Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains

Songtuan Lin, Alban Grastien, Pascal Bercher

School of Computing, The Australian National University
songtuan.lin@anu.edu.au, alban.grastien@anu.edu.au, pascal.bercher@anu.edu.au

Abstract

Designing a planning domain is a difficult task in AI planning. Assisting tools are thus required if we want planning to be used more broadly. In this paper, we are interested in automatically correcting a flawed domain. In particular, we are concerned with the scenario where a domain contradicts a plan that is known to be valid. Our goal is to repair the domain so as to turn the plan into a solution. Specifically, we consider both grounded and lifted representations support for negative preconditions and show how to explore the space of repairs to find the optimal one efficiently. As an evidence of the efficiency of our approach, the experiment results show that all flawed domains except one in the benchmark set can be repaired optimally by our approach within one second.

Introduction

One major obstacle for planning techniques being used more broadly, particularly outside academia, is the fact that modeling a planning domain is a non-trivial task (McCluskey, Vaquero, and Vallati 2017). As an evidence of this, the competition on modeling domains, the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS), has been held for many years.

Tools are thus developed for modeling assistance, e.g., the successful online PDDL editor Planning.Domains (Muise 2016), itSIMPLE (Vaquero et al. 2012), MyPDDL (Strobel and Kirsch 2020), and the Visual Studio Code plugin for PDDL (see the work by Haslum et al. (2019) for more details about PDDL). Many of those tools provide a rich range of features for assisting writing domain models, e.g., syntax highlighting and auto-completion, whereas only few of they provide advanced support. We will have a detailed review of related works at the end of this paper.

As another contribution to modeling assistance, we revisit the scenario studied in our previous work (Lin and Bercher 2021) of correcting flawed domains by giving plans known to be valid. We studied the scenario in the context of both hierarchical and non-hierarchical planning (Bercher, Alford, and Höller 2019; Ghallab, Nau, and Traverso 2004). For the latter, we investigated the complexity of checking whether a certain number of changes to actions (alter their preconditions and effects) is able to turn the given plan into a solution (and thus repair the domain). However, we did not provide any insight into how this can be done in practice there, and neither negative preconditions nor results for lifted models were considered, both of which are essential for modeling assistance in practice.

In this paper, we present an approach for repairing flawed classical planning domains. Concretely, given a (flawed) domain and a plan that is known to be valid but contradict the domain, our approach finds a *cardinality-minimal* set of repairs to the domain that turns the plan into a solution. This scenario stems from how programmers debug their code, namely by providing test cases that shall be passed. In our context, the given plan serves as a test case, which is usually known by a domain modeler in advance (or can be easily obtained/designed), and the proposed repairs are potential corrections to the domain. The domain modeler can judge the repairs and rule out implausible ones. This process can be done iteratively until all satisfactory repairs are found. We thus believe that addressing this scenario will significantly benefit domain modelers in debugging their models. In the long run, we hope that we can integrate our approach into the editor Planning.Domain to make planning more accessible.

Our approach is based on finding *hitting sets*. This aligns with our earlier theoretical results(Lin and Bercher 2021) that the problem is $\mathbf{NP}$-complete, as finding a (minimal) hitting set is also $\mathbf{NP}$-complete. Further, we only proved those results for the restricted case of being grounded and not having negative preconditions. Here, we consider repairing not only grounded domains, but lifted ones with negative preconditions, making the approach more practical.

Preliminaries

We start by reviewing the fundamental mathematical notion, called the *set theoretical duality* explored by Slaney (2014), which we will exploit to solve the domain repair problem.

Let Σ be a universal set and Δ a set of sets such that each set $\delta \in \Delta$ satisfies a certain *property* ρ and $\delta \in 2^{\Sigma}$. The *dual set* Θ of Δ is defined as the set of all sets θ such that the complement set of θ is *not* in Δ (i.e., $\Sigma \backslash \theta \notin \Delta$), namely, the complement set of θ does not satisfy the property ρ.

A set (of sets) Δ together with its dual set Θ has many nice properties. Among these properties, the most important one which we are interested in is that if Δ is *monotonic* (i.e., if $\delta \in \Delta$, then any $\delta' \supseteq \delta$ is also in Δ), every set $\delta \in \Delta$ is a

Algorithm 1 Finding a cardinality-minimal set δ which satisfies a property ρ.

Input: A universal set Σ and a property ρ
Output: A cardinality-minimal set δ satisfying ρ

▷ Collection of known elements from the dual set Θ
$\Theta^* \leftarrow \emptyset$
loop
  $\delta \leftarrow$ a minimal hitting set of Θ^*
  if δ satisfies ρ **then**
    return δ
  $\theta \leftarrow \theta \in \Theta$ with $\theta \cap \delta = \emptyset$
  $\Theta^* \leftarrow \Theta^* \cup \{\theta\}$

hitting set of Θ, i.e., for every $\theta \in \Theta$, $\delta \cap \theta \neq \emptyset$. Intuitively speaking, this property holds because when Δ is monotonic, every $\theta \in \Theta$ can be interpreted as a set from which at least one element must be selected in order to satisfy the property ρ, and hence, a set δ satisfying ρ hits at least one element in θ. For more details, we refer to the work by Slaney (2014).

The fact that every set in Δ is a hitting set of Θ leads to an efficient algorithm for finding a *cardinality-minimal* set δ among Δ that satisfies the property ρ, as shown by Alg. 1 (Slaney 2014). The algorithm maintains a collection Θ^* of known elements from the dual set Θ. On each iteration, it computes a cardinality-minimal hitting set δ of Θ^*. If δ satisfies ρ, the algorithm returns it. Otherwise, a new element θ in Θ with $\delta \cap \theta = \emptyset$ is extracted and added to Θ^*.

Alg. 1 is sound because Θ^* is a subset of the dual set Θ, and thus, a minimal hitting set (MHS) of Θ^* has a cardinality *not* larger than a MHS of Θ, which is a cardinality-minimal set $\delta \in \Delta$. One remark is that Alg. 1 does not require knowing the entire sets Δ and Θ in advance.

Clearly, Alg. 1 relies on two operators. The first one is to find a minimal hitting set, which is already well-developed. The second operator is to decide whether a candidate δ satisfies the property ρ and to extract a new element from the dual set. This is problem-dependent. We call this operator an oracle, and this paper will mainly focus on how to implement such an oracle for the domain repair problem.

Duality is exploited in many disciplines, e.g., in optimal delete-free planning (Haslum, Slaney, and Thiébaux 2012; Slaney 2014), and in *model-based diagnosis* (Reiter 1987; de Kleer and Williams 1987; de Kleer, Mackworth, and Reiter 1992) which is to find a minimal set of faulty *components* in a system. Diagnosis is interested in the set Δ of all component sets that are faulty, and its dual set Θ is the set of all component sets θ such that at least one component in θ is faulty. A component set $\delta \in \Delta$ is called a *diagnosis*, and a $\theta \in \Theta$ is called a *conflict*. The domain repair problem is closely related to the diagnosis problem because every repair applied to a domain can be viewed as a flaw to be fixed.

Repairing Grounded Domains

Now we turn to show how to repair a flawed domain. In this section, we consider repairing a *grounded* domain restricted to *positive* preconditions, which is the simplest case. In the following sections, we will generalize the approach to *lifted* domains and lifted domains with negative preconditions.

We start with a brief introduction to the grounded classical planning formalism, and afterwards, we will formulate the domain repair problem in terms of it.

Grounded Planning Formalism

The grounded planning formalism is defined in a *propositional* way (i.e., without variables). A grounded planning problem Π is a tuple $(\mathcal{P}, \mathcal{A}, \alpha, s^I, g)$ where $\mathcal{P}$ is a set of propositions, also called atoms, $\mathcal{A}$ is a set of *actions*, $\alpha : \mathcal{A} \to 2^{\mathcal{P}} \times 2^{\mathcal{P}} \times 2^{\mathcal{P}}$ is a function, and $s_I \in 2^{\mathcal{P}}$ and $g \subseteq \mathcal{P}$ are called the initial state and the goal description of Π, respectively. In particular, $\mathcal{D} = (\mathcal{P}, \mathcal{A}, \alpha)$ and $\mathcal{T} = (s^I, g)$ are respectively the *domain* and *task* of Π, and thus Π is also written as $(\mathcal{D}, \mathcal{T})$. In this section, all planning problems are referred to as grounded ones, unless otherwise specified.

In the formalism, an action a is mapped to the respective *precondition* $prec(a) \subseteq \mathcal{P}$, *positive effects* $\mathit{eff}^+(a) \subseteq \mathcal{P}$, and *negative effects* $\mathit{eff}^-(a) \subseteq \mathcal{P}$ by the function α, written $\alpha(a) = (prec(a), \mathit{eff}^+(a), \mathit{eff}^-(a))$. Applying an action a in some *state* $s \in 2^{\mathcal{P}}$, which is a set of propositions, will lead to a new state s' such that $s' = (s \backslash \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$, written $s \to_a s'$. An action a is *applicable* in some state s *iff* $prec(a) \subseteq s$, i.e., its precondition is satisfied in s. Given a sequence of actions $\pi = \langle a_1 \cdots a_n \rangle$, we write $s \to_\pi^* s'$ to indicate that the state s' is obtained by applying the action sequence π in s, that is, there exists a state sequence $\langle s_0 \cdots s_n \rangle$ such that $s_0 = s$, $s_n = s'$, and for each $1 \leq i \leq n$, a_i is applicable in s_{i-1}, and $s_{i-1} \to_{a_i} s_i$. We say that $\langle s_0 \cdots s_n \rangle$ is the state trajectory obtained by applying π in s.

