Chair of Computer Science 6 (Machine Learning and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Learning Lifted STRIPS Models from Partial Graphs with Reduced Action Labels

Lernen von gehobenen STRIPS-Modellen aus partiellen Graphen mit reduzierten Aktionskennzeichnungen

Master Thesis
Masterarbeit

Presented by / Vorgelegt von

Niklas Jansen
378845

Supervised by / Betreut von Jonas Gösgens, M.Sc.

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Prof. Dr. rer. nat. Christopher Morris

Aachen, February 18, 2025

Abstract

Planning problems can be solved by a domain-independent planner, but for this the planner needs a formal domain description. Since creating them is complex, learning the domains is a relevant problem. Recently, SIFT was introduced, which is a scalable and simple algorithm that learns action models only from action traces. In these traces, the states of the underlying problem are not observable. The only information used are the grounded actions that must have a STRIPS-like structure. In this thesis, we will first give some additional explanations of SIFT. Afterward, we show that STRIPS-like labels are not needed to learn a domain. For this we will introduce a preprocessing of the traces which adds additional arguments. We will formally show which arguments can be recovered under which conditions. Our experimental evaluation shows that the algorithm can find additional arguments, but is not computationally feasible for all domains.

Contents

1 Introduction

To solve a planning problem, on the one hand, a planner is needed and on the other hand a formal description of the planning problem. While planners are extensively researched, there is not yet much about automatic description learning, especially when there is no state observability. The manual creation of planning domains is complex and time consuming, therefore algorithms that learn these domains are interesting. The problem of learning a domain is very relevant and there are even plans to host a competition on domain learning [3].

Recently, the algorithm SIFT [9] was introduced. This algorithm learns the predicates and the action schema of a domain from traces alone. That means that the algorithm does not have any information about the states. The only information are action labels, which must have a STRIPS-like structure. That means that an action needs to contain all arguments which are contained in an atom that changes by the action or on which the action has a precondition.

The goal of this thesis is to learn the action schema with traces that do not have STRIPS-like labels. That means that for the action schema that is used to generate the action labels, there can be missing arguments. Therefore, an action schema does not need to contain all objects contained in preconditions or effects. The proposed algorithm is a preprocessing for the SIFT algorithm. The input for the proposed algorithm is an action trace with state equality. That means that the input is a graph-like structure. Then, by preprocessing, additional arguments for actions can be detected and added to the argument lists. That means that the arguments that are not contained in the action labels are recovered. It will be shown under what conditions this is possible. The goal is that after the preprocessing, the action labels have a STRIPS-like structure. Therefore, after the proposed preprocessing, the SIFT algorithm can learn the domain from the traces with the added arguments.

An example of the arguments that may be removed from the inputs can be found in the blocks world domain. Here, the action $unstack(x, y)$ removes a block x from a block y. In this domain, a block x can be stacked on at most one block y in each state. To $unstack$ the block x from the block y, on which x is currently stacked, y does not need to be given. This is because by the sequence of actions that reaches this state, it is clear on which block x is currently stacked. Therefore, the argument y may be missing in the traces. Later, it will be shown that the block y can be recovered.

The thesis is structured as follows. In Section 2 there will be a brief introduction to planning problems. In Section 3 there will be an overview of current action model learning algorithms and some insight into planning problems. In this section, especially, the algorithm SIFT will be

introduced. For the SIFT algorithm, there will be some additional explanations in Section 4. In Section 5 the preprocessing algorithm will be introduced and it will be theoretically proven that missing arguments can be recovered. This will be followed by an experimental evaluation in Section 6. Afterward, there will be a conclusion and an outlook to future work.

2 Background

This section gives a brief introduction to the languages used in planning. For this first, a definition of classical planning problems is given. Then a language in which planning problems can be formulated is introduced.

2.1 Classical Planning and STRIPS

Classical planning [8] is a part of AI where the goal is to reach a goal state through action sequences. A classical planning problem can be described in STRIPS [7] as a tuple $P = (\mathcal{D}, \mathcal{I})$ such that:

- $\mathcal{D} = (F, O)$ is the domain,
- $\mathcal{I} = (I, G, \mathcal{O})$ is the instance.

The domain $\mathcal{D}$ contains a set of predicates F and a set of actions O. The instance $\mathcal{I}$ consists of an initial situation I, a goal situation G, and a finite set of objects $\mathcal{O}$.
An action can be split into three lists, namely the add-, delete-, and precondition-lists. Each element in the precondition list $pre(o)$ for an action $o \in O$ must hold so that the action is applicable. The elements in the list $add(o)$ and $del(o)$ for an action $o \in O$ have to be true and false, respectively, in the successor state.
A planning problem $P = (\mathcal{D}, \mathcal{I})$ with $\mathcal{D} = (F, O)$ and $\mathcal{I} = (I, G, \mathcal{O})$ defines a state model $S(P) = \langle S, s_0, S_G, Act, A, f, c \rangle$, where

- $s \in S$ is a collection of atoms from F,
- $s_0 \in S$ is the initial state,
- S_G is a set of states $s \in S$ such that $G \subseteq s$,
- Act, is a set of grounded actions,
- $A(s) \subseteq Act$ are the applicable actions in $s \in S$,
- $f(a, s) = s' = (s \cup add(a) \setminus del(a))$ is a deterministic transition function, $s, s' \in S$, $a \in A(s)$
- c is a cost function.

This state model can be transferred to a labeled graph $G(P) = (V, E, L)$. For each state $s \in S$, there is a node $g(s) \in V$. For each transition $f(a, s) = s'$, there is an edge $(g(s), g(s')) \in E$. In addition to this, there is a set of labels such that for each transition $f(a, s) = s'$ there is a label $(g(s), l, g(s')) \in L$. The label $l = (a(o))$ contains on the one hand the action name $a \in O$ and on the other hand a list of objects $o = o_1, o_2, ..., o_n \in \mathcal{O}$. This list contains all objects that are

$\mathcal{D}$: F = { on(?x, ?y), free(?x), table(?x) }
O = { stack(?x, ?y): P = { free(?x), free(?y), table(?x) },
A = { on(?x, ?y) },
D = { free(?y), table(?x) } ;
unstack(?x, ?y): P = { free(?x), on(?x, ?y) },
A = { free(?y), table(?x) },
D = { on(?x, ?y) }
$\mathcal{I}$: I = { on(block2, block3), free(block1), free(block2),
table(block1), table(block3) }
G = { on(block2, block3), on(block3, block1) }
$\mathcal{O}$ = { block1, block2, block3 }

Figure 2.1: STRIPS representation of the blocks world domain with 2 actions.

part of an atom that is contained in any of the lists $add(a)$, $del(a)$, and $pre(a)$.
An example STRIPS description for a blocks world instance can be seen in Figure 2.1. The instance contains 3 blocks where in the initial situation there is a tower of height 2 and a tower of height 1. The goal is that $block3$ is stacked on $block1$ and $block2$ on $block3$.
The STRIPS representation in Figure 2.1 is a first-order symbolic representation. That means that the actions are parameterized by variables, where the variables correspond to objects of the domain. These representations are also called lifted representations. Another way to express this blocks world instance would be as a grounded representation. The grounded representation can be obtained from the first-order symbolic representation by replacing the symbolic variables with binary variables. For example, $table\_block1$ would be a binary variable for the symbolic variable $table(?x)$ with $?x = block1$. Then also the action schema has to be changed. The action has to be defined for each combination of binary variables for which it is applicable.

In this STRIPS representation, there can only be preconditions on the true atoms. To argue about the false atoms of a predicate p there needs to be a predicate $\bar{p}$. For these predicates, the effects are switched, meaning that an atom is deleted for $\bar{p}$ if it is added for p. The same is done for adding effects of $\bar{p}$. Therefore, an atom is true for $\bar{p}$ iff it is false for p. By this, there can be preconditions for negative atoms of p.

2.2 PDDL

PDDL [13] is a syntax that is used to describe classical planning problems. For each problem, there is a domain and an instance description. The domain contains the action schemas, and the instance contains the objects and the initial and goal situation.
PDDL descriptions are first-order symbolic representations. Therefore, predicates are parameterized by objects. These objects can have different types, and actions can be defined over specific types of objects.

```
(define (domain miconic)
        (:requirements :typing)
        (:types floor person)
        (:predicates    (lift_pos ?x -floor)  (in_lift ?x -person)
                        (in_floor ?x -person ?y -floor)
                        (above ?x -floor ?y -floor))
        (:action unboard
         :parameters    (?x -person ?y -floor)
         :precondition  (and (lift_pos ?y) (in_lift ?x))
         :effect        (and (in_floor ?x ?y) (not (in_lift ?x))))
        (:action board
         :parameters    (?x -person ?y -floor)
         :precondition  (and (lift_pos ?y) (in_floor ?x ?y))
         :effect        (and (in_lift ?x) (not (in_floor ?x ?y))))
        (:action move_up
         :parameters    (?x ?y -floor)
         :precondition  (and (lift_pos ?x) (above ?y ?x))
         :effect        (and (lift_pos ?y) (not (lift_pos ?x))))
        (:action move_down
         :parameters    (?x ?y -floor)
         :precondition  (and (lift_pos ?x) (above ?x ?y))
         :effect        (and (lift_pos ?y) (not (lift_pos ?x)))))
```

Listing 2.1: Miconic domain in PDDL syntax.

```
(define (problem miconic3x2)
        (:domain miconic)
        (:objects floor1 floor2 floor3 -floor
                  person1 person2 -person)
        (:init  (above floor2 floor1) (above floor3 floor2)
                (in_lift person1) (in_lift person2) (lift_pos floor1))
        (:goal  (and (in_floor person1 floor2)
                (in_floor person2 floor3))))
```

Listing 2.2: Miconic Instance in PDDL syntax.

An example domain description for *miconic* can be seen in Listing 2.1. In this domain, the *above* predicate is static since its value is not changed by an action. The types of objects are $floor$ and *person*.

Typing is an extension of PDDL, and it can be translated to classical PDDL by adding a static unary predicate for each type [12]. Then, instead of defining the type of variable in the parameter section, there will be a precondition on the corresponding predicate. Another extension of PDDL are negative preconditions [12]. By this, there can be negative preconditions without introducing a new predicate.

The instance contains the initial state, the goal states, and the objects. The objects contained in the *miconic* instance in Listing 2.2 are 3 floors and 2 persons. In the initial state, both persons are in the lift and the lift is in $floor1$. The goal is that *person*1 left the lift in $floor2$ and *person*2 in $floor3$.

3 Related Work

This thesis will focus on the area of lifted action model learning. This chapter will give an overview of current action model learning algorithms. In particular, the algorithm SIFT will be introduced, which will be the basis of this thesis. Also, it will be shown how classical planning problems depend on the initial state.

3.1 Action Model Learning

The goal of action model learning is to learn a symbolic action model [2]. In most approaches, the input to this learning task is based on traces. The action model can also be learned online [2].

In many approaches that try to learn an action schema, the states are (partially) observable [1, 2, 15]. That means that for a state that is reached via a trace, it is (partially) known which predicates are true. That also means that the predicates are already known to the algorithm. Therefore, a significant amount of domain knowledge must be available before learning. In this setting, learning boils down to finding the dynamics of the already known predicates. That means that it only needs to be determined which actions change which predicates and what preconditions are contained in the domain.

A family of algorithms that uses only a sequence of action labels to learn a domain is LOCM [5, 6, 10]. These algorithms first learn the types of the domain. Then, for each type, there will be an automaton build that describes the behavior of this type. For this, the automaton of a type contains a single transition for each action position $a(x_i)$ of an action $a(x_1, ..., x_n)$. The automaton is then modeled such that all action positions that are successive in the trace are also successive in the automaton. That means if there is an object applied first in position $a(x_i)$ and then in position $b(y_j)$, then these actions must be successive in the automaton.

An extension of this algorithm is LOCM2 [5], which learns multiple automata for each type. The goal of this algorithm is again to model all successive actions for a type in the automaton. The difference is that this algorithm also tries to model actions that are not successive in the traces. That means that the action positions $a(x_i)$ and $b(y_j)$ that are not applied successively in the trace for an object, should not be successive in at least one automaton. For both algorithms, LOCM and LOCM2, it is not studied under what conditions the algorithms are sound and complete. For the algorithm *LOCM2*, there is an additional step to learn the static predicates

of the domain. The is done by the algorithm *LOP*, which uses optimal goal-oriented plans to do so.

It is also possible to learn a domain when only action names and no objects are given [4, 14]. This can be done if the state graph of a problem is given, for which the transitions are only labeled with the corresponding action name. Then, by a combinatorial solver, objects and action schemas can be learned. Since the combinatorial approach needs to find the predicates, the groundings of the actions, and the effects, it is very expensive.

3.2 SIFT

Another approach that uses action traces without state observability is SIFT [9]. The difference from LOCM is that the approach has a soundness and completeness result. A key assumption of this approach is that the hidden domain $\mathcal{D}$, from which the traces are drawn, is well formed. That means that an action has for every effect a precondition on the negated effect. For this, it is assumed that the hidden domain is described in STRIPS with negation [12].

The input to the learning algorithm are traces from m problems $P_i = (\mathcal{D}, \mathcal{I}_i)$, $1 \leq i \leq m$ over the same hidden domain $\mathcal{D}$. Traces are sequences of grounded actions $a(o)$, where a is an action name, and $o = (o_1, ..., o_n)$ an object tuple. There is no information about the predicates of the domain $\mathcal{D}$, and the states are not observable. As input also graphs can be used, where the difference is that for graphs there exists state equality. Because of this, the state space of P or any partial graph of it can also be used as input. If there is state equality for the traces, they are called extended traces. For learning the domain, extended traces are not always needed.

The approach is based on the observation that a predicate in most cases is not changed by an action and all its arguments but only by a subset of its arguments. For example, in gripper $pick(ball, room, gripper)$ does not change a predicate of arity three but multiple predicates of lower arity. The action affects a predicate $carry(ball, gripper)$ and $in\_room(ball, room)$. This means that a predicate is changed by a tuple of arguments contained in the action grounding. To account for this, action patterns are introduced that bind an action name and a tuple of argument positions.

Definition 1 (taken from Gösgens *et al.* [9])**.** An *action pattern* $a[t]$ of arity k is an action name a of arity $k' \geq k$ followed by a tuple t of k different indexes $t = \langle t_1, \dots, t_k \rangle$, $1 \leq t_i \leq k'$, $i = 1, \dots, k$.

The action $pick(ball, room, gripper)$ has the positive effect $carry(ball, gripper)$, which corresponds to the action pattern $pick[1, 3]$. A negative effect of this action is $in\_romm(ball, room)$, which corresponds to the action pattern $pick[1, 2]$. The grounded actions that fit the action patterns $pick[1, 3]$ for a grounding $\langle ball_i, gripper_j \rangle$ are all actions that have the form $pick(ball_i, \_, gripper_j)$. In this case, "_" means that this position can take any object.

Action patterns are similar to effects in a PDDL domain. Now action patterns can be combined to form features, where features are similar to predicates in a PDDL domain.

Definition 2 (taken from Gösgens *et al.* [9])**.** A *feature* f of arity k is a pair $f = \langle k, B \rangle$, where B is a non-empty set of action patterns of arity k. B is called the *feature support*, also referred to as B_f.

Any feature $f = \langle k, B \rangle$ is an assumption that there exists a predicate p with arity k in $\mathcal{D}$ that is affected exactly by the action patterns in B. A feature f can be seen as a dual representation of a predicate p from a domain $\mathcal{D}$. In a PDDL domain, each action changes an arbitrary number of predicates. In the feature representation, each feature is affected by an arbitrary number of action patterns.

For example, in *gripper* there is the action $move(room_a, room_b)$ with effects on $at(room_a)$ and $at(room_b)$. This is the only action that changes the value of the predicate at. This predicate can be described as a feature $f = \langle 1, \{move[0], move[1]\} \rangle$.

A grounding $\langle o \rangle = \langle o_1, ..., o_k \rangle$ of a feature $f = \langle k, B \rangle$ is affected by all and only the actions that fit the grounding with an action pattern $a[p] \in B$. That means that a grounding $\langle o \rangle$ is affected by an action $a(o')$ with an action pattern $a[p]$ if $o_i = o'_{p_i}$ for $1 \leq i \leq k$.

Definition 3 (taken from Gösgens *et al.* [9])**.** The *action grounding* $A_f(o)$ of a feature f in a set of traces T refers to the set of ground actions $a(o')$ in T such that $a[t]$ is a pattern in B_f and $o = t[o']$.

