorphism that maps the fact $\mathtt{at}(p_0, c_1)$ to $\mathtt{at}(p_0, c_2)$, the operator $\mathtt{drive}(a, c_1, c_0)$ to $\mathtt{drive}(a, c_2, c_0)$, and so on for all other facts and operators where c_1 and c_2 appear. In our previous work (Horčík and Fišer 2021), we introduced endomorphisms on the ground level (STRIPS and FDR) and discussed the relation to structural symmetries (e.g., Pochter, Zohar, and Rosenschein 2011; Shleyfman et al. 2015), h^m heuristics (Haslum and Geffner 2000; Alcázar and Torralba 2015), operator mutexes (Fišer, Torralba, and Shleyfman 2019), and dominance pruning (Torralba and Hoffmann 2015) there.

On the lifted level, there is a connection between PDDL endomorphisms and structural symmetries (Röger, Sievers, and Katz 2018; Sievers et al. 2019). In fact, any bijective PDDL endomorphism (i.e., a PDDL automorphism) is a structural symmetry. Nevertheless, the structural symmetries as defined by Röger, Sievers, and Katz (2018) and Sievers et al. (2019) are more general than PDDL automorphisms as we define them, because they allow to permute not only objects, but also predicate symbols and action schemata. On the other hand, unlike structural symmetries, PDDL endomorphisms allow us to find redundant objects, because they are not restricted to permutations of objects.

Experiments

The inference of PDDL endomorphisms and pruning of redundant objects was implemented[2] in C and experimentally evaluated on a cluster of computing nodes with Intel Xeon Scalable Gold 6146 processors and 8 GB memory limit per task. We used all planning domains from International Planning Competitions (IPCs) from 1998 to 2018 that are typed (54 domains in the optimal track, and 51 domains in the satisficing track). For the domains from the optimal track, we computed PDDL endomorphisms preserving optimal plans, i.e., costs of actions were considered, and for the satisficing track, we disregarded costs.

[2]Repository https://gitlab.com/danfis/cpddl, branch icaps21-lifted-endomorphism

domain	#ps	%obj	%op	%fact
agricola18 (20)	20	1.15	0.00	0.00
airport04 (50)	50	4.00	0.00	0.00
caldera18 (20)	20	15.15	0.00	0.00
citycar14 (20)	5	7.76	0.00	0.00
data-network18 (20)	10	4.12	0.00	0.00
parcprinter08 (30)	10	7.05	0.00	0.00
parcprinter11 (20)	6	7.28	0.00	0.00
pipesworld-notankage04 (50)	5	6.58	0.00	0.00
pipesworld-tankage04 (50)	11	2.88	0.00	0.00
woodworking08 (30)	1	3.23	0.00	0.00
childsnack14 (20)	2	2.91	2.16	2.78
rovers06 (40)	33	3.84	11.32	6.43
satellite02 (20)	17	7.61	20.40	11.48
tpp06 (30)	1	2.22	13.44	0.42
transport08 (30)	13	3.51	6.22	5.31
transport11 (20)	11	4.65	9.28	7.63
transport14 (20)	15	5.44	8.59	7.44
visitall11 (20)	8	15.57	21.28	16.71
visitall14 (20)	6	11.38	13.33	11.51
overall from above (530)	244	5.83	5.33	3.61
overall from op-pruned (220)	106	5.99	12.26	8.31

Table 1: Pruning on the domains from the optimal track. #ps: the number of tasks with detected non-trivial PDDL endomorphisms; %obj: an average percentage of removed objects in tasks with non-trivial PDDL endomorphisms; %op: an average percentage of removed operators after grounding; %fact: an average percentage of removed facts after grounding; overall from above: sum for #ps and averages over all tasks above for the rest; overall from op-pruned: considers only tasks where at least one operator was pruned.

