

Explaining Symbolic Action Policies using Facet Reasoning

Jennifer Santos¹, Johannes K. Fichte¹, Daniel Gnad^{1,2}

¹ Linköping University, Sweden

² Heidelberg University, Germany

jennifer.santos@liu.se, johannes.fichte@liu.se, daniel.gnad@uni-heidelberg.de

Abstract

Learning symbolic action policies is a recent approach to solving classical planning problems. While policies are in principle human readable, understanding how exactly they solve a domain, or, more importantly, why they fail on a given instance, is challenging, as the logic expressions are often complex. We introduce a framework that allows us to analyze policies learned over description-logic features. Each policy rule classifies a state transition as good, bad, or unsolvable, based on features of the current state and of the transition. We encode the policy rules in answer-set programming (ASP), extending an established ASP formalism for classical planning. Using faceted reasoning we analyze how individual policy rules affect the space of possible plans and find issues in the policy when it fails to solve an instance. Our analysis is feasible on instances that can be solved optimally by strong classical planners.

Introduction

In classical planning, the objective is to find an action sequence that transforms an initial state into a goal state. A recent approach in generalized planning is to learn symbolic action policies that generalize across the instances of a domain and use them to guide the search process (Francès, Bonet, and Geffner 2021; Musayev et al. 2025; Bonet and Geffner 2025). Such policies consist of collections of rules that classify transitions using description-logic (DL) features (Baader et al. 2003). These features are built from concepts and roles, where concepts represent sets of objects and roles represent relations between them. Although the underlying planning problems are deterministic, a learned policy is generally non-deterministic, since it can admit several actions in a state and thus multiple plans for an instance. A policy is therefore not a single solution but a generator of plans. Although these symbolic rules are in principle human readable, this does not necessarily make them easy to understand. Even individual features can be surprisingly complex. Figure 1 shows a Boolean DL feature for the Spanner domain from a policy learned by Musayev et al. (2025). Although the expression is symbolic and follows the simple grammar of DL features, the property it captures is difficult to read. In practice, features tend to be even more complex, consisting of many sub-expressions. When a policy fails, behaves unexpectedly, or prunes necessary transitions, it can

```
b.empty(  
  c.some(  
    r.primitive(at,0,1),  
    c.some(  
      r.primitive(link,0,1),  
      c.some(  
        r.inverse(r.primitive(at,0,1)),  
        c.primitive(usable,0)))
```

Figure 1: Example of a Boolean DL feature in the Spanner domain. The feature is true if there is no object that has a usable spanner one step away from its current location.

therefore be challenging to determine which rule is responsible and what exactly the issue is. Without tool support, this requires manual inspection of rules, feature definitions, and affected transitions. While existing research has focused on how to learn and use such policies, we study the problem of analyzing, debugging, and explaining policies.

Our framework extends the `PlanPilot` tool (Gnad et al. 2025), which uses faceted reasoning to interactively and iteratively navigate the solution space of a planning task. Here, a facet is an action that occurs in some plans but not in all plans, and the number of facets can be used to quantify the flexibility in the solution space. We extend the pipeline with an encoding of the policy rules in answer-set programming (ASP), building on an established ASP formalism for classical planning, and restrict the solver to only choose actions justified by a fired rule. A rule fires if its conditions hold in state s and its effects apply over the transition to a successor state s' . By selectively enabling and disabling rule sets, we can analyze how different rules affect the plan space. We do so by treating individual rules as facets, which allows us to interactively enforce and prohibit specific rules and analyze the resulting changes in the solution space. This allows us to identify problematic rules that prune solutions.

We evaluate how policy rules restrict the solution space across instances from common IPC benchmarks (Taitler et al. 2024). Empirically, this is feasible on instances that a state-of-the-art planner can solve optimally. Using two case studies in the Spanner and Rovers domains, we show that the framework can identify overly restrictive rules. In the first case, a rule is identified that, while intended to prune transi-

tions to unsolvable states, in fact prunes valid solutions. In the second case, a rule is identified that incorrectly prunes actions necessary to solve the planning task. This process is automated and does not require human input, offering, to the best of our knowledge, the first approach for such a systematic analysis of action policies learned in planning. The identified rules provide concrete feedback for improving the learned policies: misclassified rules can be relabeled, and biases in the training data can be addressed by retraining.

Background

We review classical planning, answer-set programming, faceted reasoning, and policies learned over description-logic features.

Classical Planning

A classical planning problem is a pair $\Pi = \langle D, I \rangle$ that consists of a planning domain D and a planning instance I . The domain $D = \langle T, P, A \rangle$ consists of a set of types T , a set of predicates P , and a set of action schemas A . Each predicate $p \in P$ has a fixed arity $|p|$, and each of its parameters $\mathbf{x} = \langle x_1, \dots, x_{|p|} \rangle$ is associated with a type $\tau(x_i)$. An atom is a predicate for which each parameter is associated with a variable symbol or object. Each action schema $a \in A$ has a set of preconditions $pre(a)$ and a set of effects $eff(a)$, both of which are sets of atoms that are instantiated with variables only. The effects are partitioned into add effects $add(a)$ and delete effects $del(a)$. A planning instance is a tuple $I = \langle O, s_0, G \rangle$, where O is a set of objects with associated types. An atom is ground if it is instantiated with objects only and a state s is a set of ground atoms. s_0 is the initial state, and G is the goal condition, a set of ground atoms. A ground action is obtained by instantiating all parameters of the schema with objects $o \in O$ of matching types. A ground action a is applicable in state s if its preconditions are satisfied in s , i.e., $pre(a) \subseteq s$. Applying an applicable action yields the successor state $s' = (s \setminus del(a)) \cup add(a)$. A plan is a sequence of ground actions $\pi = \langle a_0, \dots, a_n \rangle$ whose repeated application transforms s_0 into a state s' that satisfies G , i.e., $G \subseteq s'$. We assume that actions have uniform costs, so an optimal plan is a plan with minimal length. A planning problem is solvable if there exists at least one plan. By $Plans(\Pi_H)$ we denote the set of all plans of Π of length at most H .