This definition can be explained using the predicate *carry* from the domain *gripper*. This predicate is affected by the actions *pick* and *drop*. This predicate can be translated to a feature that is defined as $f_{carry} = \langle 2, \{pick[1,3], drop[1,3]\} \rangle$. The grounding $f_{carry}(ball_i, gripper_j)$ is affected by all actions that fit this grounding with any of the action patterns. For $pick[1,3]$, there is $pick(ball_i, \_, gripper_j) \in A^f(\langle ball_i, gripper_j \rangle)$. For the action pattern $drop[1,3]$, there is $drop(ball_i, \_, gripper_j) \in A^f(\langle ball_i, gripper_j \rangle)$. Here, the "_" can take any object that is contained in the domain.

To learn a domain $\mathcal{D}_L$ from traces T, a feature must be admissible for all groundings. For this, first all possible features f are created, and then features that do not fit the traces T are discarded. This can be done since there are only finitely many features. This is because there are only finitely many action names a and each action name has at most arity k and therefore only a limited number of action patterns for each arity. Therefore, there can only be finitely many subsets of action patterns for each arity $k' \leq k$ and by this only finitely many features f.

The filtering process is based on the assumption that the hidden domain $\mathcal{D}$ is well formed. In a well formed domain $\mathcal{D}$ whenever an effect is applied in a state s, in this state, the negated value of the changed atom has to be true. Because of this, only actions with opposing effects can be successive in the traces. Therefore, successive action patterns in the trace must have

different signs. Since each action must either be adding or deleting, the set of action patterns can be split into two sets of action patterns. One set contains adding actions, and the other set contains deleting actions.

In a feature, there is no assumption whether an action pattern is a positive or negative effect for the feature. For this, there will be a sign for each action pattern $a[p] \in B$. For an action pattern $a[p]$, there can be either $sign(a[p]) = 0$ or $sign(a[p]) = 1$. Here, $'0'$ can be interpreted as a negative effect and $'1'$ as a positive effect. Since STRIPS with negation is used as language, positive and negative effects are interchangeable. The signs will be obtained by checking which action patterns must have the same sign and which must have different sings. This must hold for every grounding $\langle o \rangle$ and for every trace.

Which action patterns must have the same sign can be seen in the traces. First, two action patterns that are consecutive in a trace with respect to a grounding $\langle o \rangle$ must have opposing signs. This holds since the hidden domain is well formed. Second, two action patterns must have the same sign if they reach the same state. To be more precise, if there are two paths leading to the same state, the last action patterns contained for a grounding $\langle o \rangle$ in the paths must have the same sign. This is because the value of a feature depends on the last action that changed it and each grounding must be either true or false in each state. The same holds for two paths that originate from the same node. In this case, the first action patterns in the paths must have the same sign.

To gather the pattern constraint, a 0/1-coloring is used, which is not explained further in the paper. This algorithm will be explained in detail in Section 4. The runtime of the coloring algorithm is linear in the number of transitions contained in the graph. The coloring must be done for a feature for each grounding. To reduce the runtime of the coloring algorithm, the size of the graph can be reduced by merging edges in the graph that do not affect the grounding for a set of features. That means edges on which do not contain an action pattern in this set of features, can be merged. This is because the connected states must have the same value for every grounding $\langle o \rangle$. This significantly reduces the number of nodes and transitions in the graph and, therefore, reduces the runtime of the coloring algorithm.

From the obtained constraints, the signs of the action patterns can be determined. This can be done by finding a sign for each action pattern such that all constraints are satisfied. That means that there is a set of deleting and adding action patterns. For this, each pair of successive action patterns can not be contained in the same set, and each pair of action patterns that form a fork must be contained in the same set. If this is possible, a feature is consistent with the input traces T. Checking if the constraints are satisfiable is a computationally easy problem and can be reduced to 2-CNF satisfiability. The set $F(T)$ then contains all the consistent features for the traces T.

With the consistent features $F(T)$, the domain $\mathcal{D}_L$ can be defined. For this, there are preconditions for the consistent features $F(T)$ defined. Preconditions are defined by action patterns that have the same arity as the features. For a feature $f = \langle n, B \rangle$ any action pattern $a[p]$ of arity n can be a precondition if $a[p] \notin B$. Action patterns $a[p] \in B$ are already preconditions, because the negated effect must hold in any state where an action with this action pattern is applied.

The action pattern $a[p]$ is an admissible positive precondition for the feature f if in any state, where an action pattern action $a(c)$ is applied, the precondition is fulfilled. Therefore, for the corresponding grounding $\langle o \rangle$ of the action pattern $a[p]$ and the action $a(c)$, either $f(\langle o \rangle) = 1$ in s or the grounding $\langle o \rangle$ had no value assigned in the trace. A positive precondition is only added to a feature if it is fulfilled at least once, so $f(\langle o \rangle) = 1$ for a state where $a(c)$ is applied and $\langle o \rangle$ is the grounding for the action pattern $a[p]$. The same is done for negative preconditions, where instead of $f(\langle o \rangle) = 1$ here $f(\langle o \rangle) = 0$ must hold.

In the learned domain $\mathcal{D}_L$ for every predicate $p \in \mathcal{D}$ of a well formed domain there is a corresponding feature f. For this feature f, one can show that it is admissible for every set of traces T that is drawn from $\mathcal{D}$. The intuition behind this is that the effect corresponding to the action pattern $a[p]$ in $\mathcal{D}$ is either adding or deleting the predicate. From this, signs for the action patterns can be obtained. That means an action pattern that adds the predicate gets sign 1 and an action pattern that deletes the predicate gets sign 0. By definition, this feature needs to be valid for these constraints, since in a well defined formed only effects with opposing effects are successive. Also, only action patterns with the same sign can reach the same state or originate from the same state.

The algorithm has a soundness and completeness result. For this, the traces must set the value of each grounded atom in the domain $\mathcal{D}$. To achieve this, it is assumed that each grounded atom is at least affected once and that the traces are connected. Soundness in this case means that all traces that are contained in the input are applicable in the learned domain. Completeness means that if the complete traces reach a state where an action is not applicable, they are also not applicable in the learned domain.

The problem of generalization lies on the one hand in filtering all features f that are not contained in $\mathcal{D}$ and on the other hand in filtering all non-valid preconditions of $\mathcal{D}_L$. As seen before, for any predicate of the domain, there will be a corresponding feature that is admissible for all traces. This does not mean that every learned feature or every learned precondition must be contained in the domain $\mathcal{D}$. This is because these predicates and preconditions can be redundant to the already contained predicates and preconditions. Therefore, these predicates and preconditions would make the domain $\mathcal{D}$ larger without giving any additional information or restriction to $\mathcal{D}$.

In *gripper*, for example, an additional unary feature f is learned that states whether a ball is grabbed or not. This feature is not contained in $\mathcal{D}$, but it can be added to it without changing

the domain. Then the new (additional) predicate is true if the ball is in a gripper and false if the ball is in a room. The predicate is not wrong, but does not provide new information, as the predicate *carry* already contains it.

This type of features can and will be learned and will be part of $\mathcal{D}_L$. It is possible that for a domain $\mathcal{D}$ there exist multiple domains $\mathcal{D}'$ that generate the same state space. These domains can be combined into a domain $\mathcal{D}_{max}$ that is equivalent to $\mathcal{D}$ and contains all possible features for this domain.

The two main steps of the algorithm are the generation of features and the filtering of features. To reduce the number of generated features, there are types learned. This is done using an algorithm introduced by LOCM [6]. This algorithm obtains for each position x_i of an action $a(x_1, ..., x_n)$ a type. Since action patterns are defined over positions of actions, now each action pattern also has a type tuple. Then features can only contain action patterns with the same type tuples.

The experimental results in the paper showed that for domains that fulfil the well formed assumption, 100% of the learned domains have been verified, with exceptions that were explained with a bad sampling strategy. The learned domains were tested for a set of positive and negative examples from a larger instance of the same domain. For negative samples, an additional action is added to a state where it is not applicable in the original domain. The verification is successful if any positive sample is applicable in the learned domain and any negative sample is not applicable. In the results, it was also shown that the algorithm can learn from instances with hundreds of thousands of states and transitions.

3.3 Valid PDDL States

It has been shown that a PDDL domain does not fully define a planning problem, but only defines its dynamics [11]. This is because domains only describe which predicates exist and how actions change these predicates. It is also defined under which conditions an action is applicable. Which actions are applicable and which predicates are true in a state depends on the initial situation of the problem. The initial situation contains only a set of predicates and does not have any restrictions on that set. Actually, for most planning problems, there are restrictions on the initial situation of a planning problem. These restrictions are not given by a PDDL domain, but are in most cases defined by some language description of a domain. Therefore, there can be valid and invalid initial situations. From valid initial situations, only valid situations can be reached. The authors showed that these restrictions can be added to PDDL domains with first-order logic sentences.

As an example, the blocks world domain can be used [11]. For this domain in an initial state, there are restrictions that are not explicitly stated. For example, a block needs to be placed

in exactly one position. That means it is either stacked onto a different block or on the table. Another restriction is that it cannot hold that for blocks b_1, b_2, there is b_1 stacked on b_2 and at the same time the block b_2 is stacked on b_1. Both of the behaviors can be stated in an initial situation, but the corresponding instance does not describe a blocks world instance as it is intended. Therefore, these instances are invalid.

4 Sift Coloring Algorithm

The SIFT [9] algorithm checks whether a feature is admissible by checking which action patterns can be applied successively and which can be applied in parallel. This can be done since the domain is assumed to be well formed. Therefore, adding and deleting actions for a feature must be alternating, so successive action patterns must have opposing effects. Since a predicate needs to be either true or false in each state, action patterns that reach the same state either both add, respectively, delete the grounding.

In [9] it was mentioned that the constraints can be obtained using a 0/1-coloring. In the coloring, the color 0 states that the feature is $false$ for a grounding $\langle o \rangle$ and the value 1 states that the *grounding* is true. Since the input language is STRIPS with negation, these values are interchangeable. The coloring must be done for a feature $f = \langle n, B \rangle$ for every possible grounding $\langle o \rangle$. For a feature f, it is known that an action grounding $a(x) \in A^f(\langle o \rangle)$ needs to change the value of the predicate for the grounding $\langle o \rangle$. Therefore, two states connected by a transition labeled with $a(x)$ must have different colors for the grounding $\langle o \rangle$.

The coloring is done for each grounding $\langle o \rangle$ separately. The algorithm starts for grounding $\langle o \rangle$ by labeling an arbitrary state s with value 0, so there is $\chi_{\langle o \rangle}(s) = 0$. Then the color of a state is propagated to all states to which it is connected. If this transition is labeled with an action $a(x) \in A_f(\langle o \rangle)$, then the states must be colored differently. If the transition is labeled with an action $a(x) \notin A_f(\langle o \rangle)$, the states must have the same color. This can be defined by the following propagation rule.

Definition 4 (Propagation Rule)**.** Let there be a grounding $\langle o \rangle$ for the feature $f = \langle n, B \rangle$. Let there be a state s for which its color will be propagated to its neighbors. If there is a transition (s, s') or (s', s) that is labelled with an action grounding $a(x)$, then $\chi_{\langle o \rangle}(s') \neq \chi_{\langle o \rangle}(s)$ if $a(x) \in A_f(\langle o \rangle)$ and $\chi_{\langle o \rangle}(s') = \chi_{\langle o \rangle}(s)$ if $a(x) \notin A_f(\langle o \rangle)$.

For every action that changes the coloring, there needs to be a corresponding action pattern. Since every effect is either adding or deleting, this also needs to hold for action patterns. Therefore, the action patterns are also colored. The action pattern is colored with the color of the reached state.

Definition 5 (Pattern Color)**.** For a grounding $\langle o \rangle$, let there be a transition (s, s') that is labeled with an action $a(c) \in A_f(\langle o \rangle)$. Then there exists an action pattern $a[p_1, ..., p_n] \in B$ such that $c_{p_i} = o_i$ for $1 \leq i \leq n$. Then the action pattern $a[p_1, ..., p_n]$ gets the same color as the state s', so $\chi_{\langle o \rangle}(a[p_1, ..., p_n]) = \chi_{\langle o \rangle}(s')$.

Then for the obtained coloring, there must hold two conditions. First, since a predicate needs to have a unique value in each state, each node needs to have exactly one color. Second, each action pattern is either adding or deleting, therefore each action pattern can be colored with at most one color. At most one, since an action pattern may not be contained in a trace.

Definition 6 (Admissible Coloring)**.** A coloring is admissible if for every state s there is either $\chi_{\langle o \rangle}(s) = 0$ or $\chi_{\langle o \rangle}(s) = 1$. For every action pattern $a[p]$ that is contained in the trace, there is can not be $\chi_{\langle o \rangle}(a[p]) = 0$ and $\chi_{\langle o \rangle}(a[p]) = 1$.

In the following, it is shown that a coloring is admissible iff the pattern constraints of the feature are consistent for this grounding. For this we proof that if the coloring is not admissible, there will be conflicting constraints.

Lemma 1. *If there are 3 action patterns on an undirected path, then the first and the third have the same sign iff they have the same direction. For this patterns there exist the corresponding constraints.*

Proof. Let there be action patterns $a[p_a]$, $b[p_b]$, and $c[p_c]$. There is an undirected path from $a[p_a]$ to $b[p_b]$ that does not contain any other action pattern. The same holds for $b[p_b]$ and $c[p_c]$. Two connected action patterns are successive if they have the same direction and they form a fork if they have different directions.

Now, let $a[p_a]$ and $c[p_c]$ have the same direction on the path from $a[p_a]$ to $c[p_c]$. Then, if $b[p_b]$ also has the same direction, there is $sign(a[p_a]) \neq sign(b[p_b])$ and $sign(b[p_b]) \neq sign(c[p_c])$ and therefore $sign(a[p_a]) = sign(c[p_c])$. If $b[p_b]$ has a different direction than the other patterns, there is $sign(a[p_a]) = sign(b[p_b])$ and $sign(b[p_b]) = sign(c[p_c])$ and, therefore, there is $sign(a[p_a]) = sign(c[p_c])$.

Let $a[p_a]$ and $c[p_c]$ have different directions on the undirected path from $a[p_a]$ to $c[p_c]$. Then $b[p_b]$ can have the same direction as $a[p_a]$, then there is $sign(a[p_a]) \neq sign(b[p_b])$ and $sign(b[p_b]) = sign(c[p_c])$ and therefore $sign(a[p_a]) \neq sign(c[p_c])$. If $b[p_b]$ has the same direction as $c[p_c]$ and there is $sign(a[p_a]) = sign(b[p_b])$ and $sign(b[p_b]) \neq sign(c[p_c])$ and, therefore, $sign(a[p_a]) \neq sign(c[p_c])$. □

Lemma 2. *Let there be an undirected path on which there are $2n + 1$ action patterns. Then the first and the last pattern have the same sign if they have the same direction and the corresponding constraints exist.*

Proof. By Lemma 1, this holds for $n = 1$. For $n > 1$, we prove this by induction from $n \to n + 1$. In the following a_i will refer to an action pattern $a[p]$.

If a_{2n-1} has the same direction as a_1 there is $sign(a_1) = sign(a_{2n-1})$ by induction hypothesis. Now if a_{2n+1} has the same direction as a_{2n-1} there is by Lemma 1 $sign(a_{2n-1}) = sign(a_{2n+1})$

and, therefore, $sign(a_{2n+1}) = sign(a_1)$. If a_{2n+1} has a different direction than a_{2n-1} then there is by Lemma 1 $sign(a_{2n-1}) \neq sign(a_{2n+1})$ and, therefore, there is $sign(a_{2n+1}) \neq sign(a_1)$.

If a_{2n-1} has a different direction than a_1 there is $sign(a_1) \neq sign(a_{2n-1})$ by induction hypothesis. Now if a_{2n+1} has the same direction as a_{2n-1} (a different direction as a_1), there is by Lemma 1 $sign(a_{2n-1}) = sign(a_{2n+1})$ and, therefore, $sign(a_{2n+1}) \neq sign(a_1)$. If a_{2n+1} has a different direction than a_{2n-1} (the same direction as a_1), there is by Lemma 1 $sign(a_{2n-1}) \neq sign(a_{2n+1})$ and, therefore, $sign(a_{2n+1}) = sign(a_1)$. □

Now we can show that there exist conflicting constraints if a coloring is not admissible. This needs to be shown on the one hand for a state that is colored with two colors, and on the other hand for an action pattern that gets colored with two colors.