An action sequence $\pi = \langle a_1 \cdots a_n \rangle$ is a solution to a planning problem Π if $s^I \to_\pi^* s$ for some $s \in 2^{\mathcal{P}}$ and $g \subseteq s$. In other words, the criteria demand that every action in π is *applicable*, and applying π in s^I leads to a state in which every proposition in g holds.

Domain Repair Problem

Next we formulate the domain repair problem in the context of the grounded classical planning formalism. The problem was first introduced by Lin and Bercher (2021) and proved to be **NP**-complete. Here, we refine the formulation so that we can explore the duality inside the problem more easily.

Recall that the basic configuration of the problem is that we are given a planning problem Π and a plan π which is *not* a solution to Π. We want to repair (i.e., change) the domain of Π so that π will be a solution.

For the purpose of formulating the problem precisely, we first define repairs that are allowed to be used in correcting a planning domain. Specifically, repairs we are concerned with are restricted to removing propositions from actions' preconditions, adding propositions to actions' positive effects, and removing propositions from actions' negative effects. The reason for making such a restriction is that other changes (e.g., adding propositions to actions' preconditions) only increase the chance of a plan not being executable. In later sections, we will consider more repairs when *negative preconditions* are taken into account.

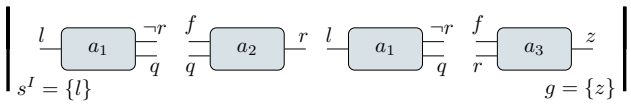

Figure 1: An example of the domain repair problem. One solution to the problem is the repair set $\{\langle F_{a_1}|_r^- \rangle, \langle F_{a_1}|_f^+ \rangle\}$, which is a cardinality-minimal one.

Definition 1. Let a be an action. The set F_a of all atomic repairs targeted at a is $F_a = F_a^p \cup F_a^+ \cup F_a^-$ in which

- $F_a^p = \{\langle F_a|_f^p \rangle \mid f \in \mathit{prec}(a)\}$
- $F_a^+ = \{\langle F_a|_f^+ \rangle \mid f \in \mathcal{P}\}$
- $F_a^- = \{\langle F_a|_f^- \rangle \mid f \in \mathit{eff}^-(a)\}$

Consequently, given a planning problem Π, the set F_Π of all atomic repairs for Π is $\bigcup_{a \in \mathcal{A}} F_a$.

Intuitively speaking, the semantics of $\langle F_a|_f^p \rangle$ is removing the proposition f from the precondition of a, and similarly, $\langle F_a|_f^+ \rangle$ and $\langle F_a|_f^- \rangle$ respectively adds f to the positive effects of a and removes f from the negative effects of a. Note that we allow adding a proposition to the positive effects of an action which has the same proposition in its negative effects. We formally define the semantics of these repairs in terms of the consequence of applying them to a planning domain.

Definition 2. Let $\Pi = (\mathcal{P}, \mathcal{A}, \alpha, s^I, g)$ be a planning problem and κ a subset of F_Π. Applying κ to Π leads to a new planning problem $\Pi' = (\mathcal{P}, \mathcal{A}, \alpha', s^I, g)$, such that for every action $a \in \mathcal{A}$ with $\alpha(a) = (\mathit{prec}(a), \mathit{eff}^+(a), \mathit{eff}^-(a))$, $\alpha'(a) = (\mathit{prec}(a) \backslash P, \mathit{eff}^+(a) \cup E^+, \mathit{eff}^-(a) \backslash E^-)$ where

- $P = \{f \mid \langle F_a|_f^p \rangle \in \kappa\}$
- $E^+ = \{f \mid \langle F_a|_f^+ \rangle \in \kappa\}$
- $E^- = \{f \mid \langle F_a|_f^- \rangle \in \kappa\}$

We write $\Pi \Rightarrow_\kappa \Pi'$ to indicate that Π' is obtained by applying a set κ of repairs to Π.

Having defined the set of all atomic repairs for a planning problem together with the respective semantics, we now formulate the problem of finding a set of repairs that turns a non-solution plan into a solution.

Definition 3. Given a planning problem Π and an action sequence $\pi = \langle a_1 \cdots a_n \rangle$ with $a_i \in \mathcal{A}$ for each $1 \leq i \leq n$, a domain repair problem is the tuple (Π, π) that is to find a cardinality-minimal subset $\delta^* \subseteq F_\Pi$ such that $\Pi \Rightarrow_{\delta^*} \Pi^*$ for some Π^*, and π is a solution to Π^*.

Notice that here we demand that a repair set found for the problem must have the *minimal cardinality*, because otherwise, we can *always* find the repair set which empties the precondition of *every* action in the plan and adds all propositions in the goal description to the last action in the plan.

For clarity, throughout the paper, we will let δ be *any* repair set that turns π into a solution, δ^* a cardinality-minimal one that does so, and $\kappa \subseteq F_\Pi$ an *arbitrary* repair set.

As mentioned earlier, we could also interpret each repair in F_Π as a flaw. For instance, $\langle F_a|_f^+ \rangle$ can be interpreted as the flaw that the positive effects of a lack the proposition f. We could thus regard a domain repair problem as a diagnosis problem by viewing each repair as a component which is faulty if the respective flaw exists, and the goal of the domain repair problem is to find a cardinality-minimal set of faulty components which must be repaired to turn π into a solution. We will bear this connection in the paper and call each repair set that turns π into a solution a *diagnosis* (Reiter 1987).

Algorithm 2 Computing a conflict for a grounded domain.

Input: A diagnosis candidate κ
A domain repair problem (Π, π)
Output: A conflict θ

1: $\Pi \Rightarrow_\kappa \Pi'$ ▷ Apply κ to the domain of Π
2: Find the first inapplicable action a_j in π with an unsatisfied proposition $f \in \mathit{prec}(a_j)$ in the domain of Π'
3: $\theta \leftarrow \left\{\langle F_{a_j}|_f^p \rangle\right\}$
4: **for** $i \leftarrow j - 1$ to 1 **do**
5: $\quad \theta \leftarrow \theta \cup \left\{\langle F_{a_i}|_f^+ \rangle\right\}$
6: $\quad$ **if** $f \in \mathit{eff}^-(a_i)$ **then**
7: $\quad\quad \theta \leftarrow \theta \cup \left\{\langle F_{a_i}|_f^- \rangle\right\}$
8: $\quad\quad$ **break**
9: **return** θ

Fig. 1 illustrates an example of the domain repair problem. Consider the planning problem $\Pi = (\mathcal{P}, \mathcal{A}, \alpha, s^I, g)$ where $\mathcal{P} = \{l, f, q, r, z\}$, $\mathcal{A} = \{a_1, a_2, a_3\}$, $s^I = \{l\}$, and $g = \{z\}$. For each action in $\mathcal{A}$, its precondition and effects are depicted in the figure where propositions on the left are those in the precondition, and those on the right are effects. A proposition (on the right) with negation is a negative effect, and a proposition without negation is a positive one.

The plan $\langle a_1\, a_2\, a_1\, a_3 \rangle$ in Fig. 1 is not a solution because a_2 and a_3 are not applicable in the respective states where they are executed. Concretely, the proposition f is missing in those two states and r is deleted after executing a_1. The domain repair problem is to find a cardinality-minimal subset δ^* of F_Π such that π will be a solution to the planning problem Π^* obtained by applying δ^*.

One repair set which can turn the plan π into a solution is $\{\langle F_{a_2}|_f^p \rangle, \langle F_{a_3}|_f^p \rangle, \langle F_{a_3}|_r^p \rangle\}$, i.e., removing the unsatisfied preconditions from a_1 and a_3. This is however *not* a solution to the domain repair problem because it is not optimal. One solution is $\{\langle F_{a_1}|_r^- \rangle, \langle F_{a_1}|_f^+ \rangle\}$, which is a cardinality-minimal repair set.

Solving the Domain Repair Problem

We move on to show how to solve a domain repair problem (Π, π) with $\pi = \langle a_1 \cdots a_n \rangle$ by exploiting Alg. 1. Here, we are interested in the set Δ of all repair sets (diagnoses) δ such that $\Pi \Rightarrow_\delta \Pi'$ and π is a solution to Π' (i.e., the property ρ that every $\delta \in \Delta$ should satisfy), and our goal is to find a cardinality-minimal diagnosis δ^* in Δ. The dual set Θ of Δ is such that $\Theta = \{\theta \mid F_\Pi \setminus \theta \notin \Delta\}$. One can observe that the set Δ is monotonic, and henceforth, we could interpret every $\theta \in \Theta$ as a repair set containing at least one repair that must be applied in order to turn π into a solution. We also follow the convention in diagnosis to call each θ a *conflict*.