Planning problems are commonly specified in the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) and can be represented in ground form using SAS+ (Bäckström and Nebel 1995).

Answer-Set Programming

In Answer-Set Programming (ASP), a rule is of the form

$$h \leftarrow b_1, \dots, b_n, \text{not } c_1, \dots, \text{not } c_m$$

where h is the head of the rule and the remaining literals form the body. The rule derives h if all atoms $b_1, \dots, b_n$ hold and none of the atoms $c_1, \dots, c_m$ hold. A rule without a body is called a fact, which makes the head always true. A rule without a head is a constraint, meaning that the

body must not be satisfied. A collection of rules is called an answer-set program (Brewka, Eiter, and Truszczyński 2011). Its solutions, called answer sets, are sets of ground atoms that satisfy the program under stable model semantics (Gelfond and Lifschitz 1988). An answer-set program may have zero, one, or multiple answer sets. Beyond these *normal* rules, we make use of the following ASP constructs. A choice rule

$$\{h\} \leftarrow b_1, \dots, b_n$$

allows h to be chosen as true or false if the body is satisfied. The head can also contain a conditional literal $h : d$, where h is eligible to be chosen if d holds. Bounds can be added to restrict how many eligible head atoms may be chosen:

$$l \{h : d\} u \leftarrow b_1, \dots, b_n$$

where l is the lower bound and u is the upper bound, with $0 \leq l \leq u$. The `#minimize` statement selects answer sets that minimize a given objective, and the aggregate `#count` counts the number of elements satisfying a condition.

Faceted Reasoning

The notion of facets has been employed in ASP to navigate answer sets (Alrabbaa, Rudolph, and Schweizer 2018; Fichte, Gaggli, and Rusovac 2022). A facet is an atom that occurs in some answer sets of the program but not in all of them, and this property makes it possible to refine the solution space. Enforcing a facet keeps only the answer sets in which the corresponding atom holds, while prohibiting it keeps only the answer sets in which it is false. In classical planning, this notion has been adapted by treating action occurrences as facets, allowing users to efficiently restrict and analyze the set of possible plans (Speck et al. 2025; Gnad et al. 2025). The advantage of facet reasoning is that deciding whether an atom is a facet lies in NP and is therefore computationally much easier than counting the number of polynomially bounded plans (#P), deciding whether a given number of such plans exists (PP), or enumerating all such plans (EnumP) (Speck et al. 2025). The significance of a facet is measured by the reduction in remaining facets when the facet is enforced or prohibited.

Description Logic

Description logics (DL) are a family of formal languages used to represent knowledge about a domain by defining concepts and roles (Baader et al. 2003). Given a planning domain D , atomic concepts correspond to unary predicate symbols and types, and atomic roles correspond to binary predicate symbols. In this work, we interpret these over a universe Δ defined over the objects O of a planning instance. To express more complex relations, concepts $\mathcal{C}, \mathcal{D}$ and roles $\mathcal{R}, \mathcal{S}$ can be constructed from atomic ones. We use the following constructors from the DLPlan library (Drexler, Francès, and Seipp 2022): universal concept $\top$, bottom concept $\perp$, intersection $\mathcal{C} \sqcap \mathcal{D}$, negation $\neg \mathcal{C}$, value restriction $\forall \mathcal{R}. \mathcal{C}$, existential quantification $\exists \mathcal{R}. \mathcal{C}$, equality $\mathcal{R} = \mathcal{S}$, inverse $\mathcal{R}^{-1}$, transitive closure $\mathcal{R}^+$, and restriction $\mathcal{R}|_{\mathcal{C}}$.

DL features are expressions built from these constructors and are evaluated over planning states. The complexity of a

feature is determined by the number of constructors in its expression. We consider Boolean features defined by concept or role emptiness or by the truth value of a nullary predicate, and numerical features defined by the cardinality of a concept $\mathcal{C}$ or a role $\mathcal{R}$.

Symbolic Action Policies

Action policies can be learned from sets of Boolean and numerical DL features, which allows generalization across instances of a planning domain. For example, Francès, Bonet, and Geffner (2021) learn general policies by solving a Weighted Max-SAT problem. Bonet and Geffner (2025) build on hitting-set algorithms to learn general policies. Similarly, Musayev et al. (2025) learn transition classifiers over DL features using decision trees, distinguishing between good, bad, and unsolvable transitions. In their work, a good transition decreases the goal distance, a bad transition does not decrease the goal distance, and an unsolvable transition leads to an unsolvable state.

Both general policies and transition classifiers consist of rules that distinguish preferred transitions from undesired transitions and can therefore be used to guide the search. Such a rule consists of a set of feature conditions C and a set of feature effects E , both based on description logic. The conditions are evaluated in the current state s , and the effects are evaluated over the transition from s to a successor state s' . A rule fires for a transition if all conditions in C hold in s and all effects in E hold over the transition from s to s' .

Although the underlying planning problems are deterministic, the learned policies are generally non-deterministic: a state may admit several transitions that are classified as good, so a policy can allow multiple plans for an instance. A policy is therefore not a single solution but a generator of plans. In particular, such a policy is not a policy in the sense of fully observable non-deterministic (FOND) planning, where actions are selected to cope with non-determinism in the environment; here, the non-determinism lies in the permissiveness of the policy itself.

We consider policy rules with Boolean feature conditions that evaluate to true or false in state s . For feature effects, both Boolean and numerical features are considered. A Boolean effect is positive if the feature changes from false to true over the transition, and negative if it changes from true to false. Numerical effects describe how the feature value changes over the transition. The possible comparisons are strict increase, strict decrease, increase or equal, decrease or equal, and no change.