Lemma 3. *If a state s is colored with both colors, then there exists a conflicting constraint.*

Proof. Let there be a state s that is labeled with both colors. Then this state must be reachable via 2 paths from the initial state s_{init}, from which the coloring was started. Since the paths provide different colors, one path needs to contain an uneven number of actions that change the value of the grounding, and the other path needs to have an even number of changes. Therefore, there exists an undirected cycle in the trace that contains $2n-1$ action patterns. Therefore, there can be an undirected path that contains $2n$ action patterns where the first and last action pattern are labeled with the same action pattern $a[p]$. These two action patterns must have the same direction. By Lemma 2 the action pattern $a[p]$ at position 1 has the same sign as the pattern $b[p']$ at position $2n-1$ iff they have the same direction. The pattern $b[p']$ at position $2n-1$ has the same sign as the pattern as the action pattern $a[p]$ at position $2n$ iff they have a different direction. Since the action patterns $a[p]$ at the positions 1 and $2n$ have the same direction, there exist constraints $sign(a[p]) \neq sign(b[p'])$ and $sign(a[p]) = sign(b[p'])$. Therefore, there exist conflicting constraints. □

Lemma 4. *If there is an action pattern $a[p]$ that is colored with both colors, there is a conflicting constraint.*

Proof. Let there be an action pattern $a[p]$, which is colored with both colors. Then there exist undirected paths from the start to states s, s' where the stats s and s' are reached by an action with the action pattern $a[p]$. The states s and s' are colored differently, since the action pattern is colored with two colors. Therefore, one of these paths has an even number of action patterns that change the value of the feature for the grounding, and the other has an uneven number of changes. These two paths can be combined to a path that contains an uneven number of changes, which starts at s and ends at s'. In both states, an action with the same action pattern is applied. Therefore, these actions have a different direction and are on a path that contains an uneven number of action patterns. Because of this, the corresponding action patterns need

to have a different sign by Lemma 2. Since at both ends of the undirected path there is the same action pattern, there is the constraint $sign(a[p]) \neq sign(a[p])$. □

Theorem 1. *Iff the action patterns can be split into two sets without a conflict, then there is an admissible coloring.*

Proof. "⇒" Let there be a admissible split but a non-admissible coloring. Then either an action pattern is colored with two colors or a state gets two colors. By Lemma 3 and Lemma 4 this is not possible.

"⇐" Let there be a valid coloring but a conflict for the constraint satisfiability. Since the coloring is admissible, the action patterns can be split into sets without a conflict. Therefore, there needs to be a constraint that was not covered by the coloring. This is not possible since every pair of successive action patterns must have different signs in the coloring. Also, every pair of action patterns that reach or come from the same state must have the same sign. Therefore, it is not possible that a constraint was not covered by the coloring. This is a contradiction. □

Now, a lemma will be introduced that will be needed later. For a predicate p there can be a feature f_p that is valid for all traces drawn from the corresponding domain [9]. We show that for a predicate p the color of a grounding $\langle o \rangle$ in a state s will be equivalent to the value of $p(\langle o \rangle)$ in a state s.

Lemma 5. *Let there be a predicate p and the corresponding domain feature f_p. Then for a grounding $p(\langle o \rangle)$ the coloring of a state s where $p(\langle o \rangle)$ is true is either always 1, or always 0. The same holds for states where the grounding is false.*

Proof. By definition of the domain feature f_p, the feature has an effect on a grounding $\langle o \rangle$ iff the grounding $p(\langle o \rangle)$ is affected by an action [9]. We assume, that the coloring is always initialised with 0 for a random state s_{init}. Now the atom $p(\langle o \rangle)$ can either be true or false in the initial state s_{init}. By definition, an action pattern effects a grounding iff the corresponding action has an effect for on the predicate with the same grounding. We know that the value of the predicate needs to be change from true to false and vice versa. For a feature the coloring needs to be changed from color 1 to 0 and vice versa. Therefore, on any path the value of $p(\langle o \rangle)$ is change iff the color $c(\langle o \rangle)$ is changed. If $p(\langle o \rangle)$ is true in the initial state there is $\chi_{\langle o \rangle}(s) = 0$ iff $p(\langle o \rangle)$ is true in s and $\chi_{\langle o \rangle}(s) = 1$ iff $p(\langle o \rangle)$ is false in s. Similar for the case where $p(\langle o \rangle)$ is false in the initial state. □

5 Sift Preprocessing for Non-STRIPS Labels

The goal of this section is to remove the dependence of SIFT [9] on action labels that have the form of STRIPS [7] action groundings. The proposed algorithm is a preprocessing of the traces that are used for SIFT to learn the domain. The goal of this preprocessing is to find the missing arguments and to add them to the action labels. We will show that it is possible to remove arguments from a STRIPS action schema, which can later be recovered.

As in SIFT, we assume that the domain is well formed. That means that an action has, for each effect, a precondition on the inverted effect.

Assumption 1 (Well formed)**.** The input domain is well formed. Therefore, an action is only applicable if, for all of its effects, the negated effect is true.

Currently, for SIFT to learn a domain, the traces must be labeled with STRIPS-like action labels. In a STRIPS-like action label, all objects that are contained in an atom for which the value is changed must be given explicitly. Also, all objects that are contained in a precondition must be given. This is not always necessary, since some objects can be inferred from the context of an action. The context of an action are all paths that lead to the state where the action is applied.

An example of this can be found in the $ferry$ domain, which contains the $at(?car, ?location)$ predicate. In each state, a car can be in at most one location. Therefore, the location of a car can be concluded without stating the location explicitly, if the location that was added is known. In a STRIPS-like action label, the car and the position must be given when the grounding is deleted, or when an action has a precondition on it. Later, it will be shown that these types of arguments, under some conditions, do not need to be contained in the input.

As one can see, this observation depends on the given problem. Especially, it depends on the initial situation. This is because in the initial situation there can be a car in multiple locations. That initial situation does not describe the problem as intended. In the following, we assume that the instances fulfill the restrictions of the domain.

In the previous example, one can see that the location can be reconstructed since the car can be in at most one position. Therefore, the full grounding is identified by a partial grounding. This is only possible if for every partial grounding there is at most one full grounding. If this is not the case, based on the partial grounding, it is not clear which full grounding needs to be reconstructed.

Here, again the predicate $at(?car, ?location)$ from the ferry domain can be used as an example. Two cars can be in the same location, that is, $at(car_1, location_1)$ and $at(car_2, location_1)$ can be

true in the same state s. When the action *board* removes a car from this location, it is not clear which car is boarded when only the location is given and, therefore, it is not clear which grounding of *at* is deleted.

In STRIPS with negation, it is possible to switch all effects and preconditions of a predicate without changing the domain. This can be done for the predicate *at* in the $ferry$ domain. As a result, there is at most one grounding $at(?car, ?location)$ false in every state for each *car* in the instance.

In a state s, one can see whether an $identifying$ relation is built by adding or deleting actions. This is because the number of groundings of the relation, which are currently not in a relation, is asymptotically higher. This can be demonstrated using the $ferry$ domain. Let there be a car car_1 and locations $location_1, location_2$, and $location_3$. Then in each state, the car can be in at most 1 location, assume that $at(car_1, location_1)$ is true. Therefore, the predicate must be false for all other groundings, therefore $at(car_1, location_2)$ and $at(car_1, location_3)$ are not true in the state. As a result, there are always more groundings false than true in a state. The same can be seen when all effects of the action are negated. In the following, we will assume w.l.o.g. that a predicate, for which arguments can be recovered, is defined in such a way that there can be at most one grounding true per state.

Assumption 2. A relation that is used to $identify$ arguments is always added by adding actions of the domain and deleted by deleting actions.

With this assumption $identifying$ predicates can be defined. These are predicates in a STRIPS domain that fulfill the conditions described above.

Definition 7 (Identifying Predicate)**.** Let there be a predicate $p(x_1, ..., x_n)$ and a subset of positions $I = \{i_1, ..., i_m\} \subset \{1, ..., n\}$. Let $g = \langle g_1, ..., g_n \rangle$ be a full grounding for the predicate and $g^I = \langle g'_1, ..., g'_n \rangle$ its partial grounding for positions I, where $g'_j = g_j$ if $j \in I$ and $g'_j = \_$ if $j \notin I$. If for the positions I, for every partial grounding g_{id}, there exists in each state at most one grounding g such that $g^I = g_{id}$, then the predicate is $identifying$ with $identifying$ positions I. The set $Is_p(p)$ contains all possible sets of identifying positions I for a predicate p.

The predicate $at(?car, ?location)$ in the $ferry$ domain can serve as an example of this definition. We know that for an object car_1 there can only be one grounding $g = \langle car_1, location_i \rangle$ true in each state. That means that for the set of identifying positions $I = \{1\}$, there is a partial grounding $g^{\{1\}} = \langle car_1, \_ \rangle$. For a partial grounding $\langle car_1, \_ \rangle$, there can only be one full grounding with this partial grounding true in each state. Therefore, the full grounding $\langle car_1, location_i \rangle$ can be identified by the partial grounding $g^{\{1\}} = \langle car_1, \_ \rangle$. The positions that can be identified by the partial grounding will be called in the following $identified$ positions.

For identifying predicates, there can be arguments removed. This is because, after adding the identifying predicate for a full grounding, this full grounding can already be $identified$ by a

partial grounding. Therefore, when deleting this grounding or when there is a precondition on it, only the partial grounding needs to be given. That means that for a deleting action, a set of arguments needs to be given from which the grounding is identified. Searching for missing arguments is similar to searching for features in the SIFT algorithm. The difference is that, for deleting effects, there are arguments missing.

Now we want to show which groundings can be removed from an action schema and then recovered afterwards. These are the arguments that are contained in deleting effects or positive preconditions of the *identifying* predicates. This is because these are the positions that argue about a grounding that is true. For these groundings, there can only be one full grounding for a partial grounding with these *identifying* positions. Therefore, all arguments for adding action patterns for an *identifying* feature must be included in the input. For every deleting action and positive precondition, there only needs to be a set of arguments that identifies the full grounding.

Based on this, it can be defined which arguments of STRIPS-like labels are necessary to find all *identifying* predicates.

Definition 8 (Necessary Arguments)**.** Let there be an action $a(x_1, ..., x_n)$. Any argument x_i that is contained in an adding effect for an *identifying* predicate is necessary. For every negative effect or positive precondition on an *identifying* predicate, there needs to be a set of *identifying* positions. Every argument x_i that is contained in an *identifying* position of a deleting effect or positive precondition is necessary.

Now, based on the necessary arguments, also unnecessary arguments can be defined. These are the arguments that can later be recovered if all necessary arguments are contained in the input.

Definition 9 (Unnecessary Arguments)**.** Let there be an action $a(x_1, ..., x_n)$ and a set of necessary arguments. For each negative effect and positive precondition on an *identifying* predicate, an argument x_i is unnecessary if it is not necessary. That means it is contained in an *identified* position of a precondition or deleting effect and not in an adding effect or *identifying* position of a predicate.

To show which arguments can be removed from an action, we will use the blocks world domain with 3 actions. First, we know that there is only one predicate of arity 2 and all other predicates have arity 1. Predicates of arity 1 cannot have both *identifying* and *identified* positions and are therefore not relevant. The only binary predicate is the predicate *on*, which is identifying for $I = \{1\}$ and $I' = \{2\}$. This is because for each block there can be at most one block above it and there can be at most one block below it.

For the action $stack(x, y)$ there is a positive effect $on(x, y)$ which contains both arguments, therefore, both arguments are necessary. The action $unstack(x, y)$ has a deleting effect $on(x, y)$,

therefore, both sets of *identifying* positions can be used to identify the full grounding. That means either x is necessary and y is unnecessary, or vice versa. For the action $move(x, y, z)$ there is a positive effect $on(x, z)$, therefore, these arguments are necessary. This action also has a negative effect $on(x, y)$. When using the set of *identifying* positions $I = \{1\}$, the argument y is only contained in an *identified* position and therefore unnecessary. For $I = \{2\}$, the argument y is used to identify the full grounding and is therefore necessary. The argument x is contained in an *identified* position but is necessary for an adding effect and therefore cannot be unnecessary.

Therefore, a possible action schema is $stack(x, y)$, $unstack(\cancel{x}, y)$, $move(x, \cancel{y}, z)$. Here, the crossed-out arguments refer to arguments that are unnecessary. The other possibility is the action schema $stack(x, y)$, $unstack(x, \cancel{y})$, $move(x, \cancel{y}, z)$.

Now, it is defined which arguments need to be contained in the input to find all *identifying* predicates. Therefore, the input of the algorithm can be defined. Since the proposed algorithm is a preprocessing step for the SIFT algorithm, the input will be traces. These are defined in the same way as in SIFT [9]. Here we assume that the traces are extended, which means that there are state equalities. Later, we will show that extended traces are more informative than non-extended traces. From now on, we will refer to extended traces as traces, making it explicit when talking about non-extended traces. From the traces, unnecessary arguments can be removed.

Definition 10 (Input)**.** Let there be an extended trace $T(P)$ which has been sampled from a problem $P = \langle \mathcal{D}, \mathcal{I} \rangle$. For each action schema $a(x_1, ..., x_n)$, let $r = \left(x_{r_1}, ..., x_{r_n}\right) \subset \{x_1, ..., x_n\}$ be a set of unnecessary arguments that are removed. Now the input to the algorithm is an extended trace $T'(P)$ such that 1) $T'(P)$ has the same states and transitions as $T(P)$ and 2) for each action grounding $a(c) \in T(P)$ there exists $a(c') \in T'(P)$ where c' contains an argument c_i when $x_i \notin r$.

As an example, we can again use the blocks world domain. We use the action schema $stack(x, y)$, $unstack(\cancel{x}, y)$, $move(x, \cancel{y}, z)$, where the crossed-out arguments are removed. Now, there is some hidden trace $T(P)$ that was sampled from a problem P, which represents the block world domain with a specific instance.

$$T(P) = stack(b_1, b_2), move(b_1, b_2, b_3), stack(b_2, b_4), unstack(b_1, b_3)$$

From this trace, the unnecessary arguments are removed, resulting in the trace

$$T'(P) = stack(b_1, b_2), move(b_1, b_3), stack(b_2, b_4), unstack(b_3).$$

This trace will be used in the preprocessing algorithm.

5.1 Identifier Features

To recover the removed arguments, features that are similar to those used in SIFT [9] will be introduced. In SIFT, the features consist of a single set of action patterns. The algorithm later determines for each action *pattern* a sign. For the $identifier$ features used in this approach, it will be known which actions add the feature. This is because for an adding action, all arguments must be known since they are necessary. For the deleting effects, there are potentially arguments removed, since they are unnecessary. For this, *partial* patterns are introduced, which are similar to the *action* patterns in SIFT [9].

Definition 11 (Partial Patterns)**.** A *partial* pattern $a\langle p\rangle$ is a combination of an action name a that has arity k together with a pattern $p = \langle t_1, ..., t_l\rangle$ with $p_i \in \{1, ..., k\} \cup \{\_\}$. There must be at least one "_" in each pattern p. For a pattern p, its identifying positions are exactly the positions that are not "_", so $Ip_p(p) = \{j | p_j \neq \_ \text{ for } 1 \leq j \leq l\}$.

In this case "_" is a "don't know", which means that the arguments of this position of the feature are not known. Positions that do not contain "_" are used to identify the arguments at the "_" positions. For a partial pattern and an action grounding, the partial grounding can now be defined.

Definition 12 (Partial Grounding)**.** For an partial pattern $a\langle p_1, ..., p_n\rangle$ and action grounding $a(c_1, ..., c_m)$ the partial grounding is $g_{id} = \langle c'_1, ..., c'_n\rangle$ such that for $1 \leq j \leq n$ there is $c'_j = c_{p_j}$ if $p_j \neq$ "_" else $c'_j =$ "_". A partial grounding has the same $identifying$ positions as the corresponding partial pattern $a\langle p\rangle$, so $Ip_g(g_{id}) = \{j | c'_j \neq \_ \text{ for } 1 \leq j \leq n\}$

Let there be for example an action grounding $a(o_1, o_2, o_3)$ with partial pattern $p[3, \_, 2]$. Then the partial grounding is $g_{id} = \langle o_3, \_, o_2\rangle$ and $Ip_g(g_{id}) = \{1, 3\}$. For the partial pattern $p[1, \_, 3]$, the partial grounding is $\langle o_1, \_, o_3\rangle$. For the action grounding $a(o_3, o_4, o_2)$ with partial pattern $p[1, \_, 3]$, the partial grounding is $\langle o_3, \_, o_2\rangle$.