The core of using Alg. 1 to find a cardinality-minimal diagnosis δ^* is the oracle for 1) deciding whether a candidate $\kappa \subseteq F_\Pi$ satisfies ρ, and 2) extracting a set θ from the dual set Θ of Δ provided a candidate $\kappa \notin \Delta$ such that $\theta \cap \kappa = \emptyset$.

For the former, the procedure is straightforward. We first apply the candidate repair set κ to Π with $\Pi \Rightarrow_\kappa \Pi'$, and, afterwards, we verify whether the solution criteria are satisfied by the plan π in Π'.

For extracting a conflict θ with $\theta \cap \kappa = \emptyset$ given a $\kappa \notin \Delta$, we first observe that for each $\theta \in \Theta$, since the complement θ' of θ (i.e., $\theta' = F_\Pi \setminus \theta$) does not satisfy the property ρ, we have that π is *not* a solution to the planning problem Π' with $\Pi \Rightarrow_{\theta'} \Pi'$. Consequently, let κ be a repair set with $\kappa \notin \Delta$ and $\Pi \Rightarrow_\kappa \Pi'$ and a_j ($1 \leq j \leq n$) an action in π which has a proposition q in its precondition that is not satisfied in Π' (such a and q must exist because θ is not a diagnosis). A repair set θ is a conflict with $\theta \cap \kappa = \emptyset$ if it is a set of repairs at least one of which is required to turn q from being unsatisfied into being satisfied in Π' (or in other words, q can not be satisfied if *none* of the repairs in κ is applied). The key observation for this is that the complement θ' of θ contains no repairs which can make q be satisfied, and hence $\theta' \notin \Theta$, and $\theta \cap \kappa = \emptyset$ holds, because otherwise, it contradicts the fact that q is not satisfied after applying κ.

More concretely, we compute a conflict θ with $\theta \cap \kappa = \emptyset$ as follows: We first apply κ to Π such that $\Pi \Rightarrow_\kappa \Pi'$ (i.e., κ is applied to the input domain on each iteration). Let a_j ($1 \leq j \leq n$) be the first action in π which has a proposition q in its precondition that is not satisfied in Π'. We find the *largest* index i with $1 \leq i < j \leq n$ such that $q \in \mathit{eff}^-(a_i)$, and for each k with $i < k < j$, $q \notin \mathit{eff}^+(a_k)$. The conflict θ is thus the set:

$$\theta = \left\{\langle F_{a_i}|_q^-\rangle, \langle F_{a_j}|_q^p\rangle\right\} \cup \left\{\langle F_{a_k}|_q^+\rangle \mid i \leq k < j\right\}$$

Additionally, if such an i does not exist, i.e., the proposition q is not satisfied because it is missing from the initial state, then the conflict θ is simply the set:

$$\theta = \left\{\langle F_{a_j}|_q^p\rangle\right\} \cup \left\{\langle F_{a_k}|_q^+\rangle \mid i \leq k < j\right\}$$

The whole procedure is summarized in Alg. 2.

The oracle can then be employed by Alg. 1 to find an optimal set of repairs. The soundness of the entire procedure is assured by the correctness of Alg. 2.

Example As a demonstration, we show an example about how Alg. 1 together with the oracle presented above works on the domain repair problem shown in Fig. 1. The algorithm starts with an empty set Θ^* of known conflicts. On the first iteration, the MHS solver will first extract an empty set κ of repairs as a diagnosis candidate, and our oracle will then compute $\theta_1 = \{\langle F_{a_2}|_f^p\rangle, \langle F_{a_1}|_f^+\rangle\}$ as the conflict. This is because a_2 is the first action which has f in its precondition that is not satisfied, and in order to make a_2 applicable, we should either add f to a_1's positive effects or remove it from a_2's precondition.

On the second iteration, let's assume that the MHS solver returns $\kappa = \{\langle F_{a_2}|_f^p\rangle\}$ as the candidate. The oracle will then check whether κ is a diagnosis by applying it to the domain and validating the plan. The candidate κ here is not a diagnosis because it cannot turn the plan into a solution, and thus the algorithm continues to find the next conflict. Since a_3 is now the first action after applying κ which has *two* propositions f and r in its precondition that are not satisfied, our oracle will choose any one of those two to compute a new conflict. We assume that the computation is based on f, which results in the conflict $\theta_2 = \{\langle F_{a_3}|_f^p\rangle, \langle F_{a_1}|_f^+\rangle, \langle F_{a_2}|_f^+\rangle\}$.

On the third iteration, we now have the collection of the known conflicts $\Theta^* = \{\theta_1, \theta_2\}$. The MHS solver will thus return $\kappa = \{\langle F_{a_1}|_f^+\rangle\}$ as the candidate, which further results in $\{\langle F_{a_3}|_r^p\rangle, \langle F_{a_1}|_r^-\rangle\}$ as a new conflict. Lastly, on the fourth iteration, a cardinality-minimal diagnosis will be returned.

Repairing Lifted Domains

Thus far we have presented how to repair a flawed *grounded* planning domain. However, in practice, a planning domain is usually engineered as a *lifted* model. Thus, in this section, we generalize our previous approach to repair a lifted planning domain. To this end, we first introduce briefly the lifted planning formalism based on the one by Lauer et al. (2021).

Lifted Planning Formalism

The lifted planning formalism is defined on a variable set $\mathcal{V}$. A *lifted* planning problem is a tuple $\Pi = (\mathcal{P}, \mathcal{A}, \alpha, \mathcal{O}, s^I, g)$ where $\mathcal{D} = (\mathcal{P}, \mathcal{A}, \alpha)$ and $\mathcal{T} = (\mathcal{O}, s^I, g)$ are respectively the domain and task of Π. Π is thus also written as $(\mathcal{D}, \mathcal{T})$. Compared with a grounded problem, a lifted one has an extra component $\mathcal{O}$ which is a set of *objects*. Each object $o \in \mathcal{O}$ is of some *type*. For a type t, we use $\mathcal{O}[\![t]\!] \subseteq \mathcal{O}$ to refer to the subset of objects each of which is of the type t. $\mathcal{P}$ in the context of the lifted formalism is a set of *predicates*. A predicate $\mathbf{f}$ consists of a predicate name that is *unique* and a tuple of variables as parameters, written $P(v_1|_{t_1}, \cdots, v_n|_{t_n})$ for some $n \in \mathbb{N}$, where P is the predicate's name, and for each $1 \leq i \leq n$, $v_i \in \mathcal{V}$ is a variable, and the subscript t_i is a type specifying the restriction that v_i can only be substituted by an object of the type t_i. More specifically, a variable substitution on the predicate $\mathbf{f}$ is to substitute each variable v_i with an object o_i with $o_i \in \mathcal{O}[\![t_i]\!]$. A variable substitution on a predicate is also called *grounding*, and a grounded predicate is equivalent to a proposition (atom) in the grounded formalism. Given a variable substitution function $\varrho : \mathcal{V} \to \mathcal{O}$ defined over $\mathcal{V}$, the notation $\mathbf{f}[\varrho]$ indicates that $\mathbf{f}$ is grounded under the variable substitution rule defined by ϱ.

Notably, if $\mathbf{f} = P(v_1|_{t_1}, \cdots, v_n|_{t_n})$ is a predicate in $\mathcal{P}$, then any $P(v'_1|_{t_1}, \cdots, v'_n|_{t_n})$ with v'_i being an arbitrary variable for each $1 \leq i \leq n$ is also a *syntactically* correct predicate even if it is not explicitly in $\mathcal{P}$.

In the lifted formalism, $\mathcal{A}$ is a set of *action schemas*. An action schema $\mathbf{a}$ again consists of a *unique* action name and a tuple of parameters, written $A(v_1|_{t_1}, \cdots, v_n|_{t_n})$ ($n \in \mathbb{N}$). Each action schema is mapped to its precondition, positive effects, and negative effects by the function α each of which is a set of *predicates* $P(v_{i_1}|_{t_{i_1}}, \cdots, v_{i_j}|_{t_{i_j}})$ for some $j \geq 1$ where $i_k \in \{1, \cdots, n\}$ for each k with $1 \leq k \leq j$, written $\alpha(\mathbf{a}) = (\mathit{prec}(\mathbf{a}), \mathit{eff}^+(\mathbf{a}), \mathit{eff}^-(\mathbf{a}))$, i.e., every parameter of such a predicate is a parameter of the action schema.

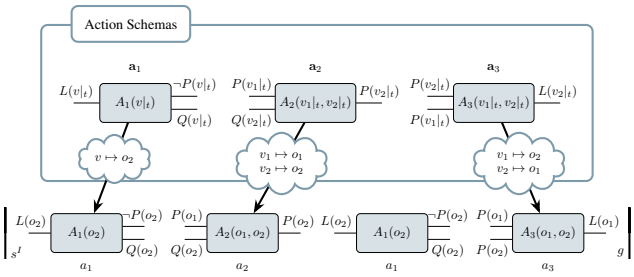

Figure 2: The lifted version of the domain (and the domain repair problem) shown in Fig. 1 together with the variable substitution functions that ground the domain.