Encoding Policies in ASP

To evaluate symbolic action policies over a planning problem $\Pi = \langle D, I \rangle$, we first translate Π into ASP. Then we extend the ASP encoding with DL features and policy rules.

Planning as ASP

We use the translator `plasp` (Dimopoulos et al. 2019), which takes a PDDL or SAS+ planning problem with a planning horizon H as input and produces an answer-set program. We use a bounded plan encoding, in which a plan may

have at most H steps. To represent plans shorter than H , the encoding introduces `no-op` actions to ensure that each time step is assigned exactly one action. The placement of the `no-op` actions is regulated: they cannot occur at the final time step, and if there are one or multiple `no-ops`, they must appear consecutively right before the final step. In this program, planning states are represented by `holds` atoms indexed by time steps $T \in \{0, 1, \dots, H\}$.

Under these adaptations, a ground action a of a planning problem Π is a facet at time step T if it occurs at time step T in some plan in $Plans(\Pi_H)$ with horizon H , but not at time step T in all plans in $Plans(\Pi_H)$. A `no-op` action is a valid action and is therefore counted as a facet.

Encoding Symbolic Policies in ASP

Our encoder takes a set of rules per transition class (good/bad/unsolvable) as input, where each rule is defined over Boolean and numerical DL features. To extract the syntactic structure of each rule, we use the `DLPlan` library (Drexler, Francès, and Seipp 2022), which parses each rule into a constructor tree. The root of this tree identifies the rule and its class label, while its children represent the conditions and effects of the rule. Each condition or effect specifies how its associated feature is evaluated, for example whether a numerical feature decreases, and points to the feature it evaluates. The feature is then recursively decomposed into its DL constructors, ending in leaves that represent primitive concepts and roles defined over the domain predicates. Figure 2 shows an example of this representation for a rule classified as bad in the Sokoban transition classifier learned by Musayev et al. (2025).

After parsing, the constructor tree is translated into ASP facts, where each node specifies the node type, its index, and the indices of its child nodes. The semantics of the DL constructors are defined by a domain-independent set of ASP rules included by the encoder (see Example 1), with both static and dynamic variants to account for time-dependent and time-invariant domain predicates.

Example 1 *The constructor for the intersection $\mathcal{C} \sqcap \mathcal{D}$ is encoded in ASP as follows:*

```
concept (Idx, X, T) :-
  and_concept (C, D, Idx),
  concept (C, X, T),
  concept (D, X, T).
```

The rule derives that object X belongs to the intersection concept (“and”) with index Idx at time step T if X belongs to both $\mathcal{C}$ and $\mathcal{D}$ in T , given that `and_concept (C, D, Idx)` exists in the constructor tree.

To evaluate the features and rules over planning states, we connect them with the ASP planning encoding via bridge rules. We define a universe Δ consisting of the objects O of the planning instance I and the constants of the planning domain D . Each dynamic predicate is then grounded over Δ , and a bridge rule links every grounded predicate to the corresponding `holds` atom generated by `plasp`. Finally, static and goal predicates are added as ASP facts.

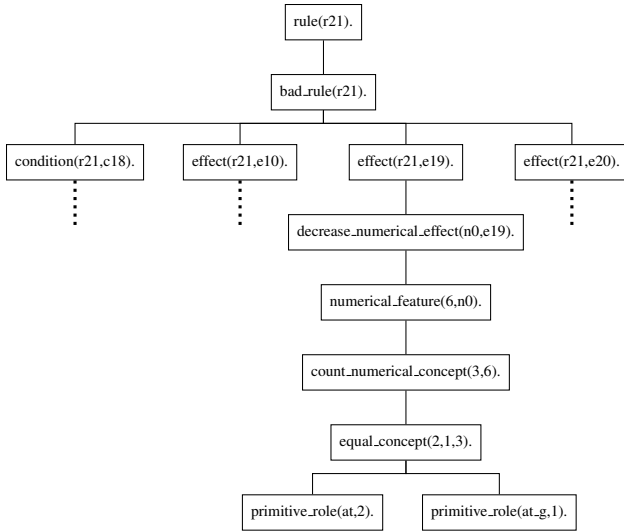


Figure 2: The constructor tree of a rule classified as bad in the Sokoban domain with one effect branch expanded. The dotted edges indicate omitted conditions and effects.

With this encoding, a planning task can be solved under rule guidance, where an action can occur at a time step T only if there exists a policy rule that fires along the transition. We integrate the policy such that the user can selectively enable or disable individual rules or entire transition classes. A rule R is enabled by adding the ASP fact `enabled(R)`, and is disabled by default otherwise.

A rule fires at time step T if it is enabled, its conditions hold at time step T , and its effects hold over the transition to time step $T + 1$. See Example 2 for how a feature effect is evaluated. Furthermore, a rule must fire at every state from which an action is executed, and every executed action must be preceded by a fired rule. The exception is a `no-op` action: a rule is not required to fire at time step T if a `no-op` action occurs at time step $T + 1$.

Example 2 *The numerical decrease effect is defined in our ASP encoding as follows:*

```
effect_satisfied(E,T,T1) :-
  decrease_numerical_effect(N,E),
  T1 = T+1,
  numerical(N,V0,T), numerical(N,V1,T1),
  V0 > V1.
```

The effect E holds over the transition from time step T to $T + 1$ if the value V of the numerical feature N strictly decreases over the transition.

Framework

Our framework extends the PlanPilot tool (Gnad et al. 2025), which allows users to navigate the plan space of a planning task by treating action occurrences as facets. It takes a PDDL or SAS+ planning problem, together with a planning horizon H , and uses the tool `plasp` (Dimopoulos et al. 2019) to translate the problem into ASP. The ASP planning problem is then passed to `fasb` (Fichte, Gaggl, and

Rusovac 2022), which uses `clingo`¹ as its solver. In the interactive mode, the user can enumerate and count facets and plans, query the significance of a facet, and iteratively restrict the plan space by enforcing or prohibiting facets. Each query, including counting facets after a facet has been enforced or prohibited, is answered by a separate solver run on the updated program; the plan space is not enumerated once and then reused. The cost of an interactive step therefore corresponds to a single facet computation, in the range of the runtimes reported in our experiments. The tool also supports a script mode, where the user provides a file of query commands to be executed.