Definition 13 (Fitting Groundings)**.** A partial grounding $g_{id} = \langle g'_1, ..., g'_n\rangle$ and a full grounding $g = \langle g_1, ..., g_n\rangle$ are fitting if for $1 \leq j \leq n$ there is either $g_j = g'_j$ or there is $g'_j = \_$.

That means a partial grounding and a full grounding are fitting if they contain at all positions, where the partial grounding has no blank, the same objects. That means for a full grounding $g = \langle object_1, object_2, object_3\rangle$ for example, the partial groundings $\langle object_1, \_, \_\rangle$, $\langle \_, \_, object_3\rangle$, $\langle \_, object_2, object_3\rangle$ or $\langle object_1, \_, object_3\rangle$ fit the full grounding.

As mentioned above, an $identifier$ feature contains two separate sets of patterns. The first set of patterns contains the patterns defined in SIFT, which in the following will be called $full$ patterns. The second set of patterns, used to delete the feature, will contain only *partial* patterns. The *partial* patterns will be used to delete a feature, where the non-blank positions are an assumption that these positions identify the complete grounding.

Definition 14 (Identifier Feature)**.** An identifier feature $f = (n, \mathbf{A}, \mathbf{D})$ contains the arity n, a set of $full$ patterns $\mathbf{A}$ and a set of $partial$ patterns $\mathbf{D}$. For every $a[p] \in \mathbf{A}$ and $b\langle p' \rangle \in \mathbf{D}$ the pattern p/p' must have length n. The $full$ patterns $a[p] \in \mathbf{A}$ add the feature, and the $partial$ patterns $b\langle p' \rangle \in \mathbf{D}$ delete the feature.

Now we describe a possible $identifier$ feature. We assume that we have the blocks world domain, which includes the action schema $stack(x, y)$, $unstack(x, y)$, and $move(x, y, z)$. As a result, there are actions $stack(x, y)$, $unstack(x)$, and $move(x, z)$. For this action schema, a feature $f_1 = (2, \mathbf{A}_1, \mathbf{D}_1)$ can be defined by the sets $\mathbf{A}_1 = \{stack[1, 2]\}$ and $\mathbf{D}_1 = \{unstack[1, \_]\}$. This feature is an assumption that there is an $identifying$ predicate in the hidden domain where $stack$ builds a relation, which is only removed by $unstack$ and can be identified by the first position.

For an $identifier$ feature, there are again $identifying$ positions that are similar to the $identifying$ positions of a predicate p.

Definition 15 (Identifying Positions)**.** If a feature $f = (n, \mathbf{A}, \mathbf{D})$ contains a $partial$ pattern $a\langle p \rangle \in \mathbf{D}$, then the positions $Ip_p(p)$ identify this feature. The set $Is_f(f)$ contains all sets of identifying positions for the feature f, that means $Ip_p(p) \in Is_f(f)$ for every $partial$ pattern $a\langle p \rangle \in \mathbf{D}$.

For the feature f_1, given above as an example, there is $Is_f(f_1) = \{\{1\}\}$. It is also possible to define a feature $f_2 = \langle n, \mathbf{A}_2, \mathbf{D}_2 \rangle$ with $\mathbf{A}_2 = \{stack[1, 2]\}$ and $\mathbf{D}_2 = \{unstack[1, \_], move[\_, 2]\}$. For this feature, the set of all $identifying$ positions is $Id_f(f_2) = \{\{1\}, \{2\}\}$.

For each identifying feature, there is a set that contains all possible partial groundings. This set contains all partial groundings for the $identifying$ positions of the feature.

Definition 16 (Possible Partial Groundings)**.** Let there be an $identifying$ feature $f = (n, \mathbf{A}, \mathbf{D})$ with sets of identifying positions $Is_f(f)$. Now, let G be the set of all possible groundings of arity n. Then for every set of identifying positions $I \in Is_f(f)$ and every possible grounding $g \in G$ the corresponding partial grounding g^I is a possible partial grounding $g^I \in IG(f)$. The set $IG(f)$ is the set of all partial groundings of the feature f.

Now, let there be objects $block_1$, $block_2$, $block_3$ where any combination is a possible grounding of arity 2. For the feature f_1 there is $Is_f(f') = \{\{1\}\}$, then for the grounding $g = \langle block_1, block_2 \rangle$ there is $g^{\{1\}} = \langle block_1, \_ \rangle \in IG(f)$. Therefore, the set of all partial groundings is $IG(f_1) = \{\langle block_1, \_ \rangle, \langle block_2, \_ \rangle, \langle block_3, \_ \rangle\}$. For the feature f_2, there is $Is_f(f') = \{\{1\}, \{2\}\}$. Then for $\langle block_1, block_2 \rangle$ there is $\langle block_1, block_2 \rangle^{\{1\}} = \langle block_1, \_ \rangle$ and $\langle block_1, block_2 \rangle^{\{2\}} = \langle \_, block_2 \rangle$. Because of this, there is $\langle block_i, \_ \rangle \in IG(f_2)$ and also $\langle \_, block_i \rangle \in IG(f_2)$ for $i \in \{1, 2, 3\}$.

The goal is now to find for each action grounding $a(c)$ for $partial$ pattern $a\langle p \rangle$ the grounding g that is deleted. From this grounding, the removed arguments can be recovered. To do this, we first remember which grounding g is added by an action a with a $full$ pattern $a[p]$. For

an action $a(c_1, ..., c_n)$ with a $full$ pattern $a[p_1, ..., p_m]$ then the grounding $\langle c_{p_1}, ..., c_{p_m} \rangle$ is added [9].

When defining an *identifier* feature f, it is already known which actions add the feature for a grounding c. Therefore, it is possible to define for each partial grounding g_{id} a set of actions that add a grounding c that fits this partial grounding.

Definition 17 (Added Partial Groundings)**.** Let there be a feature f with $Is_f(f)$. Now, let there be a partial grounding $g_{id} = \langle g_1, ..., g_m \rangle$ with identifying positions $Ip_g(g_{id})$. Then the set $Add^f(g_{id})$ contains all actions that add a grounding g such that $g^{Ip_g(g_{id})} = g_{id}$.

Let there be a feature f and the set $Add^f(\langle object_1, \_ \rangle)$. The partial grounding $\langle object_1, \_ \rangle$ has the set of identifying positions $I = \{1\}$. The set $Add^f(\langle object_1, \_ \rangle)$ contains all actions that add a grounding where the partial grounding fits $\langle object_1, \_ \rangle$. An example is the action $a(object_4, object_1)$ with the action pattern $a[2, 1]$. For this, the grounding $g = \langle object_1, object_4 \rangle$ is added for the set of identifying positions $I = \{1\}$ and there is $g^{\{1\}} = \langle object_1, \_ \rangle$. Therefore, there is $a(object_4, object_1) \in Add^f(\langle object_1, \_ \rangle)$.

A full grounding g is deleted by any action grounding $a(c)$ and partial pattern $a\langle p \rangle$ for which the partial grounding g_{id} fits the grounding g. For example, a grounding $g = \langle object_1, object_2 \rangle$ can be deleted by the partial grounding $g_{id} = \langle object_1, \_ \rangle$ or $g'_{id} = \langle \_, object_2 \rangle$.

Definition 18 (Deleting Actions)**.** Let there be a grounding $g = \langle g_1, ..., g_n \rangle$. For an action grounding $a(c)$ and a partial pattern $a\langle p \rangle$, let g_{id} be the partial grounding. Then $a(c)$ deletes g if $g^I = g_{id}$. The set $Del^f(g)$ contains all action groundings $a(c)$ that delete the grounding g for the feature f.

Let there be a feature $f = (2, \mathbf{A}, \mathbf{D})$ with identifying sets of positions $Is_f(f) = \{\{1\}, \{2\}\}$ and a full grounding $\langle object_1, object_2 \rangle$. Then $Del^f(\langle object_1, object_2 \rangle)$ contains all actions that delete this grounding. The grounding $\langle object_1, object_2 \rangle$ can be deleted by the partial groundings $\langle object_1, \_ \rangle$ and $\langle \_, object_2 \rangle$. Therefore, any action $a(c)$ deletes this grounding if there is a partial pattern $a\langle p \rangle$ such that the corresponding partial grounding g_{id} is either $\langle object_1, \_ \rangle$ or $\langle \_, object_2 \rangle$. An example is the action grounding $a(object_3, object_2)$ with a partial pattern $a[2, \_]$, since the partial grounding is $\langle object_1, \_ \rangle$. So, this action is deleting for the grounding and there is $a(object_3, object_2) \in Del^f(\langle object_1, \_ \rangle)$.

A second set of deleting actions are definitely deleting actions. These sets do not contain all deleting actions for a full grounding, but for a partial grounding.

Definition 19 (Definitely Deleting Actions)**.** Let there be a feature $f = (n, \mathbf{A}, \mathbf{D})$ and a *partial* grounding g_{id}. Then the set of $definitely\ deleting$ actions $Def^f(g_{id})$ contains all action groundings $a(c)$ for which there exists a partial pattern $a\langle p \rangle \in \mathbf{D}$ such that for the corresponding partial grounding g_{id}.

Using the above example, the set $Def^f(\langle object_1, \_\rangle)$ contains all action groundings $a(c)$ for which there exists a partial pattern $a\langle p\rangle$ such that the resulting partial grounding is $\langle object_1, \_\rangle$.

These sets are called definitely deleting, since the set $Def^f(g_{id})$ contains all actions $a(c)$ that must delete a grounding g with partial grounding g_{id}. This is because an action $a(c)$ with a partial pattern $a\langle p\rangle$ has a partial grounding g_{id}, which defines the identifying part of the grounding that is deleted. Later, these sets are used to check whether every action grounding of a *partial* pattern deletes a grounding g.

5.2 Admissibility of Identifier Features

Now, it will be defined under what conditions a feature is admissible. This will be done at the level of *partial* groundings. For a feature f, there is the set $IG(f)$, which contains all possible partial groundings for this feature. For these groundings, there can be at most one full grounding true in each state. For a partial grounding g_{id} of a feature f, the set $Add^f(g_{id})$ contains all actions that add a grounding g that fits g_{id}. If a full grounding g is added, the set $Del^f(g)$ contains all actions that delete the grounding.

For a feature to be admissible, it needs to fulfil three constraints. The first constraint is that the feature must be $identifying$. Therefore, for each *partial* grounding g_{id} with $identifying$ positions $Ip_g(g_{id})$, there can be in each state $s \in T(P)$ at most one grounding g true such that $g^{Ip_g(g_{id})} = g_{id}$. That means that for a partial grounding, there is only one full grounding that fits this partial grounding.

Definition 20 (Unique Value)**.** Let there be a state $s \in T(P)$. This state can contain for a partial grounding g_{id} with $identifying$ positions I at most one grounding g such that $g^I = g_{id}$.

If there is more than one grounding for a *partial* grounding, the arguments in the *non - identifying* positions cannot be concluded from the given arguments. For this constraint to be admissible, there are two possibilities. First, on each path to a state that contains an action that adds g with $g^I = g_{id}$, the last added grounding was g and the grounding was not deleted afterward. Second, on each path to a state that contains an action that adds g with $g^I = g_{id}$, the grounding was deleted before reaching s. Therefore, either the grounding g is true in the state or there is no grounding true that fits the partial grounding.

The second constraint is that for any partial pattern $a\langle p\rangle$ for every grounding $a(c)$ there needs to be a grounding g deleted. This constraint ensures that successive deleting actions are not possible.

Definition 21 (Unique Delete)**.** Let there be a feature $f = (n, \mathbf{A}, \mathbf{D})$ with $Is_f(f)$ and $a\langle p\rangle \in \mathbf{D}$ with $Ip_p(p)$. Let there be a state s with an outgoing action grounding $a(c)$ where g_{id} is the partial grounding of $a(c)$ with a partial pattern $a\langle p\rangle$. This action grounding needs to delete a grounding g that fits the partial grounding. The grounding g fits g_{id} if it holds that $g^{Ip_p(p)} = g_{id}$.

This does not need to hold if there does not exist a path to s that adds a grounding g with $g^{Ip_p(p)} = g_{id}$.

This constraint guarantees that successive deleting actions are not possible. This can be observed at the level of a *partial* grounding g_{id}. For a set of *identifying* positions I one can observe all actions that add a grounding g with $g^I = g_{id}$. Then, all definitely deleting action groundings $a(c) \in Def^f(g_{id})$ must delete one of these groundings. Thus, it can be checked for grounding $a(c)$ whether, in each state s where $a(c)$ is applied, there exists a grounding g with $g^I = g_{id}$. This boils down to checking whether on each path that reaches a state s the same grounding g is added and not deleted.

This constraint does not need to hold for actions that are not reached by any path that adds a grounding g with $g^I = g_{id}$, because then it is not clear which grounding g is true. Therefore, the *identified* arguments cannot be recovered. If an action is reached through a path that adds a grounding g and a path where no grounding is added, it is assumed that on all paths g was added. This also means that the path, where no fitting grounding is added, cannot contain an action that deletes this partial grounding.

The third constraint states that the same grounding g cannot be added two times in a row. This is because the domain is assumed to be well formed, and, therefore, a predicate that is added needs to be false previously. We only need to formulate this for the same grounding g (and not *partial* groundings g_{id}) since for different groundings g, g' this would already conflict with the constraint in Definition 20.

Definition 22 (Defined Value)**.** A grounding cannot be added or deleted at the same time. Therefore, for a state s where grounding g is true for a feature f, there cannot be an action grounding $a(c)$ applied that adds grounding g.

For this, there are two possibilities. First, the action $a(c)$ adds grounding g, and does not delete it, that means $a(c) \notin Del^f(g)$. The action then adds a grounding g, which is already true and not deleted. Therefore, the domain is no longer well formed. The second possibility is that $a(c) \in Del^f(g)$ and therefore the action deletes the grounding g. Then, the full grounding g is added and deleted at the same time, and it is not clear whether g is true or false in the following state. Both cases should not be possible.

Definition 23 (Admissible Feature)**.** An *identifier* feature f is admissible if for all possible *partial* groundings $g_{id} \in IG(f)$ the constraints in Definition 20, Definition 21, and Definition 22 are fulfilled.

5.3 Labeling Algorithm

Whether a feature is valid can be checked with a simple labeling algorithm. This is done at the level of *partial* groundings g_{id}. The labeling algorithm will label each state with all groundings

g that fit the partial grounding g_{id} and were added on a path to the state and not deleted. To do this, the algorithm receives as input the traces and a *partial* grounding g_{id} with identifying positions $Ip_g(g_{id})$.

For a *partial* grounding g_{id} with a set of *identifying* positions $Ip_g(g_{id})$, the label $l_{g_{id}}(s)$ contains all groundings g that are true in s and for fit g_{id}. There is an additional label $"-1"$ that can be contained in $l_{g_{id}}(s)$. The label "-1" states that on a path to this state a grounding g with $g^I = g_{id}$ was added and afterward deleted. Therefore, in the state s, there cannot be a grounding $g \in l_{g_{id}}$, unless it is added again. If there is a state s such that $g, -1 \in l_{g_{id}}(s)$, there must be two paths to this state that propagate different values. That means that one path states that there can not be a grounding g with partial grounding g_{id}, and the other path states that there is a grounding g true for the partial grounding g_{id}. For an admissible feature, this cannot be possible, since a grounding must have a unique value in each state.

Let there be a feature $f = (n, \mathbf{A}, \mathbf{D})$, and a *partial* grounding g_{id} with an *identifying* set of positions $Ip_g(g_{id})$. In the following, it is explained how the states are labeled. The set $Add^f(g_{id})$ contains all actions that add a grounding g with a partial grounding g_{id}. For each state s reached by an action $a(c) \in Add^f(g_{id})$ there is $g \in l_{g_{id}}(s)$, where g is the grounding added by $a(c)$. All labeled states are added to a queue, and for these states, the successors are labeled according to the following labeling rule. All states that are labeled by the labeling rule are also added to the queue.

Definition 24 (Labeling Rule)**.** Let there be an action grounding $a(c)$ on a transition from s to s', where s is a labeled state.