Similar to grounding a predicate, grounding an action is to substitute each variable parameter of the action with an object of the respective type. Given a variable substitution function ϱ, we again use $\mathbf{a}[\varrho]$ to indicate that $\mathbf{a}$ is grounded under ϱ. Further, every predicate $\mathbf{f}$ in the action's precondition and effects is grounded simultaneously under the same substitution rule, i.e., $\mathbf{f}[\varrho]$. A grounded action schema is thus equivalent to an action in the grounded formalism.

Throughout the paper, we will use boldface symbols, e.g., $\mathbf{f}$ and $\mathbf{a}$, to denote predicates and action schemas in order to distinct them from (grounded) propositions and actions.

s_I and g in Π are respectively the initial state and the goal description of Π, which are two sets of *propositions* and are equivalent to their counterparts in a grounded planning problem. A solution to a lifted problem Π is an *action* sequence $\pi = \langle a_1 \cdots a_n \rangle$ such that $s_I \rightarrow_\pi^* s'$ for some s' with $g \subseteq s'$, and for each $1 \leq i \leq n$, there exist an action schema $\mathbf{a} \in \mathcal{A}$ and a variable substitution function ϱ such that $a_i = \mathbf{a}[\varrho]$.

As an example, Fig. 2 depicts the lifted version of the domain shown in Fig. 1. The lifted domain has two objects o_1 and o_2 which are of the same type t, three predicates $P(v|_t)$, $Q(v|_t)$, and $L(v|_t)$, and three action schemas each of which can be grounded to an *action* in the plan in Fig. 1 with the respective variable substitution function, as shown in Fig. 2.

Lifted Domain Repair Problem

We now formulate the domain repair problem for lifted planning domains. In contrast to the one for grounded domains, the major difference here is that the allowed repairs are also defined in a lifted way. Let $\mathbf{a} = A(v_1|_{t_1}, \cdots, v_n|_{t_n})$ be an action schema. The set $\mathbf{F_a}$ of all atomic repairs targeted at $\mathbf{a}$ is again defined as $\mathbf{F_a} = \mathbf{F_a^p} \cup \mathbf{F_a^+} \cup \mathbf{F_a^-}$ in which

- $\mathbf{F_a^p} = \{\langle \mathbf{F_a}|_{\mathbf{f}}^p \rangle \mid \mathbf{f} \in \mathit{prec}(\mathbf{a})\}$
- $\mathbf{F_a^-} = \{\langle \mathbf{F_a}|_{\mathbf{f}}^- \rangle \mid \mathbf{f} \in \mathit{eff}^-(\mathbf{a})\}$

In particular, the set $\mathbf{F_a^+}$ is the following:

$$\mathbf{F_a^+} = \left\{ \langle \mathbf{F_a}|_{\mathbf{f}}^+ \rangle \;\middle|\; \begin{array}{l} \mathbf{f} = P(v_{i_1}|_{t_{i_1}}, \cdots, v_{i_j}|_{t_{i_j}}), \\ 1 \leq k \leq j,\ i_k \in \{1, \cdots, n\} \end{array} \right\}$$

that is, $\langle \mathbf{F_a}|_{\mathbf{f}}^+ \rangle$ is a repair to $\mathbf{a}$ if every parameter of the predicate $\mathbf{f}$ is also a parameter of $\mathbf{a}$. In other words, if a repair $\langle \mathbf{F_a}|_{\mathbf{f}}^+ \rangle$ with $\mathbf{f} = P(v_{i_1}|_{t_{i_1}}, \cdots, v_{i_j}|_{t_{i_j}})$ is in $\mathbf{F_a^+}$, and for each v_{i_k} with $1 \leq k \leq j$, there exists a set $\mathcal{V}_{i_k}$ of the variables in $\mathbf{a}$'s parameters each of which is of the type t_{i_k}, then $\langle \mathbf{F_a}|_{\mathbf{f}'}^+ \rangle$ is also in $\mathbf{F_a^+}$ in which $\mathbf{f}' = P(v'_{i_1}|_{t_{i_1}}, \cdots, v'_{i_j}|_{t_{i_j}})$ with $v'_{i_k} \in \mathcal{V}_{i_k}$ for each k with $1 \leq k \leq j$. For instance, in Fig. 2, the predicates that can be added to the action schema $A_2(v_1|_t, v_2|_t)$ form the following set:

$$\{Q(x|_t), L(x|_t) \mid x \in \{v_1, v_2\}\} \cup \{P(v_1|_t)\}$$

Henceforth, given a lifted planning problem Π, the set $\mathbf{F}_\Pi$ of all atomic repairs for Π is $\bigcup_{\mathbf{a} \in \mathcal{A}} \mathbf{F_a}$. The semantics of a lifted repair is defined in the same way as that of a grounded repair, namely, the predicate is removed from or added to the action schema's precondition or effects. Given a set of repairs $\mathbf{F}'_\Pi$, we again use $\Pi \Rightarrow_{\mathbf{F}'_\Pi} \Pi'$ to indicate that Π' is obtained from Π by applying $\mathbf{F}'_\Pi$. The lifted domain repair problem $\Phi = (\Pi, \pi)$ is again to find a *cardinality-minimal* repair set $\mathbf{F}^*_\Pi$ such that $\Pi \Rightarrow_{\mathbf{F}^*_\Pi} \Pi^*$, and π is a solution to Π^*. In Fig. 2, one cardinality-minimal repair set to the lifted domain repair problem is to add $P(v_1|_t)$ to $\mathbf{a}_2$'s positive effects, remove $P(v_1|_t)$ from $\mathbf{a}_2$'s precondition, and remove $P(v_1|_t)$ from $\mathbf{a}_3$'s precondition.

Solving the Lifted Domain Repair Problem

We again solve the domain repair problem (Π, π), this time with a lifted domain, by exploiting the duality. Similar to the grounded setting, suppose Δ is the set of all repair sets (i.e., all diagnoses) δ turning π into a solution. Its dual set Θ (i.e., the set of all conflicts) is again the set of all repair sets θ whose complement *cannot* make π a solution, and we want to find a cardinality-minimal set $\delta^* \in \Delta$ in light of Alg. 1.

The oracle for Alg. 1 in the lifted setting is an extension of the one in the grounded setting. The core of the extension is finding the variable substitution function ϱ for each action a in π with $a = \mathbf{a}[\varrho]$ for some action schema $\mathbf{a}$, which thus *grounds* the domain repair problem.

More concretely, given $\pi = \langle a_1 \cdots a_n \rangle$, the procedure for deciding whether a repair set δ is in Δ is as follows:

1) We first compute the variable substitution function ϱ_i for each a_i with $1 \leq i \leq n$ such that $a_i = \mathbf{a}[\varrho_i]$ for some $\mathbf{a} \in \mathcal{A}$. This can be done by searching for an action schema $\mathbf{a}$ which has the same action name as a_i and matching each object in a_i with the respective parameter in $\mathbf{a}$.
2) We then apply δ to Π with $\Pi \Rightarrow_\delta \Pi'$ and verify whether π is a solution to Π' under the variable substitution function ϱ_i for each $1 \leq i \leq n$.

For the procedure which, given a repair set κ that is not in Δ, extracts a conflict $\theta \in \Theta$ with $\kappa \cap \theta = \emptyset$, we have seen that in the *grounded* setting, a repair set is a conflict if at least one in it is required to make a proposition f in the precondition of some action a_i ($1 \leq i \leq n$) be satisfied. This still holds in the lifted setting due to the same argument in the grounded case. Thus, we could develop the extraction procedure for a lifted domain by extending the one for a grounded domain.

Concretely, let $\Pi \Rightarrow_\kappa \Pi'$, a_j ($1 \leq j \leq n$) be the first action in π that has a proposition f in its precondition that is not satisfied in Π', and a_i the last action in π with $1 \leq i < j$ and $f \in \mathit{eff}^-(a_i)$. We first add $\langle \mathbf{F_{a_1}}|_{\mathbf{f}_1}^- \rangle$ and $\langle \mathbf{F_{a_2}}|_{\mathbf{f}_2}^p \rangle$ to θ where $\mathbf{a}_1, \mathbf{a}_2 \in \mathcal{A}$ are two action schemas with $a_i = \mathbf{a}_1[\varrho_i]$ and $a_j = \mathbf{a}_2[\varrho_j]$, and $f = \mathbf{f}_1[\varrho_i] = \mathbf{f}_2[\varrho_j]$. Further, for each a_k with $i < k < j$, we add $\langle \mathbf{F_a}|_{\mathbf{f}}^+ \rangle$ to θ with $a_k = \mathbf{a}[\varrho_k]$ for

Algorithm 3 Computing a conflict for a lifted domain.