We extend PlanPilot with two analysis modes. In the first mode, the ASP program generated by `plasp`, together with the encoded policies, is solved directly by `clingo` without faceted reasoning. This mode is used for batch experiments, where many planning tasks and rule configurations are evaluated automatically. In the second mode, the problem is passed to PlanPilot, either through scripted queries or through the interactive interface. This mode supports a more detailed incremental inspection of how rule choices affect the remaining plan space.

In both modes, the rule configuration can be controlled at different levels. First, complete transition classes (good/bad/unsolvable) can be enabled or disabled. Second, a transition class can be minimized, where `clingo` selects a minimum subset of the rules of that class that still admits a solution. Third, in the interactive setting, individual rules can be treated as facets by making them optional, e.g., `{enabled(R)} :- rule_bad(R)`. When a rule is enabled in some plans but not in all plans, the user can enforce or prohibit this rule using its identifier. This allows us to analyze how individual rules affect the plan space. Note that enabling a rule only permits it to fire and thereby justify transitions; it does not enforce its use. To find exactly the plans allowed by a policy, we enforce that good rules fire for every transition and that bad/unsolvable rules are prohibited from firing.

Configuration Our framework supports arbitrary policies defined over DL features. We showcase possible analyses on the transition classifiers learned by Musayev et al. (2025) with default parameters. These rules are grouped into the three transition classes: good, bad, and unsolvable. Table 1 shows the number of rules per class and domain of the IPC learning track (Taitler et al. 2024). Based on these classes, we evaluate five ASP configurations:

- *ASP*: base encoding generated by `plasp`.
- *ASP_g*: *ASP* with all good rules enabled.
- *ASP_{g,b}*: *ASP_g* with all bad rules enabled.
- *ASP_{g,b̃}*: *ASP_g* with a minimized subset of bad rules enabled.
- *ASP_{g,b,u}*: *ASP_{g,b}* with all unsolvable rules enabled.

ASP_{g,b̃} uses `clingo` in optimization mode, finding the minimum number of bad rules that need to be allowed such that the instance is solvable.

¹<https://github.com/potassco/clingo>

Domain	Good rules	Bad rules	Unsolvable rules	Total
blocksworld	12	8	0	20
childsnap	16	17	7	40
ferry	7	13	0	20
floorile	7	10	3	20
miconic	5	5	0	10
rovers	11	9	0	20
satellite	22	28	0	50
sokoban	3	6	1	10
spanner	23	15	12	50
transport	5	5	0	10

Table 1: Number of rules per domain, divided into good, bad, and unsolvable transition classes.

Case Study

We present two case studies that illustrate how our framework can be used to diagnose specific failures of policies. In both case studies, we analyze the transition classifiers learned by Musayev et al. (2025) with default parameters, i.e., the same classifiers as in our experiments. The first, in the Spanner domain, examines why unsolvable rules prune valid plans. The second, in the Rovers domain, identifies a rule that incorrectly prunes actions necessary to solve the task.

In the case studies, we make rules of one class, either bad or unsolvable, optional in interactive mode. This yields two types of facets: action facets and rule facets. The total number of facets therefore counts both types. To make it explicit that we count only action facets, we refer to their number as the *action facet count*.

Spanner

In the Spanner domain, a worker Bob walks from a shed sh to a gate gt along a corridor of ℓ connected locations $l_1, \dots, l_\ell$. On the way, Bob can pick up spanners $s_1, \dots, s_m$ placed at specific locations and use them to tighten the nuts $n_1, \dots, n_k$ at the gate. There are three actions: $w(x, y)$ walks Bob from location x to y ; $pu(s)$ picks up a spanner s at the current location; and $tn(s, n)$ uses spanner s to tighten nut n . As Bob can only move forward and each spanner can only be used once, Bob must pick up enough spanners before reaching the gate to tighten all nuts, else the goal cannot be achieved.

The transition classifier for Spanner contains 12 unsolvable rules that are intended to prevent the planner from choosing actions that lead to a dead-end state. We study how these rules affect the plan space by comparing the number of action facets under two configurations: ASP and $ASP_{g,b}$. If the unsolvable rules work as intended, no plan contains a transition that they classify as unsolvable, so every plan in ASP should also be a plan under $ASP_{g,b}$ and the number of action facets should not decrease. To analyze this in more detail, we use the interactive mode of the framework and treat unsolvable rules as facets. For each instance, we first count the number of action facets under ASP and $ASP_{g,b}$ configurations. If there is a reduced facet count, we iteratively prohibit one unsolvable rule at a time under

Instance	$\sum f_{ASP}$	$\sum f_{ASP_{g,b}}$	Prohibited rule	Rule impact
easy p05	22	12	r126	100 %
easy p10	108	80	r143	≈ 14 %
easy p12	110	102	r126	100 %
easy p15	228	216	r126	100 %
easy p16	308	278	r145	≈ 27 %
easy p18	310	292	-	-
easy p20	498	426	r143	≈ 11 %
easy p21	508	430	r143	≈ 18 %
easy p22	634	534	r143	10 %
easy p23	660	628	r126	100 %
easy p24	662	542	r143	≈ 23 %

Table 2: Action facet count $\sum f$ for instances of the Spanner domain under the base encoding ASP and the $ASP_{g,b}$ configuration. The reported percentage is the share of the total action facet reduction caused by the prohibited rule.