domain	#ps	%obj	%op	%fact
agricola18 (20)	20	1.01	0.00	0.00
airport04 (50)	50	4.00	0.00	0.00
caldera18 (18)	18	12.72	0.00	0.00
citycar14 (20)	2	3.52	0.00	0.00
data-network18 (20)	11	3.08	0.00	0.00
parcprinter11 (20)	4	5.72	0.00	0.00
pipesworld-notankage04 (50)	5	6.58	0.00	0.00
pipesworld-tankage04 (50)	11	2.88	0.00	0.00
woodworking08 (30)	1	11.54	0.00	0.00
woodworking11 (20)	5	10.42	0.00	0.00
rovers06 (40)	33	3.84	11.32	6.43
satellite02 (20)	17	7.61	20.40	11.48
tpp06 (30)	1	2.22	13.44	0.42
transport08 (30)	25	6.56	11.05	9.59
transport11 (20)	19	6.51	8.77	8.32
transport14 (20)	18	5.47	6.75	6.57
overall from above (458)	240	5.36	5.41	3.85
overall from op-pruned (160)	113	5.70	11.49	8.17

Table 2: As Table 1, but for the satisficing track.

For the inference of lifted mutex groups we used the algorithm proposed by Fišer (2020). We removed all atoms containing only fixed variables from all lifted mutex groups. Consequently, we used all remaining non-empty mutex groups for covering atoms in delete effects. We can preprocess lifted mutex groups in this way, because every subset of a mutex group is also a mutex group and removing atoms from lifted mutex groups clearly generate subsets.

For solving CSPs, we used CP Optimizer from IBM ILOG CPLEX Optimization Studio v12.9. This solver contains an objective function that returns the number of different values assigned to a specified set of variables. We use the minimization of such objective function over object variables to obtain optimal solutions.

The CSP formulation was solved optimally for all tested tasks. The computation took less than 50 seconds for each task, but less than 1 second in all domains except for nomystery11 (the median over all tasks was 5 milliseconds, and the average was 126 milliseconds). We were able to find non-trivial PDDL endomorphisms in 19 domains from the optimal track, and in 16 domains from the satisficing track.

Table 1 and Table 2 report average percentage of pruned objects for optimal and satisficing track, respectively. The tables also show an average percentage of pruned STRIPS operators and facts in a sense that reducing the number of objects in the PDDL representation led to a smaller number of operators and facts after grounding using relaxed reachability (Helmert 2009).

The tables are split so that the top part lists domains where objects were pruned, but the pruning did not reflect on the number of operators or facts after grounding. In the domains agricola18, airport04, caldera18, and parcprinter08/11, we found constant objects that are not used at all in the corresponding tasks. In the rest of the domains, the pruned objects are used in the initial state, but the relaxed reachability is already able to prune all other atoms containing the same object. Although this type of pruning is not particularly useful for the planners using ground representations, we think it could, at some point, be useful in lifted planning (e.g., Ridder 2014; Corrêa et al. 2020), or it can help in a process of designing models for real-world problems.

The bottom parts of the tables show domains where pruning of objects led to pruning of operators and facts, i.e., the PDDL endomorphism proved to be stronger than simple relaxed reachability. In childsnack14 that models making and serving sandwiches, the endomorphism recognized that one of the serving tables is not necessary for solving the task. In satellite02 that models collecting images with satellites, we found instruments used by satellites and directions where satellites should be pointed to that were not necessary. In the rest of the domains, the endomorphisms were able to prune unnecessary parts of maps of locations defined using static predicates.

Furthermore, we tested whether the pruning of operators and facts positively impacts the performance of ground planners running with 8 GB memory limit and 30 minutes time limit. For optimal planning, we used the heuristic search planner Fast Downward (Helmert 2006) with A* and the LM-Cut (`lmc`) heuristic (Helmert and Domshlak 2009); the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy and non-greedy bisimulation shrink strategy (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2016); the potential (`pot`) heuristic enhanced with disambiguations and optimized for all syntactic states with added constraint for the initial state (Seipp, Pommerening, and Helmert 2015; Fišer, Horčík, and Komenda 2020); and the Scorpion planner (Seipp 2018; Seipp and Helmert 2018) that performed

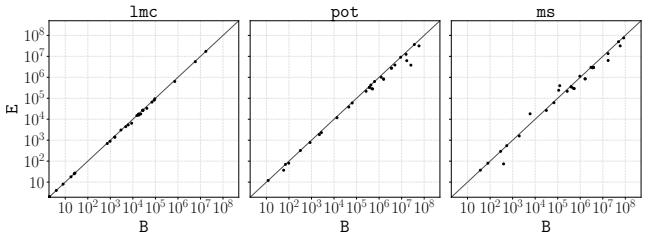

Figure 4: Number of expansions before the last f layer for tasks solved both with (E) and without (B) pruning using PDDL endomorphisms where at least one operator was pruned.