Input: A diagnosis candidate κ
A domain repair problem (Π, π)
Output: A conflict θ

$\Pi \Rightarrow_\kappa \Pi'$ ▷ Apply κ to the domain of Π
Find the first inapplicable action a_j in π with an unsatisfied proposition $f \in \mathit{prec}(a_j)$ in the domain of Π'
Find an $\mathbf{a} \in \mathcal{A}$ and an $\mathbf{f} \in \mathcal{P}$ with $\mathbf{a}[\varrho] = a_j$ and $\mathbf{f}[\varrho] = f$ for some ϱ
$\theta \leftarrow \{\langle \mathbf{F}_\mathbf{a}|^p_\mathbf{f}\rangle\}$
for $i \leftarrow j-1$ to 1 **do**
  Find an $\mathbf{a}' \in \mathcal{A}$ with $\mathbf{a}'[\varrho'] = a_i$ for some ϱ'
  for all $\langle \mathbf{F}_{\mathbf{a}'}|^+_{\mathbf{f}'}\rangle \in \mathbf{F}^+_{\mathbf{a}'}$ **do**
    if $\mathbf{f}'[\varrho'] = f$ **then**
      $\theta \leftarrow \theta \cup \{\langle \mathbf{F}_{\mathbf{a}'}|^+_{\mathbf{f}'}\rangle\}$
  if $\exists \mathbf{f}' \in \mathit{eff}^-(\mathbf{a}') : \mathbf{f}'[\varrho'] = f$ **then**
    $\theta \leftarrow \theta \cup \{\langle \mathbf{F}_{\mathbf{a}'}|^-_{\mathbf{f}'}\rangle\}$
    break
return θ

every predicate $\mathbf{f}$ such that $f = \mathbf{f}[\varrho_k]$, and every parameter of $\mathbf{f}$ is also a parameter of $\mathbf{a}$, as shown in Alg. 3.

Example We again provide an example about how our approach works on the lifted domain in Fig. 2. We will focus on illustrating the computation of conflicts, which is the major difference from the algorithm for the grounded setting. On the first iteration, the oracle will compute $\theta_1 = \{\langle \mathbf{F}_{\mathbf{a}_2}|^p_\mathbf{f}\rangle\}$ with $\mathbf{f} = P(v_1|_t)$ as the conflict. The reason for this is that the proposition $P(o_1)$ is an unsatisfied one in the action a_2, and a_2 is obtained from grounding $\mathbf{a}_2$ by substituting v_1 with o_1. Notably, $\langle \mathbf{F}_{\mathbf{a}_1}|^+_{\mathbf{f}'}\rangle$ with $\mathbf{f}' = P(v|_t)$ is *not* in the conflict because $\mathbf{a}_1$ is grounded to a_1 by the substitution function ϱ with $\varrho(v) = o_2$, and $\mathbf{f}'[\varrho] \neq P(o_1)$.

On the second iteration, the first inapplicable action is a_3 whose precondition has two unsatisfied propositions $P(o_1)$ and $P(o_2)$. The conflict can again be computed based on any of these two propositions. Here, we assume that the computation is based on $P(o_1)$. Thus, $\{\langle \mathbf{F}_{\mathbf{a}_3}|^p_{\mathbf{f}_2}\rangle, \langle \mathbf{F}_{\mathbf{a}_2}|^+_{\mathbf{f}_1}\rangle\}$ is the conflict computed by the oracle in which $\mathbf{f}_2 = P(v_2|_t)$ and $\mathbf{f}_1 = P(v_1|_t)$. In particular, $\langle \mathbf{F}_{\mathbf{a}_2}|^+_{\mathbf{f}_1}\rangle$ is in the conflict because the variable substitution function ϱ which grounds $\mathbf{a}_2$ to a_2 satisfies that $\mathbf{f}_1[\varrho] = P(o_1)$. The algorithm will call a MHS solver to find a diagnosis candidate and start the next iteration until a solution is found.

Lifted Domains with Negative Preconditions

Having discussed how to repair a grounded and a lifted domain, we turn to consider a more complicated case which is to repair a lifted domain with *negative* preconditions. A lifted planning problem with negative preconditions $\Pi = (\mathcal{P}, \mathcal{A}, \alpha, \mathcal{O}, s^I, g)$ is identical to a normal lifted problem defined in the previous section except that the function α now maps every action schema $\mathbf{a} \in \mathcal{A}$ to a tuple: $\alpha(\mathbf{a}) = (\mathit{prec}^+(\mathbf{a}), \mathit{prec}^-(\mathbf{a}), \mathit{eff}^+(\mathbf{a}), \mathit{eff}^-(\mathbf{a})) \in 2^\mathcal{P} \times 2^\mathcal{P} \times 2^\mathcal{P} \times 2^\mathcal{P}$ where $\mathit{prec}^+(\mathbf{a})$ and $\mathit{prec}^-(\mathbf{a})$ are called the positive and the negative precondition of $\mathbf{a}$. Grounding an action schema

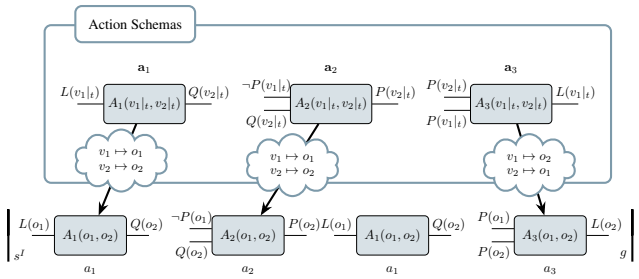

Figure 3: An example of a lifted domain with *negative preconditions*. $s^I = \{L(o_1)\}$ and $g = \{L(o_2)\}$.

with negative preconditions is again to replace each variable with an object of a respective type. Given an action schema $\mathbf{a}$ and a variable substitution function ϱ, the action a with $a = \mathbf{a}[\varrho]$ is applicable in a state s *iff* for every $\mathbf{f} \in \mathit{prec}^+(\mathbf{a})$, $\mathbf{f}[\varrho] \in s$, and for every $\mathbf{q} \in \mathit{prec}^-(\mathbf{a})$, $\mathbf{q}[\varrho] \notin s$.

Extended Atomic Repairs

Repairing a lifted domain with negative preconditions would require more atomic repairs. In particular, a predicate might need to be added to the negative effects of an action schema or removed from its positive effects in order to satisfy a negative precondition. More specifically, for each action schema $\mathbf{a} = A(v_1|_{t_1}, \cdots, v_n|_{t_n})$, we define three extra sets of repairs, $\mathbf{N}^p_\mathbf{a}$, $\mathbf{N}^+_\mathbf{a}$, and $\mathbf{N}^-_\mathbf{a}$, as follows.

- $\mathbf{N}^p_\mathbf{a} = \{\langle \mathbf{N}_\mathbf{a}|^p_\mathbf{f}\rangle \mid \mathbf{f} \in \mathit{prec}^-(\mathbf{a})\}$
- $\mathbf{N}^+_\mathbf{a} = \{\langle \mathbf{N}_\mathbf{a}|^+_\mathbf{f}\rangle \mid \mathbf{f} \in \mathit{eff}^+(\mathbf{a})\}$
- $\mathbf{N}^-_\mathbf{a} = \left\{\langle \mathbf{N}_\mathbf{a}|^-_\mathbf{f}\rangle \;\middle|\; \begin{array}{l}\mathbf{f} = P(v_{i_1}|_{t_{i_1}}, \cdots, v_{i_j}|_{t_{i_j}}),\\ 1 \leq k \leq j,\ i_k \in \{1, \cdots, n\}\end{array}\right\}$

Intuitively, each repair $\langle \mathbf{N}_\mathbf{a}|^p_\mathbf{f}\rangle$ is to *remove* a predicate from the *negative* precondition of $\mathbf{a}$, $\langle \mathbf{N}_\mathbf{a}|^+_\mathbf{f}\rangle$ is to *remove* a predicate from the *positive* effects of $\mathbf{a}$, and $\langle \mathbf{N}_\mathbf{a}|^-_\mathbf{f}\rangle$ is to *add* a predicate to its *negative* effects. Notably, like adding a predicate to an action schema's positive effects, a predicate can be added to its negative effects if every parameter of the predicate is also a parameter of the action. Consequently, in repairing a lifted domain with negative preconditions, the set $\mathbf{R}_\mathbf{a}$ of all atomic repairs targeted at an action schema $\mathbf{a}$ is thus $\mathbf{F}_\mathbf{a} \cup \mathbf{N}^p_\mathbf{a} \cup \mathbf{N}^+_\mathbf{a} \cup \mathbf{N}^-_\mathbf{a}$ where $\mathbf{F}_\mathbf{a}$ is the set of all atomic repairs targeted at $\mathbf{a}$ in repairing a domain without negative preconditions, and the set $\mathbf{R}_\Pi$ of all repairs allowed in repairing a domain with negative precondition is $\bigcup_{\mathbf{a}\in\mathcal{A}} \mathbf{R}_\mathbf{a}$.