$w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s1) \rightarrow pu(s2) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s1, n1)$
 $w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s1) \rightarrow pu(s2) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s2, n1)$
 $w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s2) \rightarrow pu(s1) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s1, n1)$
 $w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s2) \rightarrow pu(s1) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s2, n1)$
..... pruned plans
 $w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s1) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s1, n1)$
 $w(sh, l1) \rightarrow w(l1, l2) \rightarrow w(l2, l3) \rightarrow pu(s2) \rightarrow w(l3, l4) \rightarrow w(l4, gt) \rightarrow tn(s2, n1)$

Figure 3: All plans after solving instance *easy p05* with horizon $H = 10$ under the ASP configuration. The plans under the dotted lines are pruned when solving the same instance under the $ASP_{g,b}$ configuration.

the $ASP_{g,b,u}$ configuration and count the remaining action facets. The analysis is run with a time limit of 30 minutes per instance, and instances exceeding this limit are excluded. The results are presented in Table 2.

For all instances except *easy p18*, only a single unsolvable rule reduces the action facet count, while prohibiting any of the remaining rules leaves the count unchanged. Of the 12 unsolvable rules, only three restrict the plan space individually. Among these three, r126 alone accounts for 100 % of the facet reduction in four instances. In the *easy p05* instance, all 10 facets pruned by r126 are $w(x, y)$ actions. Figure 3 shows the six plans under the ASP configuration, two of which are pruned under $ASP_{g,b}$, leaving four plans. Prohibiting r126 under the $ASP_{g,b,u}$ configuration prunes the same two plans, confirming that r126 is responsible for the reduction. In the two pruned plans, Bob picks up only one of the two usable spanners before walking away. As a result, the other spanner remains at an unreachable location. In contrast, in the remaining plans, Bob collects both spanners before moving on, although only one is needed to reach the goal. The same pattern appears in *easy p12*. Four spanners are distributed equally across two locations, and the remaining plans after prohibiting r126 are those where both spanners nearest to the gate are picked up. Looking closer at r126, it consists of two numerical effects, n87 and n94:

- n87 counts the number of objects that Bob left behind along the corridor, which are unreachable to Bob.
- n94 counts the number of objects that have only loose objects ahead of themselves.

In the examples in Figure 3, when Bob moves forward leaving a spanner behind, the value of `n87` increases. At the same time, if that spanner previously had only loose nuts ahead of itself, Bob now breaks this property, since Bob is not loose, which decreases the value of `n94`.

Overall, we conjecture that the training data used for learning the transition classifiers may contain a bias toward instances in which the critical distinction is whether Bob collects all spanners at the last spanner location before reaching the loose nut(s). In such instances, unsolvable transitions correspond to Bob moving toward the gate without picking up a required spanner. Consequently, `r126` may overfit to this pattern and prune transitions that are in fact part of plans, because there are sufficiently many spanners.²

Rovers

In the Rovers domain, a set of rovers moves between waypoints to collect three types of data: soil samples, rock samples, and camera images. The goal is to communicate the required data to a stationary lander. There are nine actions: `navigate` to move between waypoints, `sample_soil` and `sample_rock` to collect samples, `drop` to empty a sample store, `calibrate` to calibrate a camera, `take_image` to capture an image, and three communicate actions to transmit the respective data.

The transition classifier for Rovers contains 20 rules in total: 11 good rules and 9 bad rules. We first analyze the effect of enabling only the good rules by counting action facets under the ASP and ASP_g configurations. The planning horizon is set to the optimal plan length for each instance, since larger horizons may allow redundant actions, and we use a time limit of 30 minutes per instance. If no plan is found under the ASP_g configuration, resulting in a facet count of zero, the good rules alone are incomplete and exclude transitions necessary to find plans. To investigate these cases further, we use the interactive mode of the framework and treat bad rules as facets. We enable all good rules and then iteratively prohibit one bad rule at a time. For each prohibited rule, we count the remaining action facets. This allows us to identify which bad rule is necessary for finding plans.

The results are presented in Table 3. Under ASP_g , no action facets remain, i.e., all plans are pruned, so the good rules alone are incomplete. The subsequent facet analysis identifies the bad rule `r74` as the cause: prohibiting `r74` removes all remaining plans, i.e., `r74` is required along every plan and accounts for the full facet reduction in all instances. Additional bad rules also have an effect in *easy p04* and *easy p08*. To understand why `r74` is necessary, we manually inspect the plans of *easy p01*, *easy p05*, and *easy p09* and find that every occurrence of the actions `drop`, `sample_rock`, `sample_soil`, `calibrate`, and `take_image` is classified as bad. This implies that `r74` does not properly distinguish necessary transitions from bad ones, causing `r74` to classify several necessary actions as bad. Relabeling the rule as good fixes the issue in all tested instances.

²Out of curiosity we checked a few training instances and could confirm this pattern, indicating a weakness in the training data.

Instance	H_{opt}	$\sum f_{ASP}$	$\sum f_{ASP_g}$	Prohibited rule	Rule impact
easy p01	9	84	0	r74	100 %
easy p02	16	396	0	r74	100 %
easy p04	16	186	0	r71	100 %
easy p04	16	186	0	r74	100 %
easy p05	6	14	0	r74	100 %
easy p08	19	346	0	r69	≈ 8 %
easy p08	19	346	0	r74	100 %
easy p08	19	346	0	r76	100 %
easy p09	6	48	0	r74	100 %

Table 3: Action facet count $\sum f$ for Rover instances under the base encoding ASP with the optimal planning horizon H_{opt} . The rule impact reflects how much of the action facet count under ASP is reduced when a bad rule is prohibited.

Experiments

We evaluate the performance of the different ASP encodings and analyze how learned policies affect the plan space.