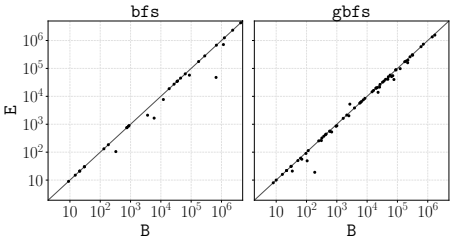

Figure 5: As in Figure 4 but for the lifted planner.

very well in the last IPC 2018; and the symbolic planner SymbA* (Torralba et al. 2017) used as a baseline in IPC 2018. All of these planners were run without the h^2 pruning (Alcázar and Torralba 2015) as a preprocessing step.

For satisficing planner, we used LAMA planner (Richter and Westphal 2010) used as a baseline planner for IPC 2018, and Mercury planner (Domshlak, Hoffmann, and Katz 2015) employing red-black planning techniques.

Unfortunately, the number of solved tasks remained the same except for two more tasks solved by Scorpion in satellite02, and one more task solved by SymbA* in transport11. Regarding the cost of the first found plans for the satisficing planners in the pruned domains, LAMA found cheaper plans in 28 tasks, but more expensive in 17 tasks in the transport08/11/14 domains. Mercury found 22 cheaper, and 18 more expensive plans also in transport08/11/14. The number of expanded states also is not significantly affected by the pruning as can be seen in Figure 4. So, it seems that the pruning with PDDL endomorphisms is not particularly beneficial for grounded planning techniques on IPC domains.

To test the impact of pruning on lifted planning, we evaluated the lifted planner introduced by Corrêa et al. (2020) also with 8 GB memory limit and 30 minutes time limit. We tested two variants of a search: a simple breath-first search (denoted by `bfs`) and a greedy best-first search with goal-count heuristic (Fikes and Nilsson 1971) (denoted by `gbfs`). We did not modify the planner, but instead the tasks were pruned first and new reduced domain and problem PDDL files were generated as inputs for the planner.

Since we were not able to find any non-trivial PDDL endomorphisms in the *hard-to-ground* domains used by Corrêa et al.,[3] we used the domains from the satisficing track of IPC where we found some redundant objects. As for the ground planners, the resulting coverage was almost identical. The `bfs` variant with pruning solved one more task in pipesworld-tankage04, and `gbfs` with pruning solved one more task in sattelite02 and one more in transport11. The number of expanded states (Figure 5) was reduced in only a small fraction of tasks. However, we think this picture can change in the future when we will have available a wider range of hard-to-ground domains.

[3]Organic synthesis domains designed by Russell Viirre and coverted into PDDL by Hadi Qovaizi (https://www.cs.ryerson.ca/~mes/publications); large Pipesworld-tankage domain based on the one introduced in IPC 2004 (Hoffmann et al. 2006); and Genome edit distance domains introduced by Haslum (2011).

Conclusion

In this paper, we started a formal study of plans preserving maps between PDDL tasks. We introduced a notion of a PDDL endomorphism as a self-map on the set of objects which preserves (optimal) plans. We applied this notion and developed a method allowing to prune redundant objects from PDDL tasks. We experimentally tested this method on the tasks from the IPC domains. It turned out that our method is able to detect a moderate amount of redundant objects in roughly one-third of the considered domains. However, the pruning of redundant objects did not significantly affect the performance of planners working on the propositional level.