If $g \in l_{g_{id}}(s)$ and the grounding is not deleted ($a(c) \notin Del^f(g)$), then also for the reached state s' there must be $g \in l_{g_{id}}(s')$. If $g \in l_{g_{id}}(s)$ and $a(c) \in Del^f(g)$, then grounding g is deleted. Now, if no grounding is added, $a(c) \notin Add^f(g_{id})$, there is $-1 \in l_{g_{id}}(s')$. If a grounding is added, that means $a(c) \in Add^f(g_{id})$, then this grounding is already in the label of s'. Here, the coloring fails if the same full grounding g is added and deleted.

If for a state s there is $-1 \in l_{id}(s)$ and no grounding is added, that means $a(c) \notin Add^f(g_{id})$, then also for the successor state there is $-1 \in l_{g_{id}}(s')$.

By this labeling, every state that is reached via a directed path that contains an adding action for the partial grounding is labeled. This labeling is stopped early if one of the following conditions is violated. First, each state can contain only one value which can be a grounding g or the value -1. If there is a state s for which there is $g, g' \in l_{g_{id}}(s)$ then for partial grounding, the *identified* positions cannot be recovered. If there is a state s such that $-1, g \in l_{g_{id}}(s)$, then there are two paths to this state, one where a grounding is added and one where a grounding is added and deleted afterward. Therefore, one path states that there exists a grounding g with $g^I = g_{id}$ and the other states that there cannot be such a grounding. So, for the state, there is no unique value for the partial grounding.

The second condition is that each definitely deleting action $a(c) \in Def^f(g_{id})$ needs to delete a grounding g with identifying grounding g_{id}. For a state s in which $a(c)$ is applied, there needs to be a grounding g true that is deleted and, therefore, there needs to be $g \in l_{g_{id}}(s)$. This is because for any grounding g, that can be deleted by $a(c)$, needs to fit the partial grounding g_{id}. If $a(c)$ is applied in a state *labelled* with " -1", the action cannot delete a grounding g. Then the feature can not be admissible, since each partial pattern needs to delete a grounding.

Definition 25 (Admissible Labeling)**.** A labeling is admissible if it does not fail and for each state and identifying grounding g_{id}, the label contains at most one value. Also, for each *deleting* action $a(c)$ and partial pattern $a\langle p\rangle$, there must be a grounding g deleted.

5.4 Correctness of Labeling

Now we want to show that for each *identifying* predicate $\bar{p}$ there exists an *identifier* feature f' that is valid. For this, $\bar{p}$ needs to have at least one set of *identifying* positions I. First, this predicate can be described as a feature f that is valid for the SIFT [9] algorithm.

Lemma 6. *For a predicate $\bar{p}$ from a domain $\mathcal{D}$ there is a feature f that is admissible for the algorithm SIFT for any trace drawn from $\mathcal{D}$.*

Proof. Since the domain is assumed to be well formed, the feature is valid. A proof can be seen in [9]. □

Since f is valid, all patterns that add the feature either have sign 0 or 1. Patterns that deleted the feature have the opposite sign. Previously, it was assumed that effects that add an *identifying* relation are positive effects. To better reason about the features w.l.o.g. the action patterns with positive effects have sign 1 and the action pattern with negative effects have sign 0.

Now it is clear which action patterns add the feature f and which action patterns delete it. Since by assumption positive effects add the identifying relation, these action patterns are also adding for the *identifier* feature f'. Also, the deleting patterns need to be similar, with the difference that in f these are $full$ patterns and in f' these are *partial* patterns. For each deleting action pattern, there needs to be a corresponding partial pattern that is defined by an identifying set of positions of the predicate. The adding actions and the *partial* patterns together result in the *identifier* feature f'.

For the following proofs, we assume that the arguments are counted, even if they are not visible to the learner. This assumption does not limit the approach, but makes the formalism easier. For an action $a(x, \not{y}, z)$ where the crossed-out argument is removed and is not visible to the algorithm, the resulting action will be $a(x, z)$. Then an action pattern $a[1, 3]$ would become an action pattern $a[1, 2]$. Under the above assumption, the action will become $a(x, \_, z)$, and

therefore the action pattern will still be $a[1,3]$. With this formalism, now an *identifier* feature f' can be defined for a predicate $\bar{p}$.

Definition 26. Let there be an *identifying* predicate $\bar{p}$ with *identifying* sets of positions $Is_p(\bar{p})$ and the corresponding feature $f = \langle n, B\rangle$. Now we define the *identifier* feature $f' = (n, \mathbf{A}, \mathbf{D})$. For each full pattern $a[p] \in B$ with $sign(a[p]) = 1$ there is $a[p] \in \mathbf{A}$. For every full pattern $a[p] \in B$ with $sign(a[p]) = 0$ there needs to be a set of *identifying* positions $i \in Is_p(\bar{p})$, which will be referred to as $i_{a[p]}$. For a pattern $a[p] \in B$ with $sign(a[p]) = 0$ and a set of *identifying* positions $i_{a[p]}$, there is $a\langle p'\rangle \in \mathbf{D}$ where for p' there is $p'_j = p_j$ if $j \in i_{a[p]}$ and $p'_j = \_$ otherwise.

This feature defines which arguments are used to *identify* the full grounding for a deleting action. Therefore, when the feature is defined, it is clear which arguments are necessary to learn this feature and which are unnecessary.

To show that the *identifying* feature f' is valid, it is not interesting whether the *identified* arguments are contained in the argument list. This is because a partial pattern only uses the *identifying* positions to delete a grounding. Whether the arguments are given is only interesting when the *identified* arguments are added to the argument lists. In the following, we first show that the feature f' is valid. Afterward, it will be shown how the *identified* arguments are added to the argument lists.

Assumption 3. We assume that in the traces unnecessary arguments may be removed, but all necessary arguments are contained.

To show that f' is valid, we first show that for f there can be a labeling constructed that is admissible for the constraints. Then it will be shown that this labeling is similar to the labeling of the feature f'.

The combined labeling will be done for each possible partial grounding $g_{id} \in IG(f)$. In the combined labeling of g_{id} all groundings g, which fit the partial grounding g', are contained. That means that for a *partial* grounding $\langle obj_1, \_\rangle$ all groundings $\langle obj_1, obj_j\rangle$ are considered. The goal is to have for each state s a label $\bar{\chi}_{g_{id}}(s)$ that contains all full groundings g with partial grounding g_{id}, which are true in s.

To build labelings for the feature f, the coloring algorithm defined in Section 4 can be used. Since the predicate $\bar{p}$ is valid by Lemma 6, the colorings must be admissible for all groundings. Since it is assumed that an action pattern with a positive sign adds the feature, a state s is colored with 1 for grounding g if $p(g)$ is true in s. Now a labeling for a partial grounding g_{id} with identifying positions I combines all the colorings of the groundings g with $g^I = g_{id}$. For a state in which no grounding g with $g^I = g_{id}$ is true, the state will be labeled " -1".

Definition 27 (Combined Labeling)**.** Let there be a feature $f = \langle n, B\rangle$, and a partial grounding g_{id} with identifying positions I. For every grounding g with this partial grounding g_{id}, there

is $g \in \bar{\chi}_{g_{id}}(s)$ iff there is $\chi_g(s) = 1$. An state s where no grounding g with $g^I = g_{id}$ is true is labeled "-1".

Now we first show that the combined labeling $\bar{\chi}_{g_{id}}$ fulfills the constraints of an admissible *identifier* feature. Afterward, we show that the labels $l_{g_{id}}$ of the feature f' are equal to the label $\bar{\chi}_{g_{id}}$ for all states s that are reached via a path that adds the *identifying* grounding g_{id}.

Lemma 7. *In the combined labeling $\bar{\chi}_{g_{id}}$ for a partial grounding g_{id} that has identifying positions $Ip_g(g_{id})$ with $Ip_g(g_{id}) \in Is_p(\bar{p})$, each state s contains at most one grounding g.*

Proof. By Lemma 5 we know that the coloring χ_g for a grounding g is equal to the value of the predicate p in the state s. The predicate p is *identifying* with positions $Ip_g(g_{id})$. Therefore, there can be at most one grounding g true in s with $g^{Ip_g(g_{id})} = g_{id}$. Let there be a state s such that $g, \bar{g} \in \bar{\chi}_{id}(s)$ with $g \neq \bar{g}$. Then there exist two groundings $g, \bar{g}$ such that $\chi_g(s) = \chi_{\bar{g}}(s) = 1$. Therefore, the predicate cannot be *identifying* for the positions $Ip_g(g_{id})$, since there exist two groundings $g, \bar{g}$ with the same *identifying* grounding g_{id} that are true in s. That there is $g, -1 \in \bar{\chi}_{g_{id}}(s)$ is not possible by definition of $\bar{\chi}_{g_{id}}$. □

Lemma 8. *For every action grounding $a(c)$ for every partial pattern $a\langle p\rangle$ there is a grounding g deleted.*

Proof. The partial patterns in this context are the action pattern $a[p]$ with $sign(a[p]) = 0$. Therefore, an action grounding $a(c)$ with the *full* pattern $a[p]$ deletes the grounding g. The grounding g is already uniquely determined, since $a[p]$ is a *full* pattern. Since the domain is well formed, the grounding g needs to be true in s, otherwise $a(c)$ is not applicable. Since the grounding that is deleted is uniquely determined, the action can also delete no other grounding. □

Lemma 9. *There are no consecutive action groundings $a(c_1, ..., c_n)$, without a delete in between.*

Proof. Let there be two action groundings $a(c), b(c')$ with action pattern $a[p], b[p']$, where the actions add the same grounding and are successive in the trace $T(P)$. We know that the feature f is valid in SIFT. Therefore, the action pattern $a[p]$ and $b[p']$ must have different signs in the feature. Since both action patterns are adding effects for the feature f' they are also adding effects for f and this is a contradiction to Lemma 6. □

Theorem 2. *The combined labeling $\bar{\chi}_{g_{id}}$ fulfills the constraints for the partial grounding g_{id}.*

Proof. By Lemma 7, Lemma 8, and Lemma 9 all constraints are satisfied for all *identifying* groundings g_{id}. □

Now, to show that the feature f' is valid, we will show that the labeling $l_{g_{id}}$ is equal to the combined labeling $\bar{\chi}_{g_{id}}$ for all $g_{id} \in IG(f)$.

Lemma 10. *The labeling $\bar{\chi}_{g_{id}}$ will be equal to the labeling $l_{g_{id}}$ obtained for the feature f' for any g_{id} for all states that are reached by paths that add a grounding g that fits g_{id}.*

Proof. By definition, the feature $f = \langle n, B \rangle$ contains the same adding actions as the *identifier* feature $f' = (n, \mathbf{A}, \mathbf{D})$. Therefore, in each state s, which is reached directly through an action $a(c) \in Add^f(g_{id})$, the labels are equal. For these actions by Assumption 3 all arguments are contained in the input of the labeling algorithm. Then it is enough to show that a grounding g is deleted in $\bar{\chi}_{g_{id}}$ iff it is deleted in $l_{g_{id}}$.

" $\Rightarrow$ " Let there be a grounding g that is deleted by action $a(o_1, ..., o_n)$ in $\bar{\chi}_{g_{id}}$ but not in $l_{g_{id}}$ by action $a(o'_1, ..., o'_n)$. Therefore, there exists a deleting action pattern that has sign 0. By definition, there needs to be a set of *identifying* positions I for this effect. For these positions I and the effect $a[p]$ of f there must be a partial pattern $a\langle p' \rangle$ in the feature f'. For the pattern p' there is $p'_j = p_j$ if $j \in I$ and else there is $p'_j = \_$. The arguments for the *identifying* positions of the partial pattern p' must be contained in the arguments list since they are necessary. Since $a[p]$ has the grounding $g = \langle o_{p_1}, ..., o_{p_m} \rangle$, the partial pattern $a\langle p' \rangle$ has the partial grounding $g' = \langle o'_{p_1}, ..., o'_{p_m} \rangle$. For every position $1 \leq k \leq m$ there is either $o'_{p_j} = \_$ or $o'_{p_j} = o_{p_j}$. Therefore, g' is a partial pattern of g and by definition $a\langle p' \rangle$ needs to delete the grounding g.

" $\Leftarrow$ " Let there be a grounding g that is only deleted in the labeling $l_{g_{id}}$ by a *partial* pattern $a\langle p' \rangle$. The set of *identifying* positions for this action pattern is $i_{a[p]}$. Since there is a *partial* pattern $a\langle p' \rangle \in \mathbf{D}$, there needs to be a pattern $a[p] \in B$ with $sign(a[p]) = 0$. Then the full action $a(o_1, ..., o_n)$ needs to delete a grounding $\bar{g}$ with $\bar{g}^i = g_{id}$. Therefore, there must be two groundings $g, \bar{g}$ with the same identifying grounding $g^i = \bar{g}^i = g_{id}$ in the state s. Therefore, s is labeled with two groundings and therefore the feature is not *identifying*. □

Theorem 3 (Completeness)**.** *A removed unnecessary argument x_i of a deleting action can be recovered if all necessary arguments are contained. This holds for all deleting actions that are reachable via a path that adds a grounding that fits the corresponding partial grounding.*

Proof. Let there be an action $a(x_1, ..., x_n)$ where x_i is an unnecessary argument that was removed. Since x_i was contained in an *identified* position, there must be an *identifying* predicate $\bar{p}$. For the predicate $\bar{p}$ there is a corresponding feature f that is valid in SIFT by Lemma 6. For the feature f there can be an *identifier* feature f' by Definition 26 constructed such that x_i is in an *unnecessary* position. Since by assumption all necessary arguments are contained in the input, all arguments for adding actions are contained in the traces. Therefore, also all arguments of the *identifying* positions of the deleting effects are contained in the traces. Then the combined labeling $\bar{\chi}_{g_{id}}$ for every *identifying* grounding $g_{id} \in IG(f')$ can be constructed

by Definition 27. All labels $\bar{\chi}_{g_{id}}$ for $g_{id} \in IG(f')$ satisfy all the conditions for an *identifier* feature by Theorem 2 and are therefore admissible. Since the labeling $l_{g_{id}}$ will be equal to the labeling $\bar{\chi}_{g_{id}}$ for all states reached by an action that adds a full grounding with partial grounding g_{id}, also f' is valid (Lemma 10). We know that the labeling of $\bar{\chi}_{g_{id}}(s)$ contains for each partial pattern $a\langle p\rangle$ for each action grounding $a(c)$, which is reached by a path that adds a grounding g' that fits g_{id}, a grounding g that is deleted. From the partial pattern, the identifying positions I are known. From these positions and the grounding g the *identified* arguments can be obtained. □

The *identified* arguments that are found can then be added to the arguments lists of the action groundings.

Definition 28 (Adding Arguments for Deleting Actions)**.** For each action name a and *partial* pattern $a\langle p\rangle$ there is a set of *identified* arguments. Let $a^1, ..., a^n$ be the action groundings of a, which are reached via a path that adds a grounding that fits the corresponding identifying grounding g_{id} for the partial pattern $a\langle p\rangle$. Then for a partial pattern $a\langle p\rangle$, there are lists of *identified* arguments $o^1, ..., o^n$. Each action a^j has arity m, let a^j_i be the i-th argument of the j-th action grounding. Each list of *identified* arguments contains l objects, let o^j_i be the i-th object in the list of *identified* arguments for the j-th action grounding. An object position i is either added for all action groundings or for none. The arguments of position $k \in \{1, ..., l\}$ in the object lists are added to the argument lists iff there exists no position $i \in \{1, ..., m\}$ such $a^j_i = o^j_k$ for all $1 \leq j \leq n$. That means that the object is not added if there is already a position in the argument list that contains these objects. Each action that is not reached via a path that adds the feature of the corresponding argument is removed from the trace. This only needs to be done if a new object is added.

The arguments are not added when there already exists a position in the argument lists because this does not add new information. Any effect that uses the added objects can build with the objects that are already contained in the argument lists of the actions. Therefore, only the action arity is changed. By this the runtime of the SIFT algorithm increases, since it depends on the arity of the input actions.

5.5 Identifier Preconditions

For an *identifying* predicate, there can also be positive preconditions, for which arguments may be unnecessary. Assume that there is a predicate $p(x_1, x_2)$ that is identifying with positions $I = \{1\}$. Now an action a can have a precondition on this predicate, without changing it. If an action $a(y_1, y_2, y_3)$ has a positive precondition on $p(y_1, y_2)$, then argument y_2 can be unnecessary. Therefore, given the argument y_1 the argument y_2 can be recovered, since p can be true for partial grounding $\langle y_1, \_\rangle$ at most once. In the context of a feature, this precondition can be

stated as a partial pattern. This is similar to SIFT, where a precondition is defined by a full pattern [9].