Experimental Setup

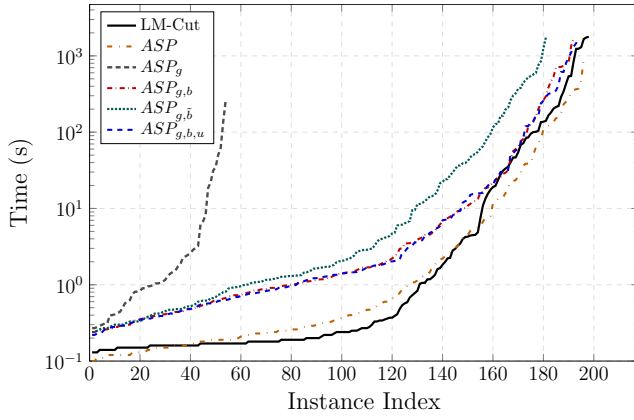
We evaluate our approach on the IPC 2023 benchmarks (Taitler et al. 2024), consisting of 10 domains with 90 instances each, grouped into easy, medium, and hard categories. All experiments are conducted using the Lab toolkit (Seipp et al. 2017) on a compute cluster with Intel Xeon Gold CPUs, with a time limit of 30 minutes and a memory limit of 4 GiB per instance. As a baseline, we use A* with the LM-Cut heuristic (Helmert and Domshlak 2009) in FastDownward (Helmert 2006). Instances not solved by the baseline are excluded from the evaluation.

The baseline also provides the optimal plan lengths used to derive the ASP planning horizons. Specifically, the optimal plan length is multiplied by 1.5 and rounded down to the nearest integer. This allows us to analyze how much rule guidance restricts the plan space beyond optimal plans. We measure this restriction by the number of facets. In these experiments, every facet is an action facet, and the *facet count* therefore corresponds to the number of action facets.

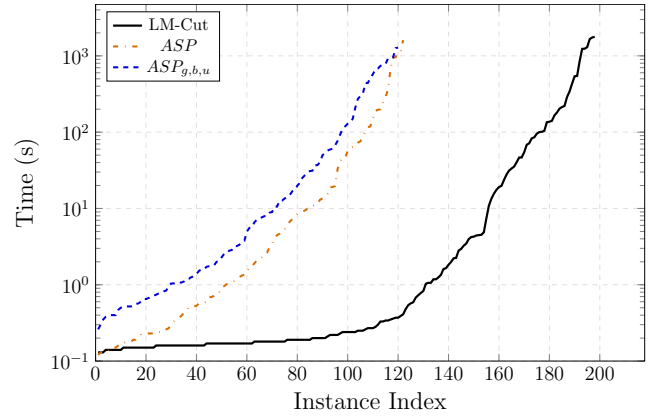
Performance

We first evaluate if the generated ASP instances can be solved by `clingo`. The objective is to find one plan under the bounded encoding with the selected rule configuration. Columns “solved”, in Table 4, show the number of solved instances for each ASP configuration and the LM-Cut baseline. The base encoding ASP solves nearly as many instances as the baseline, showing that the ASP encoding itself remains feasible. Enabling rules reduces coverage across all configurations, which can be attributed to both an overhead during solving and pruning of plans.

We show the runtime behavior of the ASP configurations in comparison to the baseline in Figure 4a. The most notable deviation is observed for ASP_g , where the runtime grows faster and stagnates earlier than the other configurations, indicating that the good rules impose strong constraints. The remaining configurations mainly follow the same trend as



(a) Runtime for solving one instance with `clingo`.



(b) Runtime for computing facet counts.

Figure 4: Runtime comparison between LM-Cut and the ASP configurations.

Domain	Solved					Facet computation				
	LM-Cut	ASP	ASP _g	ASP _{g,b}	ASP _{g,b̃}	ASP	ASP _g	ASP _{g,b}	ASP _{g,b̃}	ASP _{g,b,u}
blocksworld	11	11	5	11	11	11	5	11	11	11
childsnaek	9	9	9	9	9	3	8	3	3	3
ferry	20	20	12	20	20	13	12	10	10	10
floortile	10	10	6	10	10	0	2	0	0	0
miconic	37	36	0	35	36	22	0	24	24	24
rovers	18	18	0	17	17	9	0	9	9	9
satellite	23	23	13	22	22	13	10	12	12	12
sokoban	30	29	0	28	29	17	0	17	17	17
spanner	30	30	6	30	30	30	6	30	30	30
transport	10	10	3	10	10	4	3	4	4	4
Total	198	196	54	192	194	122	46	120	120	120

Table 4: Number of solved instances and successful facet computations per domain. The LM-Cut column reports the baseline coverage. The “solved” columns report coverage for each ASP configuration and the “facet computation” columns report the number of instances for which facet counts are successfully computed.

the baseline, with overhead introduced by the transition classifiers. An exception is $ASP_{g,b̃}$, where the minimization of the bad rules requires additional iterations to identify the minimal subset, resulting in increased runtime.

Facet Analysis

We analyze the runtime of facet counting under the ASP configurations and the number of facets to quantify how strongly the rules restrict the plan space. Table 4 shows that computing the number of facets is more demanding than finding a single plan. While it is possible to find plans for 196 instances under the base encoding ASP , facet counts can only be computed for 122 of them. A similar pattern appears for $ASP_{g,b}$ and $ASP_{g,b,u}$, where facet counts can be computed for 120 of 192 instances. For ASP_g , facet counts are successfully computed for 46 of the 54 solved instances; on the remaining instances, the computation does not terminate within the time limit. Although all configurations except ASP use rule guidance, the number of instances with successful facet computations remains close to

Domain (nr. of instances)	$\sum f_{ASP}$			$\sum f_{ASP_g}$		
	Avg	Min	Max	Avg	Min	Max
blocksworld (5)	1243.20	158	2718	188.80	0	564
childsnaek (3)	3588.00	3524	3716	1108.67	1088	1142
ferry (10)	7296.00	664	19614	306.00	32	866
satellite (10)	3074.20	448	7342	551.60	58	1200
spanner (6)	211.67	0	1214	59.00	0	354
transport (3)	792.00	44	1178	5.33	0	8

Table 5: Facet count $\sum f$ for the base encoding ASP and the corresponding reduction by enabling good rules (ASP_g). Results are reported with average, minimum, and maximum facet counts per domain, computed over instances solved by both ASP and ASP_g .

that of ASP .