Solving the Repair Problem

Let (Π, π) be a domain repair problem with negative preconditions. We again want to find a cardinality-minimal diagnosis δ^* from the set Δ containing all repair sets δ that turn π into a solution. Although one can easily notice that the dual set Θ of Δ is again the set of all repair sets θ whose complement cannot turn π into a solution, Alg. 1 is *not* applicable here because Δ is *not* monotonic, and hence a diagnosis $\delta \in \Delta$ does *not* have to be a hitting set of Θ.

Specifically, the reason causing Δ not being monotonic is the dependency between repairs introduced by the presence of negative preconditions, that is, if some repair is applied,

then another one must also be applied simultaneously. As a consequence, for some diagnosis $\delta \in \Delta$, not all super sets of δ are in Δ, but only *some* of them are.

In order to deal with the dependency and adapt the hitting set based Alg. 1, we introduce *conditional conflicts* which generalizes conflicts. A conditional conflict is a pair (φ, θ) each of which is a subset of $\mathbf{R}_\Pi$. The interpretation of it is that if *all* repairs in φ are applied to Π, then *at least one* repair (which could be anyone) in θ should also be applied in order to turn π into a solution. In particular, a conflict θ is a special conditional conflict $(\emptyset, \theta)$ which means that if *no* repairs have been applied, then at least one repair in θ shall be applied to Π. Given a set Θ of conditional conflicts, a set of repairs δ *hits* Θ if for any $(\varphi, \theta) \in \Theta$, if $\varphi \subseteq \delta$ or $\varphi = \emptyset$, then $\delta \cap \theta \neq \emptyset$. In other words, for some $(\varphi', \theta') \in \Theta$, if $\varphi' \neq \emptyset$ and $\varphi' \nsubseteq \delta$, then it is allowed that $\delta \cap \theta' = \emptyset$.

The advantage of adapting the notion of conditional conflicts is that every diagnosis $\delta \in \Delta$ is now a hitting set of the set Θ^* of *all* conditional conflicts, which follows trivially from the definition of conditional conflicts. Henceforth, we can again exploit Alg. 1 if we have the procedure for extracting a conditional conflict $(\varphi, \theta) \in \Theta^*$ given a set of repair $\kappa \notin \Delta$ such that $\theta \cap \kappa = \emptyset$.

Given a set of repairs $\kappa \notin \Delta$, the procedure for computing such a conditional conflict is again based upon a set θ of repairs at least one of which is required to turn a proposition f in the precondition (either positive or negative) of an action a_j in π from being unsatisfied into being satisfied in Π' with $\Pi \Rightarrow_\kappa \Pi'$. Concretely, we first let $\varphi = \emptyset$. If there exist no repairs in θ that *undo* a repair in κ, then $(\emptyset, \theta)$ is the conditional conflict. Otherwise, for *any* repair $\tau \in \theta$ that undoes a repair $\tau' \in \kappa$, τ is removed from θ and τ' is added to φ, and the conditional conflict is (φ, θ). In particular, by τ undoing τ', we mean that the consequence of applying τ' is eliminated by applying τ. E.g., $\langle \mathbf{F}_\mathbf{a}|^+_\mathbf{f} \rangle$ undoes $\langle \mathbf{N}_\mathbf{a}|^+_\mathbf{f} \rangle$ because the latter one removes $\mathbf{f}$ from the positive effects of $\mathbf{a}$ but the former one adds it back.

This procedure ensures that $(\varphi, \theta) \in \Theta^*$. To prove this, we first consider the case where $\varphi = \emptyset$. This implies that none of the repairs in θ undo a repair in κ. It follows that the proposition f in the precondition of a_j in π is also unsatisfied in Π. Thus, at least one repair in θ shall be applied to Π in order to satisfy f. Thus, $(\emptyset, \theta)$ is a conditional conflict. On the other hand, if $\varphi \neq \emptyset$, it means that f is initially satisfied in Π, but it then becomes unsatisfied after applying the repairs in φ to Π. This is equivalent to saying that if all repairs in φ are applied to Π, then at least one repair in θ shall also be applied, which thus makes the pair a conditional conflict.

The remaining question is how to obtain the repair set θ in repairing domains with negative preconditions which makes the proposition f in the precondition of the action a_j be satisfied. One can easily observe that if $f \in \mathit{prec}^+(a_j)$, then the procedure for this is *identical* to the one in repairing a domain without negative preconditions. On the other hand, if $f \in \mathit{prec}^-(a_j)$, we find the action a_i in π with the largest index i such that $1 \leq i < j$ and $f \in \mathit{eff}^+(a_i)$. The two repairs $\langle \mathbf{N}_{\mathbf{a}_1}|^p_{\mathbf{f}_1} \rangle$ and $\langle \mathbf{N}_{\mathbf{a}_2}|^+_{\mathbf{f}_2} \rangle$ are added to θ with $a_i = \mathbf{a}_1[\varrho_i]$, $a_j = \mathbf{a}_2[\varrho_j]$, and $f = \mathbf{f}_1[\varrho_i] = \mathbf{f}_2[\varrho_j]$ for some variable substitution functions ϱ_i and ϱ_j, action schemas $\mathbf{a}_1$ and $\mathbf{a}_2$, and predicates $\mathbf{f}_1$ and $\mathbf{f}_2$. Then, for each k from $j-1$ to $i+1$ in decreasing order, $\langle \mathbf{N}_\mathbf{a}|^-_\mathbf{f} \rangle$ is added to θ if $a_k = \mathbf{a}[\varrho_k]$ and $f = \mathbf{f}[\varrho_k]$ for some variable substitution function ϱ_k, and every parameter of $\mathbf{f}$ is also a parameter of $\mathbf{a}$. If $\langle \mathbf{N}_\mathbf{a}|^-_\mathbf{f} \rangle$ already undoes a repair in the current candidate, we stop the procedure without proceeding to the next k. Alg. 4 shows the complete procedure for computing a conditional conflict.

Additionally, in order to employ Alg. 1 here, we also need an operator which computes a hitting set for a set of conditional conflicts. Fortunately, such an algorithm is also well-developed, e.g., see the work by Struss and Dressler (1989)

Example We provide an example of how *conditional* conflicts are computed by the oracle. For this, consider the lifted domain repair problem with negative preconditions shown in Fig. 3. On the first iteration, $\{\langle \mathbf{F}_{\mathbf{a}_3}|^p_{\mathbf{f}_2} \rangle, \langle \mathbf{F}_{\mathbf{a}_1}|^+_{\mathbf{f}_1} \rangle, \langle \mathbf{F}_{\mathbf{a}_2}|^+_{\mathbf{f}_1} \rangle\}$ will be computed as the *unconditional* conflict where $\mathbf{f}_1 = P(v_1|_t)$ and $\mathbf{f}_2 = P(v_2|_t)$. For the purpose of demonstration, we assume that the MHS solver chooses $\{\langle \mathbf{F}_{\mathbf{a}_1}|^+_{\mathbf{f}_1} \rangle\}$ as the candidate on this iteration.

On the next iteration, the action a_2 will then become inapplicable with the proposition $P(o_1)$ in its *negative* precondition being unsatisfied. The repair set for resolving this problem is $\{\langle \mathbf{N}_{\mathbf{a}_2}|^p_{\mathbf{f}_1} \rangle, \langle \mathbf{N}_{\mathbf{a}_1}|^-_{\mathbf{f}_1} \rangle\}$ with $\mathbf{f}_1 = P(v_1|_t)$. Since $\langle \mathbf{N}_{\mathbf{a}_1}|^-_{\mathbf{f}_1} \rangle$ undoes $\langle \mathbf{F}_{\mathbf{a}_1}|^+_{\mathbf{f}_1} \rangle$, the oracle thus produces the conditional conflict $(\{\langle \mathbf{F}_{\mathbf{a}_1}|^+_{\mathbf{f}_1} \rangle\}, \{\langle \mathbf{N}_{\mathbf{a}_1}|^+_{\mathbf{f}_1} \rangle\})$. The remaining part of the algorithm works in the same way as in the lifted case without negative preconditions.

Domains with Multiple Problem Instances

Thus far, we only consider repairing a flawed (lifted) domain on which only *one* planning problem instance Π is defined. However, in practice, multiple planning problem instances can be defined on the *same* domain. Thus, in this section, we consider the problem of repairing a lifted domain which is associated with multiple *tasks*.

More specifically, the configuration of the problem is as follows: Given a lifted domain $\mathcal{D} = (\mathcal{P}, \mathcal{A}, \alpha)$, a set of planning problems $\Psi = \{\Pi_1, \cdots, \Pi_n\}$ each of which is defined on $\mathcal{D}$, i.e., $\Pi_i = (\mathcal{D}, \mathcal{T}_i)$ with $\mathcal{T}_i = (\mathcal{O}_i, s_i^I, g_i)$ for each $1 \leq i \leq n$, and a set of plans $\{\pi_1, \cdots, \pi_n\}$, our goal is to repair the domain $\mathcal{D}$ such that π_i is a solution to Π_i for each $1 \leq i \leq n$. In particular, the domain $\mathcal{D}$ has both positive and negative preconditions, and the set of all atomic repairs $\mathbf{R}$ is identical to $\mathbf{R}_\Pi$ for the case where only one problem Π is considered. We are again interested in finding a minimal repair set that turns each π_i into a solution to Π_i.