Definition 29 (Identifier Precondition)**.** For an admissible $identifier$ feature $f = (n, \mathbf{A}, \mathbf{D})$, a positive $identifier$ precondition is a $partial$ pattern $a\langle p_1, ..., p_n\rangle$ such that $a\langle p_1, ..., p_n\rangle \notin \mathbf{D}$.

Positive identifier preconditions can only exist for an admissible $identifier$ feature f. The partial pattern of a positive precondition does not need to be related to the $partial$ pattern of the $identifier$ feature. That means that a positive precondition $a\langle p\rangle$ on a feature f can have a set of identifying positions $Ip_p(p)$ such that $Ip_p(p) \notin Is_f(f)$. Therefore, it must be checked for any possible partial pattern $a\langle p\rangle$ with the same arity as f, whether it is a valid precondition. This is similar to checking whether a partial pattern is a deleting effect with the difference that the precondition does not delete a grounding.

When an action a has a positive precondition on an $identifying$ predicate p, then it is only applicable when a grounding g satisfies this precondition. This must also be true for the corresponding precondition $a\langle p\rangle$ on an admissible $identifier$ feature f. Therefore, for the precondition $a\langle p\rangle$ and the action grounding $a(c)$ for the resulting partial grounding g_{id} there must be a grounding g that fits g_{id}, in any state where $a(c)$ is applied. This can only be checked for states that are reached by a path on which a grounding g with $g^I = g_{id}$ is added.

Definition 30 (Valid Precondition)**.** Let there be a valid predicate p and a precondition $a\langle p\rangle$. Let g_{id} be the partial grounding of the action grounding $a(c)$ for the partial pattern $a\langle p\rangle$. Then $a\langle p\rangle$ is an admissible precondition if in any state s where an action grounding $a(c)$ is applied, for the corresponding partial grounding g_{id} there is $-1 \notin l_{g_{id}}(s)$.

The reason behind this definition is that an action with a precondition cannot be applied and the precondition is not fulfilled. Therefore, "-1" cannot be contained in a label for a partial grounding of a precondition. When the label is empty, no fitting grounding is added on the path that reaches this node. In this case, it is not clear whether and how the precondition is fulfilled, but it is assumed that it is fulfilled because it is not disproven.

For an identifying precondition $a\langle p\rangle$ to be valid, the predicate $\bar{p}$ must be $identinfying$ for the positions $Ip_p(p)$. The predicate $\bar{p}$ is valid for SIFT and identifying for the positions $Ip_p(p)$. Therefore, for every partial grounding for the identifying set of positions $Ip_p(p)$, the combined labeling can be built by Definition 27. Since $\bar{p}$ is identifying for positions $Is_p(p)$ the Lemmas 7, 8, and 9 apply and therefore also Theorem 2.

Now we know that f is valid and that for all partial grounding g_{id} with $Is_g(g_{id}) = Is_p(p)$, the labeling $l_{g_{id}}$ is equal to the labeling $\bar{\chi}_{g_{id}}$ for all states that are reached through a path that adds g with $g^{Ip_p(p)} = g_{id}$. Since $Ip_p(p)$ is a set of identifying positions, there can also be at most one grounding true.

Now we show that for every precondition on an *identifying* predicate $\bar{p}$, there is a valid precondition $a\langle p\rangle$ on the corresponding *identifier* feature f.

Theorem 4. *For a positive precondition in a predicate p with identifying set of positions I there will be a admissible precondition $a\langle p\rangle$ for f with $Ip_p(p)$. For this, all necessary arguments must be contained in the input.*

Proof. For a precondition $a\langle p\rangle$ not to be admissible, there must exist a state s in which an action $a(c)$ is applied and there is no partial grounding true for the precondition. Therefore, for an action grounding $a(c)$ and a precondition $a\langle p\rangle$, for the resulting partial grounding g_{id} there cannot be a label such that $-1 \in l_{g_{id}}(s)$ for a state s where $a(c)$ is applied.

By Lemma 14, we know that the predicate $\bar{p}$ is valid for the algorithm SIFT. Therefore, there will be a valid feature $\bar{f} = \langle n, B\rangle$ for the algorithm SIFT. For any possible precondition of $\bar{p}$, there will be a precondition for the feature $\bar{f}$. Therefore, for the precondition $a\langle p\rangle$ of f there is a precondition $a[p']$ of $\bar{f}$ such that $p'_i = p_i$ or $p_i = \_$ for $1 \leq i \leq n$.

Now assume for a state s that there is $-1 \in l_{g_{id}}(s)$ and therefore also $-1 \in \bar{\chi}_{g_{id}}(s)$. Since $a(c)$ is applied in the traces $T'(P)$ with the removed arguments there needs to be $a(\bar{c})$ applied in the traces $T(P)$ including the removed arguments. Since the action $a(\bar{c})$ is applied, there must be a grounding $\bar{g}$ that fulfills the precondition $a[p']$ of the feature $\bar{f}$. By Lemma 5, we know that the value of $\chi_{\bar{g}}(s)$ is 1. Since all necessary arguments are contained in the input, for all necessary positions i there is $c_i = \bar{c}_i$. Since $a[p']$ is the full pattern of the partial pattern $a\langle p\rangle$ there is for $1 \leq j \leq n$ either $p'_j = p_j$ or there is $p_j = \_$. Therefore, for $1 \leq j \leq n$ there is either $c_{p_j} = \bar{c}_{p'_j}$ or there is $c_{p_j} = \_$. Now, since g fulfills $a[p']$, it also needs to fulfill $a\langle p\rangle$. Therefore, there is $\bar{\chi}_{g_{id}} = 1$ and the precondition $a\langle p\rangle$ is fulfilled. □

Therefore, for each precondition there is a grounding that fulfills the precondition. The arguments of the groundings can be added to the arguments lists in the same way as arguments for deleting actions are added.

Definition 31 (Adding Arguments for Admissible Preconditions)**.** Let there be an admissible precondition $a\langle p\rangle$ for a valid feature f. Let $a^1, ..., a^n$ be the action groundings of a, which are reached via a path that adds the corresponding partial grounding for the action pattern $a\langle p\rangle$. Then for each action grounding there is a list of new arguments o, where o^j is the list of additional arguments for the action grounding a^j. We know that each action grounding has arity m and the list of new arguments has length l with $l < n$. Now, the arguments for a position position $f \in \{1, ..., l\}$ in the object tuples are not added to the argument lists if there exists a position $j \in \{1, ..., m\}$ in the argument lists that already contains exactly these arguments. That means that there is a position j in the argument list such that for $1 \leq k \leq n$ there is $a^k_j = o^k_f$. If this is not the case, the arguments are added to the argument lists. These arguments can only

be added for actions where the grounding of the preconditions is known, and all other action groundings are deleted.

Arguments of actions for which the corresponding grounding was not added for the action pattern are removed since it is not clear which grounding is true. Therefore, it is not clear which identified positions need to be added. If the actions are not deleted, there are actions with the same action name but different arities. Currently, this cannot be handled by the SIFT algorithm.

Now, it can be shown that all unnecessary arguments can be removed if all necessary arguments are contained in the input.

Theorem 5. *All unnecessary arguments that are removed from the action lists can be recovered if all necessary arguments are contained in the input. Arguments can only be recovered if the grounding, on which the action has a precondition or deleting effect, was added beforehand.*

Proof. By Theorem 3 all unnecessary arguments of *identifying* features can be recovered, if the deleting action is reached via a path that adds the grounding that is deleted. By Theorem 4 all arguments for positive preconditions can be recovered if the grounding of the precondition was added beforehand. Therefore, all unnecessary arguments can be recovered. □

5.6 Dependence on Partial Graphs

In the following, we will show that traces with state equality can filter out more features than traces without state equality. That means that extended traces contain more information than non-extended traces. This can be shown for predicates where an action adds and deletes a grounding with the same *identifying* grounding. The corresponding features will be called *switching* features.

Definition 32 (Switching Feature)**.** Let there be a feature $f = \{n, \mathbf{A}, \mathbf{D}\}$. The feature f has a switching effect if there exist $a[p] \in \mathbf{A}$ and $a\langle p' \rangle \in \mathbf{D}$ such that for $1 \leq j \leq n$ either $p_j = p'_j$ or $p'_j = \_$. That means that the action a adds a grounding g and deletes a grounding g' which must have the same *identifying* grounding. Additionally, there must be an action that adds a grounding, without deleting a grounding with the same identifying grounding and vice versa. All effects need to have the same *identifying* set of positions I.

An example of a switching feature is the *on* predicate in the blocks world domain with 3 actions. The action $move(x, y, z)$ deletes the grounding $\langle x, y \rangle$, while also adding the grounding $\langle x, z \rangle$. Since the predicate *on* has identifying positions $i = \{1\}$, for the *identifying* grounding $\langle x, \_ \rangle$ there is a grounding added and deleted by the *move* action. The action $stack(x, y)$ adds the feature for grounding $\langle x, y \rangle$ and the action $unstack(x, y)$ deletes the feature for $\langle x, y \rangle$.

For a switching feature, also the feature without the switching effect will be admissible for non-extended traces. The reason for this is that each state is only reachable via one path. Therefore, it cannot happen that a state is reached via two different adding, non-switching actions. This is exactly the situation that shows that a *switching* action is necessary for the feature to be admissible. This can be seen in the blocks world domain.

Lemma 11. *For a switching feature f there can exist an extended trace $T(P)$ such that a feature f' without a switching action is not admissible.*

Proof. In the blocks world domain with 3 actions there is the predicate $on(?x, ?y)$ that states that block $?x$ is stacked on block $?y$. The predicate *on* can be described as $identifier$ feature $f = (2, \{stack[1,2], move[1,3]\}, \{unstack[1,\_], move[1,\_]\})$. For this feature the *move* action is a switching effect. The *on* predicate without the switching effect can then be described by the feature $f' = (n, \{stack[1,2]\}, \{unstack[1,\_]\})$.

For the feature f' the action $stack(b_1, b_2)$ adds the grounding $\langle b_1, b_2\rangle$ and the action sequence $stack(b_1, b_3)$, $move(b_1, b_2)$ adds the grounding $\langle b_1, b_3\rangle$ for the feature. Therefore, the sequences add different groundings for the same $identifying$ grounding $\langle b_1, \_\rangle$. As one can see, the sequences add different groundings for the feature. In the domain, on the other hand, both sequences add the same grounding $\langle b_1, b_2\rangle$ for the $identifying$ grounding $\langle b_1, \_\rangle$, and can reach the same state s. When these two action sequences reach the same state s, the feature without the switching effect is not admissible. This is because for the $identifying$ grounding $\langle b_1, \_\rangle$ there are two groundings true. That means, for the feature f' for $identifying$ grounding $\langle b_1, \_\rangle$ there is $\langle b_1, b_3\rangle, \langle b_1, b_2\rangle \in l_{(b_1,\_)}(s)$. Therefore, the feature is not admissible.

□

The above example used in the proof can only be seen in extended traces. This is because in a non-extended trace each state is only reached by a single path. In the following, we will show that extended traces provide more information than non-extended traces. For this we will prove that for a valid switching feature f also the feature f' without the switching effects is admissible.

Lemma 12. *For a feature f that contains switching effects, the feature without switching effects will be admissible for non-extended traces.*

Proof. Let f' be the feature that is equal to a valid switching feature f without the switching effects. To prove that the feature f' is admissible, we show that if f' is not admissible, then also f is not admissible. For this, we note that any effect of f' is also an effect of f.

Let there be a state s such that there are two groundings $g, \bar{g} \in l^{f'}_{g_{id}}(s)$. Then there are either two groundings with the same partial grounding added at the same time, or they are added after each other. They cannot be added at the same time since this also needs to hold for f and

then f is also not admissible. Therefore, there needs to be a state s' such that $g \in l^{f'}_{g_{id}}(s')$ and the next action $a(c)$ adds $\bar{g}$. Then there is also $g' \in l^{f}_{g_{id}}(s')$, else a grounding is added / deleted for f' and not for f or vice versa. This grounding does not need to be equal to g since it can be switched before reaching the state. Now this grounding g' needs to be deleted by a switching action, otherwise there are two groundings true in s'. The switching action also needs to add a grounding with partial grounding g_{id}. Therefore, either there are two groundings true in the reached state, or the same grounding is added twice. Both cases are not possible.

Let there be a state s in which a deleting action cannot be applied for f'. Then, for the deleted *partial* grounding g_{id} there is $l^{f'}_{g_{id}}(s) = \{-1\}$. Then in some previous state s' a grounding g with an identifying grounding g_{id} was deleted. Since every action pattern contained in f' is also contained in f, for f also a grounding with *partial* grounding g_{id} was deleted. Since the deleting effect is contained in f' it cannot be a switching effect, and therefore there is no grounding g with partial grounding g_{id} added. For f to be admissible, there needs to be some $g \in l^{f}_{g_{id}}(s)$, therefore there needs to be a grounding added between s' and s. Every action that adds a grounding for f and does not need to delete a grounding also needs to be contained in f', therefore also $l^{f}_{g_{id}}(s) = \{-1\}$ and f is not admissible.

Let there be a grounding g that for f' is added twice in a row. Therefore, there are two groundings with the same $identifying$ grounding added. This also holds for f. Therefore, also for f there is a grounding with the same $identifying$ grounding g_{id} added twice in a row. The value of g may be switched in the meantime, therefore, either g is added twice in a row, or there are $\bar{g}, \bar{g}'$ true in the same state. In both cases f is not admissible.

□

Theorem 6. *There exists an $identifier$ features f that can be invalidated based on extended traces but not with non-extended traces.*

Proof. By Lemma 12, it is not possible to show that for a valid *switching* feature f the feature without the switching effects is not admissible. In Lemma 11 it was shown that this is possible for extended traces. □

That the feature f' without the switching effects can be admissible for non-extended traces is a problem. This can be seen in the blocks world example. If the feature without the switching effect is admissible, there will be an additional argument. This argument states which *stack* action $stack(b_i, b_j)$ was done for a block b_i. Afterwards, the block can be moved onto other blocks. For the *unstack* action, the additional argument is the block b_j on which b_i was stacked by the action *stack* and not the block on which it is currently stacked. This additional input will lead to a point where SIFT learns a predicate that is not contained in the maximal domain

description of the blocks world domain. This is because for any admissible identifier feature, there will be a feature that is admissible for SIFT.

5.7 Global Features

Identifier features are used to find additional arguments based on *identifying* predicates. Therefore, the additional arguments need to be in relation to other arguments. For this, the corresponding predicate needs to have at least arity 2. In the following, it will be shown how this can be extended to unary predicates. For this, global predicates are introduced. These are predicates that can only be true for at most one grounding g in each state.

An example can be found in the blocks world domain with 4 actions. In this blocks world version, blocks are moved by an arm which can grab one block at a time. For this, there exists the predicate *grabbed*(?*block*), which is true for a block that is currently grabbed. Now in each state, there can be at most one block grabbed, therefore *grabbed* is true for at most one grounding.

Definition 33 (Global Predicate)**.** A predicate p is a *global* predicate if in each state s the predicate p is true for at most one grounding g.

As for *identifying* predicates, also for global predicates, necessary and unnecessary arguments can be defined. Here, the necessary arguments are the arguments that are an adding effect for a global feature.

Definition 34 (Global Necessary Arguments)**.** For an action $a(x_1, ..., x_n)$, an argument x_i is necessary if it is contained in an adding effect of a global predicate.

In addition, unnecessary global arguments can be defined. These are the arguments that can be learned, when the necessary arguments are contained in the input.

Definition 35 (Global Unnecessary Arguments)**.** Let there be an action $a(x_1, ..., x_n)$. Then an action argument x_i is unnecessary if x_i is not necessary and is used as a positive precondition or a deleting effect of at least one *global* predicate.

Global features are deleted by an action name a with arbitrary grounding c. This corresponds to a *partial* pattern that has a "_" at each position of the pattern. This is because if there is only one grounding true for the predicate, there does not need to be a partial grounding stated to *identify* the grounding. These patterns will be called the *empty partial* pattern in the following.

Definition 36 (Empty Partial Pattern)**.** An *empty* partial pattern $a\langle p\rangle$ of arity n is an action name a combined with a pattern $p = \langle p_1, ..., p_n\rangle$ where $p_j = \_$ for all positions $1 \leq j \leq n$.