Table 5 reports the average, minimum, and maximum number of facets per domain, computed over instances where a solution was found under both the ASP and ASP_g configurations. Figure 5 visualizes the distribution of facet counts per domain. Across all domains, enabling good rules consistently reduces the number of facets compared to the base encoding ASP . This can be explained by the choice of horizon which is set to 1.5 times the optimal plan length. As a result, the plan space of the ASP configuration may contain redundant actions that are selected only to fill out the horizon, and are therefore excluded by the good rules.

The runtime for computing the number of facets under the ASP and $ASP_{g,b,u}$ configurations is shown in Figure 4b, together with the LM-Cut baseline. Note that all domains that do not have unsolvable rules are still included in the $ASP_{g,b,u}$ configuration. For both ASP and $ASP_{g,b,u}$, the runtime increases faster than for the baseline. This is expected, since facet counting requires reasoning about the entire plan space rather than finding one optimal solution. $ASP_{g,b,u}$ introduces additional overhead compared to ASP , because the solver must also evaluate the encoded rules. Nevertheless, both configurations compute facet counts for a comparable number of instances within the time limit.

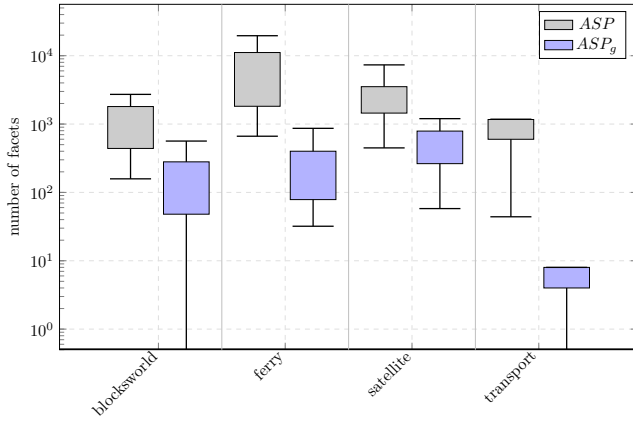


Figure 5: Distribution of facet counts per domain for ASP and ASP_g in commonly solved instances. The y-axis is in log scale, with zero facet counts plotted at the lower limit.

Related Work

As AI systems become more widely deployed throughout our society, many stakeholders are in need of systems that can *explain* their decisions to the user (Langer et al. 2021). AI Planning is no exception, and the area of Explainable AI Planning (XAIP) has recently seen increased attention (Sreedharan et al. 2019; Krarup et al. 2021; Sreedharan, Chakraborti, and Kambhampati 2021; Eifler et al. 2022). Most work in XAIP explains individual solutions of a planning system, e.g., by justifying the actions of a plan, contrasting a plan with counterfactual alternatives, or explaining why a task is unsolvable. In contrast, we explain a *policy*: we analyze how a symbolic action policy, defined over description-logic features, restricts the space of possible plans. Our explanations are comparative: we quantify which plans remain possible under a policy or a subset of its rules.

Sreedharan, Muise, and Kambhampati (2023) generalize action justification and causal links from sequential plans to policies, transferring concepts such as non-redundancy from plans to (FOND) policies. Their analysis asks whether a policy admits redundant behavior, whereas we identify rules that wrongly prune valid plans; both lines of work, however, explain policies rather than single plans.

Prior work on learning policies over description-logic features has provided detailed analyses for specific domains, such as Blocksworld or Gripper, but relied on manual interpretation of the features and policy rules (Bonet, Francès, and Geffner 2019; Bonet and Geffner 2021; Bonet, Drexler, and Geffner 2024). These investigations were not conducted on automatically learned policies, but rather the policies were handcrafted and then proven to be correct. The features used in these policies are significantly simpler, which allows defining compact policies that usually cannot be learned automatically.

Another body of work targets the verification of neural network policies (Vincent, Steinmetz, and Hoffmann 2022; Vincent, Hermanns, and Hoffmann 2026). Their analysis is

qualitative: *does a learned policy lead to unsafe behavior?* This is a question that is naturally supported by our framework by asking if an unsafe state, i.e., a state in which certain atoms hold, is reachable under the policy, which is a special case of our analyses. Yet, we are, of course, limited to symbolic policies. Action policy testing is closely related to the verification, being more efficient than a full verification process (Steinmetz et al. 2022; Eisenhut et al. 2024).

Eifler et al. (2024) analyze soft-goal conflicts given a learned neural network policy. Soft goals can be encoded in our framework as well, allowing a similar check that determines the minimal unsolvable goal subsets using repeated facet reasoning.

Finally, our framework is not limited to policies learned over description-logic features but applies to rule-based control knowledge more generally, whenever rules can be expressed over features that are evaluable in ASP. For instance, TLPlan-style search-control rules with a priority ordering (Bacchus and Kabanza 2000) could be modeled by encoding the ordering explicitly, whereas subsymbolic control knowledge requires the verification and testing approaches discussed above.

Conclusions and Future Work

We introduce a framework to analyze symbolic action policies learned over description-logic features. The framework encodes policy rules in ASP and connects them with the PlanPilot tool. This allows for systematic and interactive exploration of plan spaces using faceted reasoning under policy guidance. In particular, users can enforce and prohibit rule sets or individual rules and observe how these choices affect the plan space. The case studies demonstrate that the framework can identify concrete reasons for policy failures. In *Spanner*, an unsolvable rule prunes transitions that occur in valid plans. In *Rovers*, a bad rule classifies some necessary actions as bad, which prevents plans from being found when only good rules are enabled. We remark that most of the analysis is automated. Once problematic rules are identified, a manual inspection can provide further details about the underlying issue, such as a bias in the training data.