The main goal of this work was to introduce a novel tool for analyzing planning tasks on the lifted level. We believe that the theory behind PDDL endomorphisms can encourage further research in lifted planning. For example, lifted mutex groups are usually inferred to be grounded and used for the translation to finite domain representation (e.g., Helmert 2009; Fišer 2020). Here, we utilized lifted mutex groups directly on the lifted level, and it is clear that having available a richer mutex structure could only improve our results. Moreover, there are also other types of lifted invariants (e.g., Rintanen 2017) that should be investigated in connection to PDDL endomorphisms, because they could provide a different type of useful information about reachable states.

Another line of work that could spawn from this work is a generalization to PDDL homomorphisms. Although we investigated only endomorphisms, the definition can be easily modified so that we obtain PDDL homomorphisms, i.e., maps between PDDL tasks preserving plans. We might for instance define abstractions of PDDL tasks which are again PDDL tasks. This could possibly lead to a strong admissible heuristic for lifted planning, as abstraction heuristics currently belong to the strongest heuristics for the grounded planning.

Acknowledgements

Rostislav Horčík was supported by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 “Research Center for Informatics”. Daniel Fišer was supported by the Czech Science Foundation (GAČR) under the project No. 19-22555Y.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proc. ICAPS'15*, 2–6.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence* 11(4): 625–655.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proc. ICAPS'20*, 80–89.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-Black Planning: A New Systematic Approach to Partial Delete Relaxation. *Artificial Intelligence* 221: 73–114.

Fikes, R. E.; and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2: 189–208.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In *Proc. AAAI'20*, 9835–9842.

Fišer, D.; Horčík, R.; and Komenda, A. 2020. Strengthening Potential Heuristics with Mutexes and Disambiguations. In *Proc. ICAPS'20*, 124–133.

Fišer, D.; Torralba, Á.; and Shleyfman, A. 2019. Operator Mutexes and Symmetries for Simplifying Planning Tasks. In *Proc. AAAI'19*, 7586–7593.

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 *Proc. AAAI'19*, 7602–7609.

Haslum, P. 2011. Computing Genome Edit Distances using Domain-Independent Planning. In *Proceedings of the 5th International Scheduling and Planning Applications woRKshop (SPARK)*.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proc. 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.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS'09*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the Association for Computing Machinery* 61(3): 16.1–16.63.

Hodges, W. 1997. *A Shorter Model Theory*. Cambridge University Press. ISBN 978-0-521-58713-6.

Hoffmann, J.; Edelkamp, S.; Thíebaux, S.; Englert, R.; Liporace, F.; and Trüg, S. 2006. Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4. *Journal of Artificial Intelligence Research* 26: 453–541.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14: 253–302.

Horčík, R.; and Fišer, D. 2021. Endomorphisms of Classical Planning Tasks. In *Proc. AAAI'21*, To appear.

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

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In *Proc. AAAI'11*, 1004–1009.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research* 39: 127–177.

Ridder, B. 2014. *Lifted heuristics : towards more scalable planning systems*. Ph.D. thesis, King's College London, UK.

Rintanen, J. 2017. Schematic Invariants by Reduction to Ground Invariants. In *Proc. AAAI'17*, 3644–3650.

Röger, G.; Sievers, S.; and Katz, M. 2018. Symmetry-Based Task Reduction for Relaxed Reachability Analysis. In *Proc. ICAPS'18*, 208–217.

Russell, S.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach (Third Edition)*. Englewood Cliffs, NJ: Prentice-Hall. ISBN 0-13-103805-3.

Seipp, J. 2018. Fast Downward Scorpion. In *IPC 2018 planner abstracts*, 77–79.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research* 62: 535–577.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In *Proc. ICAPS'15*, 193–201.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In *Proc. AAAI'15*, 3371–3377.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2019. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In *Proc. ICAPS'19*, 446–454.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An Analysis of Merge Strategies for Merge-and-Shrink Heuristics. In *Proc. ICAPS'16*, 294–298.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence* 242: 52–79.

Torralba, Á.; and Hoffmann, J. 2015. Simulation-Based Admissible Dominance Pruning. In *Proc. IJCAI'15*, 1689–1695.