Since empty partial patterns only contain "_" in the patterns, they do not have a set of identifying positions.

With *empty partial* pattern *global* features can be defined. These are *identifying* features $f = (n, \mathbf{A}, \mathbf{D})$ where $\mathbf{D}$ only contains *empty partial* patterns. These features cannot have any set of *identifying* positions i, since they only contain empty partial patterns. Therefore, the previous proofs do not apply directly. This is because the proofs for an admissible *identifier* feature assume that there is a set of identifying positions I.

To ensure that every *global* predicate has an *identifying* position the problem P will be lifted with a new argument. That means that there is a new argument c_{lift} that is appended to every action in the problem. This lifting was also done previously in LOCM [6]. Each action $a(x_1, ..., x_n)$ of arity n becomes an action $a(x_1, ..., x_n, x_{n+1})$ of arity $n + 1$. For each action grounding $a(c_1, ..., c_n)$ the same object c_{lift} is appended, so there is $a(c_1, ..., c_n, c_{lift})$.

For the problem $P = (\mathcal{D}, \mathcal{I})$, let $P' = (\mathcal{D}', \mathcal{I}')$ be the corresponding lifted problem. The domain $\mathcal{D}'$ will contain the same predicates with the same effects and preconditions as $\mathcal{D}$. Each action schema in $\mathcal{D}$ will have an additional argument position, which will contain the lifted argument. Additionally there will be a predicate p' which will have the same effects and preconditions as p, but the lifted argument x_{lift} is appended to each of them. It is assumed that for an initial state, if p is true for a grounding $g = \langle c_1, ..., c_n \rangle$ there is also p' true for $\langle c_1, ..., c_n, c_{lift} \rangle$. In the following, we will show that in the domain $\mathcal{D}'$ the predicate p is true for a grounding $\langle c_1, ..., c_n \rangle$ iff p' is true for a grounding $\langle c_1, ..., c_n, c_{lift} \rangle$.

Lemma 13. *If there is a state s in which $p(c_1, ..., c_n)$ is true for P', then in this state s there is also $p'(c_1, ..., c_n, c_{lifted})$ true.*

Proof. Assume there is an initial state s_{init} such that either p and p' are false for all groundings or there exist atoms $p(c_1, ..., c_n)$ and $p'(c_1, ..., c_n, c_{lift})$ which are the only true atoms in s_{init} for p and p'. Now p' has the same effects as p, but for every effect, the lifted argument is appended. Therefore, if $p'(c_1, ..., c_n, c_{lift})$ is added, then $p(c_1, ..., c_n)$ is also added. The same holds for deleting effects. Therefore, in any state s' where $p'(c_1, ..., c_n, c_{lift})$ is true, there is also $p(c_1, ..., c_n)$ true. Since there can only be one grounding $p(c_1, ..., c_n)$ true in each state, there can also be only one grounding $p(c_1, ..., c_n, c_{lift})$ true in each state. □

From a trace $T(P)$ that was sampled from the problem P there can be a trace for the problem P' be created.

Definition 37 (Lifted Input)**.** Let there be an extended trace $T(P)$ which has been sampled from a problem $P = \langle \mathcal{D}, \mathcal{I} \rangle$. Now, let there be a new trace $T(P')$ that has the same transitions as $T(P)$. For every grounding $a(c_1, ..., c_n)$ in $T(P)$ there is a grounding $a(c_1, ..., c_n, c_{lift})$ in $T(P')$.

These traces are valid by the definition of the domain $\mathcal{D}'$. Now, it can be shown that a *global* predicate together with the added argument becomes an *identifier* feature. This is because the position of the new argument can be used as the *identifying* position of the feature.

Assumption 4. We assume that all necessary arguments are contained in the traces. That means that all necessary arguments for the global features and the *lifted* argument c_{lift} are contained in each action.

When the above assumption holds, the proofs of *identifier* features can be used to show that unnecessary arguments can be removed. Then the results of Theorem 3 can be applied.

Lemma 14. *The predicate p' has a valid identifier feature f' for the trace $T(P')$.*

Proof. The predicate $p'(x_1, ..., x_n, x_{lift})$ from the problem P' is identifying for the position $I = \{x_{lift}\}$. This is because the predicate p can only be true for one grounding in each state and by Lemma 13 this also applies for p'. Therefore, for the identifying grounding $g_{id} = \langle \_, ..., \_, o_{lift} \rangle$ there can be at most one grounding g true in s such that $g^I = g_{id}$.

By Definition 26 the feature f can be defined in a way that for every deleting effect the *identifier* is the position of the lifted argument. That means for a deleting action pattern $a[x_1, ..., x_n, x_{lift}]$ the partial pattern is $a[\_, ..., \_, x_{lift}]$. This can be done since the predicate is *identifying* for the argument used to lift the domain.

Since all necessary arguments of the global feature are contained in the input, and also the argument c_{lift} is contained for all actions, all necessary arguments are contained in the input. This is because any effect of p is also an effect of p', therefore any argument that is necessary for p is also necessary for p', besides the lifted argument.

For the predicate p' there exists a feature $\bar{f}$ that is valid for the constraint of SIFT for the trace $T(P')$. Therefore, it is possible to define the combined colorings by Definition 27. Since the combined colorings exist and the predicate is identifying, by Theorem 2 the colorings fulfill the constraint on a valid *identifier* feature. Then, since all necessary arguments are contained in the input, by Theorem 3 the feature f' is valid. □

Then, by definition, the *identified* positions of the arguments can be added.

Theorem 7. *All global unnecessary arguments can be recovered if all necessary arguments are contained in the input. Arguments for deleting actions or preconditions can only be recovered if on the paths to the action grounding the corresponding grounding is added.*

Proof. By Lemma 14 we know that for a global predicate, there is a valid feature f' in the extended traces. Also, by Lemma 14 we know that the feature can be defined in a way such that the *lifted* argument is the *identinfier* for each partial pattern. Therefore, for all deleting effects, only the *lifted* argument must be given. Then, any argument in a deleting effect that is

not used to add a global argument is unnecessary and can be recovered. This holds by Theorem 5 for all actions, where the deleted grounding was added beforehand.

Then we can show that unnecessary arguments of positive preconditions can also be found. For every precondition of p there is a precondition for p', for which the lifted argument is appended. For the precondition p', again the lifted argument can be used as identifying position. Then, the arguments of the precondition are unnecessary if they are not needed to add a global predicate. By Theorem 4 this is a valid precondition and for each state where the precondition holds, there is a grounding g, if the grounding was added beforehand. From this grounding, the unnecessary arguments of the precondition can be concluded. □

To reduce the computational effort of this preprocessing, it will be shown that any global predicate can be reduced to predicates of arity 1.

Theorem 8. *A global predicate of arity $n \geq 2$ can be described by n global predicates of arity* 1.

Proof. Let there be a *global* predicate $p(x_1, ..., x_n)$ with $n \geq 2$. Then for each position i of the predicate, a new predicate p_i can be defined. Each action $a(x_1, ..., x_m)$ that adds the predicate p for grounding $p(x_{p_1}, ..., x_{p_n})$ then also adds the predicate $p_i(x_{p_i})$. This is also done for deleting effects and preconditions. Therefore, a grounding g is added for p_j if a grounding $g' = \langle g'_1, ..., g'_n \rangle$ with $g = g'_j$ is added for p. The same holds for the deleting effects. Now, if the grounding $p(c_1, .., c_n)$ is true in the initial state, then also $p_j(c_j)$ for $1 \leq j \leq n$.

Since p is a global predicate, only one grounding g can be true for p in each state s. Since p_j is added iff p is added and deleted iff p is deleted, p_j can only be true for one grounding. Therefore, p_j is a global predicate. Now for every position of the predicate $p(x_1, ..., x_n)$ there is a *global* predicate p_j. □

Now, every *global* predicate can be described as potentially multiple unary predicates. Therefore, it is not necessary to search for global features of an arity greater than 1.

5.8 Combining Identifier and Global Features

The goal is now to use the *identifying* and *global* features to find new arguments for an input trace. This is done by first searching for the *global* features and then for the *identifying* features. Before searching for *identifying* features, the arguments of the admissible global features are added to the argument lists.

The global features will first be searched, since it is likely that these arguments are also used in an *identifier* feature. An example can be found in the blocks world domain with 4 actions. In this domain there exists the global predicate *grabbed* that states which argument is currently grabbed. The action *stack* has a precondition on this predicate and therefore it is an additional

argument for the action. The added argument is then used by the action *stack* as an adding effect of the *identifying* predicate *on*. The argument is for the *identifying* predicate necessary, but in this case not, since it can be recovered by a global feature. This can be seen in more detail in Section 6.

This changes which arguments are necessary and which are unnecessary. Since global features will first be searched for, the necessary arguments for them need to be given. Then there will also be a set of necessary arguments for the *identifying* features. These either must be contained in the action labels or need to be unnecessary arguments for the global predicates. If the arguments are unnecessary for global features, they do not need to be contained in the input since they are added by a global feature. The overall necessary and unnecessary arguments can then be defined as follows.

Definition 38 (Overall Necessary Arguments)**.** An argument is necessary if it is necessary for global features. An argument is also necessary, if it is necessary for an *identifier* features and not unnecessary for any global feature.

Definition 39 (Overall Unnecessary Arguments)**.** An argument is overall unnecessary if it is unnecessary for global features. An argument is also overall unnecessary if it is unnecessary for an *identifier* feature and not necessary for global features.

Then, using our previous results, we can show that all overall unnecessary arguments can be recovered if all overall necessary arguments are contained in the input. The arguments can only be recovered when the grounding of a precondition or deleting effect was added beforehand.

Theorem 9. *All overall unnecessary arguments can be recovered if the grounding of an effect or precondition was added on a path to the corresponding action grounding. For this, all necessary arguments must be contained in the input.*

Proof. For the global features we know that all necessary arguments are contained in the traces. Therefore, by Theorem 7, all unnecessary arguments of the global features can be found, and all arguments of action groundings that are reached via an adding action for the corresponding partial grounding, are added to the argument lists.

Then, by definition, all necessary arguments of the identifying features are contained in the input. Therefore, by Theorem 5 unnecessary arguments of *identifier* features will be found and added to the arguments lists if the corresponding grounding was added beforehand. So, all overall unnecessary arguments can be recovered. □

5.9 Implementation Details

To reduce the runtime of the algorithm, there are two optimizations that reduce the number of generated features. This will on the one hand be done by typing and on the other hand over the structure of the features.

Types

As in SIFT, types are used to reduce the number of generated features. The types are based on the objects that are applied in positions of the action arguments. This preprocessing was introduced by LOCM [6] and is the same algorithm that was used by SIFT to obtain the types. Since $identifyier$ features use ($partial$) patterns, which are defined over action positions, now typed features can be defined.

Definition 40 (Typed Identifier Features)**.** An typed $identifier$ feature $f = (t, \mathbf{A}, \mathbf{D})$ contains a type tuple $t = \langle t_1, ..., t_n \rangle$, a set of $full$ patterns $\mathbf{A}$, and a set of $partial$ patterns $\mathbf{D}$. The $full$ and $partial$ patterns must have length n and must fit the type tuple. That means that an argument in the i-th positions of a pattern p must be of type t_i. For a full pattern $a[p_1, ..., p_n] \in \mathbf{A}$ and action $a(x_1, ..., x_m)$ the argument at position i must have type t_i, that means $type(x_{p_i}) = t_i$. The same holds for all non-blank positions of partial patterns. So, for $a\langle p'_1, ..., p'_n \rangle \in \mathbf{D}$ and action $a(y_1, ..., y_m)$ there is $type(x_{p'_i}) = t_i$ if $p'_i \neq \_$.

If in a domain there are different types, this can reduce the number of features generated. This can make a difference whether the computations are feasible.

Subpatterns

By definition, a feature f can have arbitrary sets of $idenitifying$ positions $Is_f(f)$. In the following we will show that a feature f with Is_f is not needed to find identifying predicates if there are $I, I' \in Is_f(f)$ such that $I \subset I'$. If this exists in a feature, it essentially means that a grounding can be identified with positions I and also by a larger set of positions that contains I.

For example, let there be partial patterns $b[1, \_, \_]$ and $a[1, 2, \_]$. Then the first action pattern states that the full grounding can be identified from the first position in the grounding. The second action pattern states that the full grounding can be identified by a tuple of the first and second position. That the feature can be identified by the first and second position is trivial if it can already be identified by the first position.

Theorem 10. *For a predicate p with $I, I' \in Is_p(p)$ such that $I \subset I'$, the set I' is not needed to identify a predicate.*

Proof. Let there be a state s such that a grounding can only be identified by the positions I' and not by the positions I. Then, for the $identifying$ positions I there must exist a state in which

the full grounding cannot be recovered by I. Therefore, there are two groundings $g, \bar{g}$ true in the same state such that $g^I = \bar{g}^I = g_{id}$. Because of this, the positions I cannot be identifying for p, therefore $I \notin Is_p(p)$. □

For a feature $f = (n, \mathbf{A}, \mathbf{D})$ with $I, I' \in Is_f$ and $I \subset I'$ all arguments must be identifiable from the positions I, the feature will not be generated. This is because there will be a feature $f' = (n, \mathbf{A}', \mathbf{D}')$ such that for any partial pattern $a\langle p \rangle \in \mathbf{D}$ with $Ip_p(p) = I'$ there is $a\langle p' \rangle \in \mathbf{D}'$ such that $Ip_p(p) = I$. For the pattern p' there is $p'_j = p_j$ if $j \in I$, otherwise there is $p'_j = \_$. That means for a partial pattern that *identifies* the full grounding over positions I' the feature now *identifies* the grounding over positions I.

6 Results

We tested the algorithm defined in Section 5 experimentally. The implementation used for the evaluation checks for an admissible feature f with $Is_f(f)$ only if there are preconditions with an identifying set of positions $I \in Is_f(f)$. The input for the algorithm were partial graphs that were sampled from the initial state of a problem P in a breath-first search manner. In the action labels of these partial graphs, there were arguments missing. The arguments that were removed from the action schemas can be seen in Table A.1 in the Appendix on page 51. The tests were done on the local cluster that has nodes with Intel(R) Xeon(R) Platinum 8352M and Intel(R) Xeon(R) Gold 6330 CPU's. Each experiment had 60 cores available.

The goal of these tests was to check whether all missing arguments are recovered. It was also tested whether additional arguments are added that were not in the domain from which the traces were drawn. The results can be seen in Table 6.1. One can see that all domains but *sokoban* had a partial graph with around 500 transitions as input. For *sokoban* there were 10.000 transitions used since also for SIFT it took a partial graph with 10.000 transitions to filter all invalid features and preconditions. For all domains in which the algorithm terminated, only the missing arguments were added to the argument lists.

In Table 6.1 it can also be seen that the algorithm did not terminate for *sokoban* and *n-puzzle* within 12 hours. The reason for this is that both domains have, after adding the global arguments, actions with an arity that is greater than 2. Additionally, in both domains, there are multiple action positions of the same type in each action. Therefore, there are many partial and full patterns that fit the same type tuple. Because of this, the number of features, for which the algorithm checks whether they are admissible, is too large.

For *sokoban* and *n-puzzle* it was checked whether the feature obtained for arity 2 are correct. For *sokoban* there is no feature learned which is correct since there is no non-static feature of arity 2 in domain. For *n-puzzle*, on the other hand, there are 111 features of arity 2 learned. None of these features adds a new arguments to an action. When manually checking these features, all of them were correct and had similar features without the blanks for the SIFT algorithm.

In the following we will first show which feature are learned for blocks world. Then it will be shown which global features for *sokoban* and *n-puzzle* are learned.

Domain	$\#A$	$\#O$	$(\#V, \#E)$	*Time*	*Correct*	*Additional*
Blocks[1] (3 Actions)	5	6	(77,504)	4.31 s	Yes	No
Blocks[2] (3 Actions)	5	6	(77,504)	2.04 s	Yes	No
Blocks (4 Actions)	3	6	(211,500)	3.09 s	Yes	No
Gripper	4	9	(159,500)	1.40 s	Yes	No
Sokoban (*restricted*)	3	16	(4139, 10002)	- (14.86 s)	-	-
N-puzzle (*restricted*)	8	14	(242, 500)	- (45.25 s)	-	-

Table 6.1: Table showing the results of the experimental evaluation, $\#A$ is the number of arguments in the action schema used to generate the traces, $\#O$ is the number of objects in the instance from which the graphs were sampled, and $(\#V, \#E)$ refers to the number of nodes $(\#V)$ and edges $(\#E)$ in the partial graph. *Time* in seconds for the algorithms to terminate (for *restricted* problems: - means not terminated in 12 hours, $(Time)$ to search for features with artiy ≤ 2), *Correct* states whether all removed arguments are recovered and *Additional* states whether there are arguments added that are not in the original labels.