Beyond explaining failures, the diagnoses can directly be used to improve the learned policies: in *Rovers*, relabeling the identified rule as good fixes the issue in all tested instances, and in *Spanner*, the identified rule indicates a bias in the training data, which can be addressed by retraining.

Future work includes a more scalable policy encoding and a more detailed analysis of rule definitions. In particular, features could be treated as facets, allowing users to analyze which feature conditions or effects are responsible for a rule firing. Performance can be improved by adding additional planning information, such as mutexes or operator-counting constraints, to the `plasp` encoding.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computational resources were provided by the National

Academic Infrastructure for Supercomputing in Sweden (NAISS), funded by the Swedish Research Council.

References

- Alrabbaa, C.; Rudolph, S.; and Schweizer, L. 2018. Faceted answer-set navigation. In *International Joint Conference on Rules and Reasoning*, 211–225. Springer.
- Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.
- Bacchus, F.; and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116(1–2): 123–191.
- Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS⁺ Planning. *Computational Intelligence*, 11(4): 625–655.
- Bonet, B.; Drexler, D.; and Geffner, H. 2024. On Policy Reuse: An Expressive Language for Representing and Executing General Policies that Call Other Policies. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 31–39. AAAI Press.
- Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2703–2710. AAAI Press.
- Bonet, B.; and Geffner, H. 2021. General Policies, Representations, and Planning Width. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11764–11773. AAAI Press.
- Bonet, B.; and Geffner, H. 2025. Learning General Policies from Examples. In Ortiz, M.; Wassermann, R.; and Schaub, T., eds., *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, Melbourne, Australia, November 1-17, 2025*.
- Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12): 92–103.
- Dimopoulos, Y.; Gebser, M.; Lühne, P.; Romero, J.; and Schaub, T. 2019. plasp 3: Towards Effective ASP Planning. *Theory and Practice of Logic Programming*, 19(3): 477–504.
- Drexler, D.; Francès, G.; and Seipp, J. 2022. Description Logics State Features for Planning (DLPlan). <https://doi.org/10.5281/zenodo.5826139>.
- Eifler, R.; Brandao, M.; Coles, A. J.; Frank, J.; and Hoffmann, J. 2022. Evaluating Plan-Property Dependencies: A Web-Based Platform and User Study. In Kumar, A.; Thiébaux, S.; Varakantham, P.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, 687–691. AAAI Press.
- Eifler, R.; Fiser, D.; Siji, A.; and Hoffmann, J. 2024. Iterative Oversubscription Planning with Goal-Conflict Explanations: Scaling Up Through Policy-Guidance Approximation. In *ECAI*, 4092–4099.
- Eisenhut, J.; Schuler, X.; Fiser, D.; Höller, D.; Christakis, M.; and Hoffmann, J. 2024. New Fuzzing Biases for Action Policy Testing. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, 162–167. AAAI Press.
- Fichte, J. K.; Gaggli, S. A.; and Rusovac, D. 2022. Rushing and Strolling among Answer Sets - Navigation Made Easy. In Honavar, V.; and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 5651–5659. AAAI Press.
- Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11801–11808. AAAI Press.
- Gelfond, M.; and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In Kowalski, R. A.; and Bowen, K. A., eds., *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, 1070–1080. MIT Press.
- Gnad, D.; Hecher, M.; Gaggli, S.; Rusovac, D.; Speck, D.; and Fichte, J. K. 2025. Interactive Exploration of Plan Spaces. In Ortiz, M.; Wassermann, R.; and Schaub, T., eds., *Proceedings of the Twenty-Second International Conference on Principles of Knowledge Representation and Reasoning (KR 2025)*. IJCAI Organization.
- Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.
- Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.
- Krärup, B.; Krivic, S.; Magazzeni, D.; Long, D.; Cashmore, M.; and Smith, D. E. 2021. Contrastive Explanations of Plans through Model Restrictions. *J. Artif. Intell. Res.*, 72: 533–612.
- Langer, M.; Oster, D.; Speith, T.; Hermanns, H.; Kästner, L.; Schmidt, E.; Sesing, A.; and Baum, K. 2021. What do we want from Explainable Artificial Intelligence (XAI)? - A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research. *Artif. Intell.*, 296: 103473.
- McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.
- Musayev, F.; Drexler, D.; Gnad, D.; and Seipp, J. 2025. Combining Heuristics and Transition Classifiers in Classical

Planning. In Lynce, I.; and Murano, A., eds., *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, 4694–4701. IOS Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. <https://doi.org/10.5281/zenodo.790461>.

Speck, D.; Hecher, M.; Gnad, D.; Fichte, J. K.; and Corrêa, A. B. 2025. Counting and Reasoning with Plans. In Shah, J.; and Kolter, Z., eds., *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*, 26688–26696. AAAI Press.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of explanations as model reconciliation. *Artif. Intell.*, 301: 103558.

Sreedharan, S.; Muise, C.; and Kambhampati, S. 2023. Generalizing Action Justification and Causal Links to Policies. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 417–426. AAAI Press.

Sreedharan, S.; Srivastava, S.; Smith, D. E.; and Kambhampati, S. 2019. Why Can’t You Do That HAL? Explaining Unsolvability of Planning Tasks. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 1422–1430. IJCAI.

Steinmetz, M.; Fišer, D.; Eniser, H. F.; Ferber, P.; Gros, T.; Heim, P.; Höller, D.; Schuler, X.; Wüstholtz, V.; Christakis, M.; and Hoffmann, J. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 353–361. AAAI Press.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2): 280–296.

Vinzent, M.; Hermanns, H.; and Hoffmann, J. 2026. Probabilistic Safety Verification of Neural Policies via Predicate Abstraction. In Koenig, S.; Jenkins, C.; and Taylor, M. E., eds., *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, 36438–36447. AAAI Press.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In Kumar, A.; Thiébaux, S.; Varakantham, P.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, 371–379. AAAI Press.