Suppose Δ is the set of all repair sets that turn each π_i $(1 \leq i \leq n)$ into a solution to Π_i, and Θ is the set of all conditional conflicts (φ, θ) such that if any repair in φ is applied, then at least one in θ should also be applied to turn *every* π_i into a solution. Unsurprisingly, we again exploit Alg. 1 to find a minimal diagnosis $\delta^* \in \Delta$ that hits every conditional conflict in Θ. The core of designing the procedure for extracting a conditional conflict given a repair set $\kappa \notin \Delta$ is the fact that for each $1 \leq i \leq n$, every conditional conflict for (Π_i, π_i) is also in Θ. Thus, in each iteration in Alg. 1, instead of extracting only one conditional

Algorithm 4 Computing a conditional conflict.

Input: A diagnosis candidate κ
A domain repair problem (Π, π)
Output: A conditional conflict (φ, θ)

$\Pi \Rightarrow_\kappa \Pi'$ ▷ Apply κ to the domain of Π
Find the first inapplicable action a_j in π with an unsatisfied precondition f in the domain of Π'
Find an $\mathbf{a} \in \mathcal{A}$ and an $\mathbf{f} \in \mathcal{P}$ with $\mathbf{a}[\varrho] = a_j$ and $\mathbf{f}[\varrho] = f$ for some ϱ
▷ Define some notations for convenience
if $f \in prec^+(a_j)$ **then**
    $\mathbf{C} \leftarrow \mathbf{F}$; $s_1 \leftarrow +$; $s_2 \leftarrow -$
else $\mathbf{C} \leftarrow \mathbf{N}$; $s_1 \leftarrow -$; $s_2 \leftarrow +$
$\theta \leftarrow \{\langle \mathbf{C}_\mathbf{a} |^p_\mathbf{f} \rangle\}$; $\varphi \leftarrow \emptyset$
for $i \leftarrow j - 1$ to 1 **do**
    Find an $\mathbf{a}' \in \mathcal{A}$ with $\mathbf{a}'[\varrho'] = a_i$ for some ϱ'
    for all $\langle \mathbf{C}_{\mathbf{a}'} |^{s_1}_{\mathbf{f}'} \rangle \in \mathbf{C}^{s_1}_{\mathbf{a}'}$ **do**
        if $\langle \mathbf{C}_{\mathbf{a}'} |^{s_1}_{\mathbf{f}'} \rangle$ undoes an $x \in \kappa$ **then**
            $\varphi \leftarrow \varphi \cup \{\langle \mathbf{C}_{\mathbf{a}'} |^{s_1}_{\mathbf{f}'} \rangle\}$
        else $\theta \leftarrow \theta \cup \{\langle \mathbf{C}_{\mathbf{a}'} |^{s_1}_{\mathbf{f}'} \rangle\}$
    if $\exists \mathbf{f}' \in \mathit{eff}^{s_2}(\mathbf{a}') : \mathbf{f}'[\varrho] = f$ **then**
        if $\langle \mathbf{C}_{\mathbf{a}'} |^{s_2}_{\mathbf{f}'} \rangle$ undoes an $x \in \kappa$ **then**
            $\varphi \leftarrow \varphi \cup \{\langle \mathbf{C}_{\mathbf{a}'} |^{s_2}_{\mathbf{f}'} \rangle\}$ **break**
        else $\theta \leftarrow \theta \cup \{\langle \mathbf{C}_{\mathbf{a}'} |^{s_2}_{\mathbf{f}'} \rangle\}$ **break**
    if $|\varphi| > 0$ **then break**
return (φ, θ)

conflict provided the repair set κ, we compute one θ_i for *each* (Π_i, π_i), $1 \leq i \leq n$, and add it to the collections of known conditional conflicts. The minimal hitting set of the collection is thus the cardinality-minimal diagnosis.

Experimental Results

Now we present our experimental results (Lin, Grastien, and Bercher 2023) which characterize the efficiency of our approach. The experiments ran on an Intel Xeon processor with 16GB memory, and the indicator of the efficiency is the runtime for repairing flawed domains.

To our best knowledge, there existed no benchmark sets of flawed planning domains before the experiment was done. Thus, we had to create our own for the experiment. In particular, we created two benchmark sets[1]. The first benchmark set $\mathcal{G}_1$ contains flawed domains each of which is associated with one planning problem, and each domain in the second one $\mathcal{G}_2$ corresponds to multiple problem instances.

Both benchmark sets were created on top of the *fast-downward (FD) problem suite*[2]. We created one solution π_i per task $\mathcal{T}_i$ using FD. For benchmark set $\mathcal{G}_1$, we selected 10%, 30%, or 50% of the respective domain's action schemas and introduced a single error to them, by either adding an additional precondition or effect (randomly positive or negative) or removing an effect. This gives three sets of n pairs consisting of a flawed domain and its plan. For the benchmark set $\mathcal{G}_2$ we did the same, only that each of the n pairs in the three sets shares the same domain, as we don't change the action schemas per plan in each set (i.e., 10%, 30%, or 50%), but just once for the domain (recall that the n problem instances share the same domain). Both $\mathcal{G}_1$ and $\mathcal{G}_2$ contain 38 domains, and there are 5460 instances (pairs of a flawed domain and a plan) in $\mathcal{G}_1$ and 243 instances (pairs of a flawed domain and a set of plans) in $\mathcal{G}_2$.

[1]https://github.com/Songtuan-Lin/repairing-benchmarks
[2]https://github.com/aibasel/downward-benchmarks

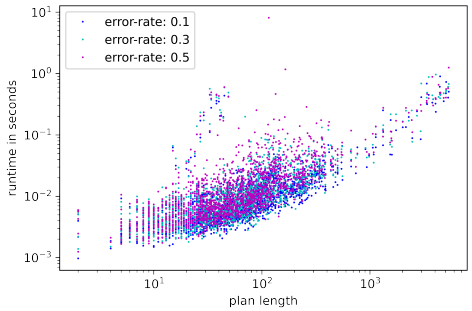

Figure 4: Runtime in seconds for solving each domain repair problem instance against the respective plan length.

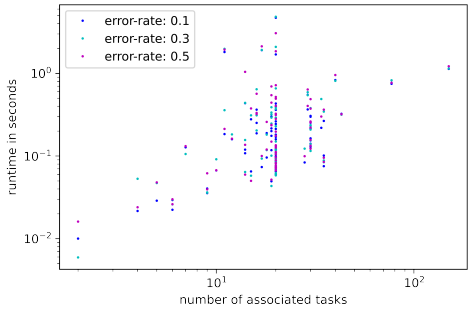

Figure 5: Runtime for repairing each domain against the number of tasks associated with it.

Fig. 4 depicts the runtime for each instance against the respective plan length for $\mathcal{G}_1$. The x-coordinate of a point indicates the runtime, and the y-coordinate is the respective plan length. The figure respectively shows the runtime for instances with 10%, 30%, and 50% errors. One can observe that all instances were solved in half a second except one which needs more than one second, and unsurprisingly, runtime grows with plan length in general.

The experimental results on $\mathcal{G}_2$ are shown in Fig. 5, which depicts the runtime for repairing each domain against the number of tasks (plans) associated with it. Domains with different error rates are again shown separately. The runtime for solving instances in $\mathcal{G}_2$ is longer on average than those in $\mathcal{G}_1$, which is unsurprising due to the larger problem sizes.

We believe that the experimental results show that our approach is efficient enough to be deployed in practice. The number of errors we introduced is larger than what we expect would happen in practice. The average length of plans in $\mathcal{G}_1$ and the average number of tasks associated with domains in $\mathcal{G}_2$ are also significantly larger than a domain mod-

eler could provide (see Fig. 4 and 5), whereas our approach can still solve them effectively. This means that our method can provide repair suggestions to a user almost instantly – as it would be required once being deployed in practice.

Lastly, we briefly discuss the quality of the repairs found by our approach. On $\mathcal{G}_1$, 72.63% of the returned repairs recover an error, and this number on $\mathcal{G}_2$ is 76.43%. However, we did not consider these numbers as an indicator of our approach. In fact, we argue that the notion of quality of repairs does not coincide with modeling assistance. More specifically, we could evaluate the "quality" of the repairs in our experiments *only* because we know in advance what errors are introduced to the domains. This is however not the case for practical modeling assistance scenarios where errors are introduced *accidentally*. In other words, the domain modeler does not know what are the errors in the domain. Consequently, the domain modeler can only decide whether a repair is *preferred* but *cannot* justify whether it is *correct*.

Discussion and Related Works

Lastly, we briefly review some works related to modeling assistance. Planning.Domains (Muise 2016), MyPDDL (Strobel and Kirsch 2020), Web Planner (Magnaguagno et al. 2020) and the VSCode PDDL plugin support a wide range of features for writing and testing PDDL files, and itSIMPLE (Vaquero et al. 2012) can automatically generate a PDDL file from scratch from a UML diagram.