6.1 Blocks World

For the blocks world domain with 3 actions, there are two reduced action schemas for which two arguments are unnecessary. The first action schema is

$$stack(x, y), unstack(x, \not{y}), move(x, \not{y}, z).$$

In both actions is the block removed, on which x is currently stacked on. For this action schema, there is no global feature learned, since in the domain there is no global predicate. Then the identifying feature

$$f_1 = (2, \{stack[1, 2], move[1, 2]\}, \{unstack[1, \_], move[1, \_]\})$$

is learned. This feature represents the *on* predicate of the blocks world domain, where the removed arguments are missing. Therefore, both removed arguments from this action schema can be recovered by this feature. There is a second feature learned that is symmetric to the feature f_1. Symmetric in this case means that for each pattern in this feature the first and second positions of the pattern are swapped. Therefore are the arguments of this feature already added by the feature f_1. These are the only admissible features and, therefore, only the removed arguments were added back to the argument lists.

The second action schema with two removed arguments is

$$stack(x, y), unstack(\not{x}, y), move(x, \not{y}, z).$$

For the *move* action the same argument position as in the first action schema is removed. For the *unstack* action, the block x is removed, and the block on which it is currently stacked is

contained in the action schema. For this action schema, the identifying predicate

$$f_2 = (2, \{stack[1,2], move[1,2]\}, \{unstack[\_,2], move[1,\_]\})$$

is learned, which is the *on* predicate without the removed arguments. From the feature f_2, now again both removed arguments can be concluded and added to the argument list. For this feature again, a symetric feature is learned that adds the same arguments, which are therefore not added.

We can see that the feature f_1 is admissible only for the first input and f_2 only for the second input. This is because the features can only be found if all of their necessary arguments are contained in the input.

For the blocks world domain with 4 actions, there is the possibility to recover all arguments for the action schema

$$pick(x), put(\not{x}), stack(\not{x}, y), unstack(x, \not{y}).$$

For this action schema first a *global* feature is learned.

$$f_3 = (1, \{pick[1], unstack[1]\}, \{put[\_], stack[\_]\})$$

This feature corresponds to the predicate *grabbed* in the domain. With this feature, the missing arguments of the actions *stack* and *put* can be recovered. This is the only global predicate, and its arguments are added to the argument lists. Afterwards, the algorithm will check whether there are admissible *identifier* features. The only admissible *identifier* feature is

$$f_4 = (2, \{stack[2,1]\}, \{unstack[1,\_]\}).$$

This feature represents the *on* relation in this domain. From this feature, the removed argument of the *unstack* action can be recovered. Therefore, all missing arguments are recovered. Here again a symmetric feature is found that does not add new arguments.

In the blocks world domain with 3 actions, it was possible to remove the lower block for the action *unstack*. In the version with 4 actions this is not done, since the upper block is necessary for the global predicate. If this argument is removed, the global feature can not be learned, since one of its adding arguments is missing.

6.2 Sokoban

For *sokoban*, only the current position of the player can be removed. This is because there is only one global predicate. Also, since there are no non-static predicates that have a higher arity as 1, there are no *identifier* features. The action schema where the current position of

the player is removed is

$$move(\sout{player}, target),\ push(\sout{player}, box, target).$$

Since the arguments are removed by a global predicate, they will be added back by the global feature

$$f_5 = (1, \{move[1], push[1]\}, \{move[\_], push[\_]\}).$$

Therefore, after searching for global predicate all arguments are already added back to the arguments lists.

Afterwards, the algorithm will check whether there are admissible $identifier$ features. For an arity 3 feature f, there can be identifying positions $Is_f(f) = \{\{1,2\},\{1,3\},\{2,3\}\}$. Any of these sets of identifying positions has 8 different partial patterns. For example, for $I = \{1,2\}$ there are $push[1,2,\_]$, $push[1,3,\_]$, $push[2,1,\_]$, $push[2,3,\_]$, $push[3,1,\_]$, $push[3,2,\_]$, $move[1,2,\_]$, and $move[2,1,\_]$. Therefore, for every set of identifying positions, there are 2^8 many different combinations of deleting actions. Since in sokoban all action positions have the same type, all of the identifying positions can be combined. For $Is_f(f) = \{\{1,2\},\{1,3\},\{2,3\}\}$ there are then $2^8 \times 2^8 \times 2^8 = 2^{24}$ many possibilities to combine the different deleting actions. These are only combinations of deleting actions for a feature where the set of identifying positions is $Is_f(f) = \{\{1,2\},\{1,3\},\{2,3\}\}$. There are also 2^6 many different combinations of adding action patterns. This is because there are 6 permutations of the arguments of the action $move(x,y,z)$. Therefore, only for the features f with $Is_f(f) = \{\{1,2\},\{1,3\},\{2,3\}\}$ there must be checked for $2^6 \times 2^{24} = 2^{30}$ features whether they are admissible.

6.3 N-puzzle

For *n-puzzle* 2 arguments need to be given per action such that all removed arguments can be recovered. First, the tile that is moved needs to be given. Second, the new coordinate of the blank needs to be given. For an up/down action this is the y coordinate, and for left/right this is the x coordinate. Then for the following actions schema all arguments can be recovered:

$$move\_up\big(tile, \sout{tile_x}, tile_y, \sout{blank_y}\big),\ move\_down\big(tile, \sout{tile_x}, tile_y, \sout{blank_y}\big),$$

$$move\_left\big(tile, tile_x, \sout{tile_y}, \sout{blank_x}\big),\ move\_right\big(tile, tile_x, \sout{tile_y}, \sout{blank_x}\big).$$

For this action schema, there is a global feature learned that describes the current x coordinate of the blank. This is because the current x coordinate of a blank was the x coordinate of the last tile that was moved before the action was applied. The corresponding learned feature is

$$f_5 = (1, \{move\_left[2], move\_right[2]\}, \{move\_left[\_], move\_right[\_]\}).$$

A similar feature is learned for the actions *move_up*, *move_down*. From this feature, the current y coordinate of the blank can be concluded. Both features will add a new argument to each action. For the contained actions, the arguments are added by the deleting action, and for the other actions the arguments are added by preconditions.

From this it follows that all the arguments are already contained in the arguments lists. To show this, we use the action $move\_up(tile, \cancel{tile_x}, tile_y, \cancel{blank_y})$ as an example. Now the current x and y location of a blank is added. Therefore, we know that $blank_y$ is contained in the input, as it describes the current y coordinate of the blank. We also know that the arguments of $tile_x$, which describe the current x coordinate of the tile, are contained in the input. This is because the tile is moved up by the action, and therefore the x coordinate of the blank and the x coordinate of the moved tile must be the same. Therefore, both removed arguments are recovered. This applies to all actions in the domain.

So, all the removed arguments can be recovered by global features. Therefore, these arguments are already added when searching for $identifier$ features. Because of this there are 4 actions, where each action has arity 4. Then for all possible $identifier$ features, it is checked whether they are admissible. That leads to a point where the calculations will be infeasible, since there are too many action patterns. Therefore, the algorithm will find all the arguments but cannot check if there are admissible $identifyier$ features that add more arguments.

7 Conclusion

The SIFT algorithm is an efficient algorithm that learns action models from traces. In this thesis we have explained how SIFT extracts constraints from traces. For this, it was proved that the coloring algorithm is correct.

In this thesis a preprocessing for the SIFT algorithm was introduced, which can find additional arguments in the traces. For this, *identifying* and *global* predicates were introduced. For these predicates, the grounding of an atom can be concluded from the context of an action grounding without knowing the full grounding. Then it was defined which arguments of the *identifying* and *global* predicates do not need to be contained in the input but can be recovered. For this, necessary and unnecessary arguments were introduced, where unnecessary arguments can be learned when the corresponding necessary arguments are contained in the input.

To find *global*/*identifying* predicates in traces, *identifying* features were introduced. Then, constraints were introduced that define when a feature is admissible. Whether these constraints are fulfilled for a feature in a trace can be checked with a simple labeling algorithm.

It was proven that for all *global*/*identifying* predicates the algorithm finds an *identifier* feature if all the necessary arguments of the predicate were contained in the traces. That means if there are all necessary arguments of a *global*/*identifying* feature are contained in the input, the unnecessary arguments of this grounding can be recovered. Therefore, the algorithm is complete.

There is no soundness result for the algorithm, therefore it may be possible that the algorithm finds additional arguments for actions, that are not contained in the original action grounding. Also, it was shown that extended traces are more informative than non-extended traces.

In the experimental evaluation of the algorithm, it has been shown that the algorithm is not feasible for the domains *n-puzzle* and *sokoban*. For these domains, there are too many features for which it needs to be checked whether they are admissible. The experiments also showed that for the domains where the algorithm terminated, all and only the arguments that were removed from the argument lists are recovered.

7.1 Future Work

The current version of the algorithm is limited by its complexity. Therefore, it is interesting whether the complexity can be reduced. This can be done by a more efficient method to extract the constraints from the traces. In the current implementation, these constraints are extracted

for each feature separately. It may be possible to do this in a more efficient manner. Also, it is interesting whether there exists a stopping criterion. When there is a stopping criterion, it does not need to be checked for each feature whether it is admissible.

Another question is whether, for example, *n-puzzle* can be learned from even fewer arguments. This is because, when a state of the *n-puzzle* is known, it is clear which action is applicable for each action name. That means that all arguments of an applicable action grounding can be concluded from the current state. To learn the *n-puzzle* from an input like this, the learning algorithm would need to know the structure of the domain.

A Appendix

Domain	Action Schemas
BlocksWorld[1] (3 Actions)	$stack(x, y)$, $unstack(x, \cancel{y})$, $move(x, \cancel{y}, z)$
BlocksWorld[2] (3 Actions)	$stack(x, y)$, $unstack(\cancel{x}, y)$, $move(x, \cancel{y}, z)$
BlocksWorld (4 Actions)	$pick(x)$, $put(\cancel{x})$, $stack(\cancel{x}, y)$, $unstack(x, \cancel{y})$
Gripper	$move(\cancel{from}, to)$, $pick(obj, \cancel{room}, gripper)$, $drop(\cancel{obj}, \cancel{room}, gripper)$
Sokoban	$move(\cancel{player}, target)$, $push(\cancel{player}, box, target)$
N-puzzle	$move\_up(tile, \cancel{tile_x}, tile_y, \cancel{blank_y})$, $move\_down(tile, \cancel{tile_x}, tile_y, \cancel{blank_y})$, $move\_left(tile, tile_x, \cancel{tile_y}, \cancel{blank_x})$, $move\_right(tile, tile_x, \cancel{tile_y}, \cancel{blank_x})$

Table A.1: Table which contains for each domain the removed arguments.

List of Symbols

Planning

$\mathcal{D}$	Domain
$\mathcal{I}$	Instance
P	Planning problem $P = \langle \mathcal{D}, \mathcal{I} \rangle$
$a(x)$	Action schema
$a(c)$	Action grounding
p	Predicate

Sift

$a[p]$	Action pattern / full pattern
$f = \langle n, B \rangle$	Feature
B	Set of action patterns
$f(\langle o \rangle)$	Grounding of a feature
$A_f(\langle o \rangle)$	Set that contains all action groundings that add the groundin $\langle o \rangle$ for feature f
$sign$	Sign for an action pattern $a[p]$
$\mathcal{D}_{max}$	Maximal domain description
$\mathcal{D}_L$	Learned Domain
$\chi_{\langle o \rangle}$	Coloring for grounding $\langle o \rangle$
$\chi_{\langle o \rangle}(s)$	Color of state s for grounding $\langle o \rangle$

Identifier Features

g	Grounding $\langle o_1, ..., o_n \rangle$
g_{id}	Partial grounding of a grounding $g_{id}\langle o_1, ..., o_n \rangle$ where an position o_i can either be an object or "_"
g^I	Partial grounding $g_{id} = \langle o_1, ..., o_n \rangle$ where every argument o_i for $i \notin I$ is replaced with a "_"
I	Set of identifying positions $I = \{i_1, ..., i_n\}$
$T(P)$	Trace drawn from problem P
$T'(P)$	Trace drawn from problem P with removed arguments
$T(P')$	Trace drawn from a liffted problem problem P'
$a\langle p \rangle$	Partial action pattern
$f = (n, \mathbf{A}, \mathbf{D})$	Identifier feature
$\mathbf{A}$	Set of adding full patterns $a[p]$ for an $indeitifying$ feature
$\mathbf{D}$	Set of deleting partial patterns $a\langle p \rangle$ for an $indeitifying$ feature

$Is_f(f_i)$	Set that contains all sets of identifying positions for a feature f_i
$Is_p(p_i)$	Set that contains all sets of identifying positions for a predicate p_i
$Ip_p(p_i)$	All identifying positions of a pattern (all non-blank positions)
$Ip_g(g_{id})$	All indentifying positions of a grounding (all non-blank positions)
$IG(f)$	Set of all partial groundings for the feature f
$Add^f(g_{id})$	All actions add a grounding that fits the partial grounding g_{id}
$Del^f(g)$	All actions that delete a grounding g for an $identifier$ feature f
$Def^f(g_{id})$	All actions that delete a partial grounding g_{id} for an $identifier$ feature f
$l_{g_{id}}$	Labeling of traces for partial grounding g_{id}
$l_{g_{id}}(s)$	Label of state s' for partial grounding g_{id}
$\bar{\chi}_{g_{id}}$	Combined labeling based on the colorings χ_g for all groundings g with partial grounding g_{id}
$\bar{\chi}_{g_{id}}(s)$	Label of state s for partial grounding g_{id} for the combined labeling

List of Figures

List of Tables

List of Listings

List of References

[1] D. Aineto, S. J. Celorrio, and E. Onaindia, "Learning action models with minimal observability," *Artificial Intelligence*, vol. 275, pp. 104–137, 2019.

[2] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty, "A review of learning planning action models," *The Knowledge Engineering Review*, vol. 33, e20, 2018.

[3] G. Behnke and P. Bercher, "Envisioning a domain learning track for the ipc," 2024.

[4] B. Bonet and H. Geffner, "Learning first-order symbolic planning representations from plain graphs," *CoRR*, vol. abs/1909.05546, 2019. arXiv: 1909.05546. [Online]. Available: http://arxiv.org/abs/1909.05546.

[5] S. Cresswell and P. Gregory, "Generalised domain model acquisition from action traces," in *Proceedings of the international conference on automated planning and scheduling*, vol. 21, 2011, pp. 42–49.

[6] S. N. Cresswell, T. L. McCluskey, and M. M. West, "Acquiring planning domain models using locm," *The Knowledge Engineering Review*, vol. 28, no. 2, pp. 195–213, 2013.

[7] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[8] H. Geffner and B. Bonet, *A concise introduction to models and methods for automated planning*. Springer Nature, 2022.

[9] J. Gösgens, N. Jansen, and H. Geffner, "Learning lifted strips models from action traces alone: A simple, general, and scalable solution," *arXiv preprint arXiv:2411.14995*, 2024.

[10] P. Gregory and S. Cresswell, "Domain model acquisition in the presence of static relations in the lop system," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015, pp. 97–105.

[11] C. Grundke, G. Röger, and M. Helmert, "Formal representations of classical planning domains," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 239–248.

[12] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, R. Brachman, F. Rossi, and P. Stone, *An introduction to the planning domain definition language*. Springer, 2019, vol. 13.

[13] D. M. McDermott, "The 1998 ai planning systems competition," *AI magazine*, vol. 21, no. 2, pp. 35–35, 2000.

[14] I. D. Rodriguez, B. Bonet, J. Romero, and H. Geffner, "Learning first-order representations for planning from black-box states: New results," *CoRR*, vol. abs/2105.10830, 2021. arXiv: 2105.10830. [Online]. Available: https://arxiv.org/abs/2105.10830.

[15] K. Xi, S. Gould, and S. Thiebaux, "Neuro-symbolic learning of lifted action models from visual traces," in *34th International Conference on Automated Planning and Scheduling*, 2024. [Online]. Available: https://openreview.net/forum?id=Kj86KzR4Xr.