Apart from supporting modeling on the *syntax* level, techniques have also been developed for more advanced modeling assistance. For instance, itSIMPLE also supports planning domain analysis done by analyzing the Petri Net obtained from an input UML diagram. Hoffmann (2011) developed the tool Torchlight which detects whether local minima exists in a planning task when employing the heuristic h^+ (Hoffmann and Nebel 2001). Zhang and Muise (2020) studied an approach for detecting unusable actions in a planning domain (which are usually caused by modeling errors). Both Torchlight and action unusability detection have been incorporated into Planning.Domain as plugins.

Additionally, Sreedharan et al. (2020) studied how to deal with an incorrect dialogue domain via techniques developed for explainable AI planning (Chakraborti, Sreedharan, and Kambhampati 2020), e.g., via model reconciliation (Sreedharan, Chakraborti, and Kambhampati 2021; Sreedharan, Bercher, and Kambhampati 2022). Lindsay et al. (2020) proposed an approach based upon machine learning which refines an inaccurate hybrid domain model. Göbelbecker et al. (2010) and Gragera, García-Olaya, and Fernández (2022) investigated approaches which turn an unsolvable planning problem into a solvable one. Coulter et al. (2022) developed an approach which can verify whether two planning problems have the same solution set and can identify actions that cause the difference. This is particularly useful in educating students how to model a domain. In that context, one might provide a reference domain model and compare it against another model created by a student, which can thus help the student identify the errors in the created model.

Apart from works contributing directly to modeling assistance, techniques from other disciplines can also be exploited to develop further tools for modeling assistance. One such example, as mentioned early, is the field of *diagnosis* on which we develop our approach. In our earlier work (Lin, Grastien, and Bercher 2022), we have rigorously framed a *grounded* domain repair problem without negative preconditions as a diagnosis problem, providing an in-depth insight into the connection between these two problems. Additionally, Crow and Rushby (1991) and Stumptner and Wotawa (1999) proposed to use diagnosis techniques to reconfigure a system, which is an alternative view of the domain repair problem. More importantly, many approaches proposed for diagnosis can be exploited for modeling assistance, e.g., sequential diagnosis (Rodler 2020) where different diagnoses are proposed in accordance with a user's input.

Another related discipline is *domain learning*. The task of domain learning is to *learn* a domain from scratch (or an incomplete model). Domain learning systems usually rely on different assumptions and restrictions. For instance, LOCM (Cresswell, McCluskey, and West 2009; Cresswell and Gregory 2011) assumes that the actions in the domain to be learned can be divided into different sorts, and the actions in the same sort behave identically, and PELA (Celorrio, Fernández, and Borrajo 2008) requires a set of sound state trajectories as input obtained by monitoring the execution of a set of plans. For a more broadly review of techniques for domain learning, we refer to the work by Arora et al. (2018).

In future works, we will incorporate more advanced diagnosis techniques and domain learning techniques into repairing flawed domain models and modeling assistance. Further, we have investigated the complexity of correcting a domain by being given both a set of positive plans (which are supposed to be solutions) and of negative plans (which are not supposed to be solutions) (Lin and Bercher 2023), though we did this only for hierarchical planning. We will extend our approach to accept both positive and negative plans.

More importantly, we are planning to build an interactive system based upon the presented approach which can repair a flawed domain iteratively. On each iteration, the system outputs a set of repairs (for the input flawed domain) by calling our approach, and the user can decide which repairs are preferred and forbid those that are not. The process continues until the user finds all preferred repairs.

Conclusion

We proposed a hitting-set based approach to repair a flawed domain, which can be deployed for providing modeling assistance. We are one of the first who developed a practical method for this purpose. Our experimental results showed that our approach is ready to be deployed in practice as it can provide repair suggestions almost instantly. The efficiency of our approach also indicates that it can be called iteratively until all satisfactory repairs are found in which implausible repairs are blocked at each iteration.

References

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review*, 33.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning - One Abstract Idea, Many Concrete Realizations. In *IJCAI 2019*, 6267–6275. IJCAI.

Celorrio, S. J.; Fernández, F.; and Borrajo, D. 2008. The PELA Architecture: Integrating Planning and Learning to Improve Execution. In *AAAI 2008*, 1294–1299. AAAI.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The Emerging Landscape of Explainable Automated Planning & Decision Making. In *IJCAI 2020*, 4803–4811. IJCAI.

Coulter, A.; Ilie, T.; Tibando, R.; and Muise, C. 2022. Theory Alignment via a Classical Encoding of Regular Bisimulation. In *KEPS 2020*.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. In *ICAPS 2011*, 42–49. AAAI.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *ICAPS 2009*, 338–341. AAAI.

Crow, J.; and Rushby, J. M. 1991. Model-Based Reconfiguration: Toward an Integration with Diagnosis. In *AAAI 1991*, 836–841. AAAI.

de Kleer, J.; Mackworth, A. K.; and Reiter, R. 1992. Characterizing Diagnoses and Systems. *AIJ*, 56: 197–222.

de Kleer, J.; and Williams, B. C. 1987. Diagnosing Multiple Faults. *AIJ*, 32: 97–130.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning – Theory and Practice*. Morgan Kaufmann.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *ICAPS 2010*, 81–88. AAAI.

Gragera, A.; García-Olaya, Á.; and Fernández, F. 2022. Repair Suggestions for Planning Domains with Missing Actions Effects. In *XAIP 2022*.

Haslum, P.; Muise, C.; Magazzeni, D.; and Lipovetzky, N. 2019. *An Introduction to the Planning Domain Definition Language*. Springer.

Haslum, P.; Slaney, J. K.; and Thiébaux, S. 2012. Minimal Landmarks for Optimal Delete-Free Planning. In *ICAPS 2012*, 353 – 357. AAAI.

Hoffmann, J. 2011. Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h+. *JAIR*, 41: 155 – 229.

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

Lauer, P.; Torralba, Á.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *IJCAI 2021*, 4119–4126. IJCAI.

Lin, S.; and Bercher, P. 2021. Change the World - How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *IJCAI 2021*, 4152–4159. IJCAI.

Lin, S.; and Bercher, P. 2023. Was Fixing this *Really* That Hard? On the Complexity of Correcting HTN Domains. In *AAAI 2023*. AAAI.

Lin, S.; Grastien, A.; and Bercher, P. 2022. Planning Domain Repair as a Diagnosis Problem. In *DX 2022*.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Experimental Results for the AAAI 2023 Paper "Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains". doi: 10.5281/ZENODO.7690016.

Lindsay, A.; Franco, S.; Reba, R.; and McCluskey, T. L. 2020. Refining Process Descriptions from Execution Data in Hybrid Planning Domain Models. In *ICAPS 2020*, 469–477. AAAI.

Magnaguagno, M. C.; Pereira, R. F.; Móre, M. D.; and Meneguzzi, F. 2020. Web Planner: A Tool to Develop, Visualize, and Test Classical Planning Domains. In *Knowledge Engineering Tools and Techniques for AI Planning*, 209–227. Springer.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *K-CAP 2017*, 14:1–14:8. ACM.

Muise, C. 2016. Planning.Domains. In *ICAPS – Demo 2016*.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *AIJ*, 32: 57–95.

Rodler, P. 2020. Reuse, Reduce and Recycle: Optimizing Reiter's HS-Tree for Sequential Diagnosis. In *ECAI 2020*, volume 325, 873–880. IOS.

Slaney, J. 2014. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In *ECAI 2014*, 843–848. IOS.

Sreedharan, S.; Bercher, P.; and Kambhampati, S. 2022. On the Computational Complexity of Model Reconciliations. In *IJCAI-ECAI 2022*, 4657–4664. IJCAI.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of Explanations as Model Reconciliation. *AIJ*, 301: 103558.

Sreedharan, S.; Chakraborti, T.; Muise, C.; Khazaeni, Y.; and Kambhampati, S. 2020. – D3WA+ – A Case Study of XAIP in a Model Acquisition Task for Dialogue Planning. In *ICAPS 2020*, 488–498. AAAI.

Strobel, V.; and Kirsch, A. 2020. MyPDDL: Tools for Efficiently Creating PDDL Domains and Problems. In *Knowledge Engineering Tools and Techniques for AI Planning*, 67–90. Springer.

Struss, P.; and Dressler, O. 1989. "Physical Negation" Integrating Fault Models into the General Diagnostic Engine. In *IJCAI 1989*, 1318–1323. Morgan Kaufmann.

Stumptner, M.; and Wotawa, F. 1999. Reconfiguration using model-based diagnosis. In *DX 1999*, 266–271.

Vaquero, T.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J.; and Beck, C. 2012. itSIMPLE4.0: Enhancing the Modeling Experience of Planning Problems. In *ICAPS – Demo 2012*, 11–14.

Zhang, Q.; and Muise, C. 2020. Action Usability via Dead-end Detection. In *KEPS 2020*.