Verifinsta: Verifying If an Instance Belongs to a Domain

Claudia Grundke, Gabriele Röger, Malte Helmert

University of Basel
Basel, Switzerland
{claudia.grundke, gabriele.roeger, malte.helmert}@unibas.ch

Abstract

A recently proposed extension of PDDL domains makes it possible to restrict the legal initial states and goals of a domain for classical planning. We show how a reduction to the plan-existence problem can be used to verify whether an instance belongs to such an augmented domain. An empirical evaluation demonstrates the feasibility of the approach and reveals inaccuracies in existing domain characterizations as well as inconsistencies in the Learning IPC benchmarks.

1 Introduction

A planning domain specified in the planning domain definition language PDDL (McDermott et al. 1998; Fox and Long 2003; Gerevini et al. 2009) captures important aspects such as the available actions and how their application affects the state of the world. However, it is not sufficient to characterize an application domain in a way that supports generalized planning, where a single (generalized) plan is expected to solve all tasks of the domain (e.g. Srivastava, Immerman, and Zilberstein 2011; Bonet and Geffner 2018; Francès et al. 2019; Illanes and McIlraith 2019; Drexler, Seipp, and Geffner 2021). Such generalized plans typically exploit structurally similar goals (e.g. that all packages should be delivered) and shared properties of the initial states (e.g. that initially every package is at exactly one location).

For this reason, Grundke, Röger, and Helmert (2024) proposed an extension that allows the domain modeler to restrict the initial states and to define a common high-level goal for all instances of the domain. Their formalization is based on a *legality predicate* whose truth value is determined by a set of axioms, called *legality axioms*. An instance, specifying a set of constants, an initial state and a goal, belongs to the domain if the legality predicate is true for the initial state and if the goal is identical to the domain goal.

We implemented an instance verifier, called verifinsta, that decides for a given such formalized domain and a given instance whether the instance belongs to the domain (Figure 1). It first checks whether the instance goal and domain goal match. For the verification of the initial state, we exploit that the legality is specified in terms of standard PDDL axioms. This allows us to easily create a *verification* task that is solvable (by the empty plan) iff the initial state is legal for the domain. By reducing initial state verification to planning task solvability, essentially any planner that supports axioms can be used for this step of the verification.

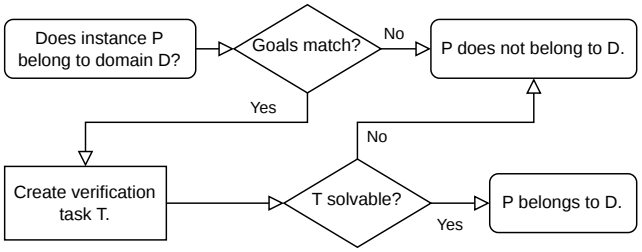

Figure 1: Flow diagram for verifinsta

After introducing the necessary background, we will present the details of this verification, in particular the construction of the verification task. Afterwards we describe how we support a variation proposed by Grundke, Röger, and Helmert (2024) that shifts some goal information into the initial state to be under the scope of the legality axioms. This is necessary to support existing STRIPS benchmarks, where instance goals are defined as conjunctions of ground atoms that cannot directly be shared between instances. Finally, we verify the feasibility of our overall approach by means of an experimental evaluation.

2 Background

This section introduces the formalism by Grundke, Röger, and Helmert (2024) and its relation to PDDL. The verifinsta tool also supports its straight-forward extension to PDDL types (Grundke, Helmert, and Röger 2025) but we omit them here for ease of presentation.

A *planning task* consists of a *(PDDL) domain* and an *instance*. The domain primarily specifies the dynamics of the world, i.e., how world states can be altered through action applications. To this end, it defines the predicates used to describe world states, as well as possibly some domain-wide constants. A domain may also introduce axioms, which define the truth of certain atoms in terms of other atoms. The same PDDL domain is typically shared across multiple planning tasks, while the instance specifies the constants, the initial world state, and the goal of a particular task.

A *state* of a task is a truth assignment for all ground atoms over the predicates and constants (including the domain-wide constants). The predicates of a domain are partitioned into the *basic* and the *derived predicates*. The actions may only directly affect basic predicates, and a *basic state* only defines the truth for the ground atoms of these basic predicates. For the derived predicates, the truth is derived from the basic state based on the *axioms*, which must be stratifiable:

Definition 1 (axioms; adapted from Thiébaux, Hoffmann, and Nebel (2005))**.** *An* axiom *is a rule of the form $H(\bar{x}) \leftarrow B(\bar{x})$, where the* head *$H(\bar{x})$ is a single first-order atom with free variables $\bar{x}$ and the* body *$B(\bar{x})$ is a first-order formula with the same free variables $\bar{x}$.*

A set $\mathcal{A}$ of axioms is stratifiable *if there exists a partition (stratification) $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of predicates such that for each predicate $P_i \in \mathcal{P}_i$ and axiom $P_i(\bar{x}) \leftarrow B(\bar{x}) \in \mathcal{A}$:*

- *if $P_j \in \mathcal{P}_j$ appears in $B(\bar{x})$, then $j \leq i$, and*
- *if $P_j \in \mathcal{P}_j$ appears negated in the translation of $B(\bar{x})$ to negation normal form, then $j < i$.*

A basic state is *extended* to a (full) state through a sequence of fixed-point computations following the stratification order. Initially, only the atoms in the basic state are true. For each stratum in turn, ground atoms of predicates in that stratum are added whenever they are the head of an instantiated axiom whose body is satisfied under the current truth assignment.

Formally, we define a PDDL domain as follows:

Definition 2 (PDDL domain)**.** *A* PDDL domain *is a tuple $\langle \mathcal{P}, \mathcal{C}, \mathcal{O}, \mathcal{A} \rangle$, where*

- *$\mathcal{P}$ is a finite set of predicate symbols that can be partitioned into a set of basic predicates $\mathcal{B} \supseteq \{=\}$ and a set of derived predicates $\mathcal{D}$,*
- *$\mathcal{C}$ is a finite set of constants (or objects),*
- *$\mathcal{O}$ is a finite set of operators (or actions)*[1]*, and*
- *$\mathcal{A}$ is a stratifiable finite set of axioms.*

In contrast to the other basic predicates, the equality predicate $=$ is interpreted as built-in constant equality and its truth values cannot be changed by action applications.

An *instance* for a PDDL domain is a tuple $\langle \mathcal{C}', \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{C}'$ is a finite set of additional instance-specific constants, the *initial state* $\mathcal{I}$ is a basic state (given as the set of true ground atoms), and the *goal* $\mathcal{G}$ is a first-order sentence. Under the PDDL standard, an instance *fits* the domain whenever its initial state and goal are defined over the domain predicates and the domain-wide and instance-specific constants. This typically over-approximates the set of possible instances of an actual application domain. For example, in the Blocksworld domain, it permits a block b to be initially placed on itself, since nothing prevents the atom $on(b, b)$ from being true in the initial state.

Grundke, Röger, and Helmert (2024) extend PDDL domains to restrict the permitted initial states and goals of domain instances. In particular, they introduce a single domain-level goal that all instances are required to use. To describe the *legal initial states* they add further axioms with corresponding derived predicates to the domain and declare one of the new predicates as the *legality predicate*. The truth of this nullary predicate in the extension of the initial state decides the legality. For expressiveness reasons they also introduce a new basic predicate $<$, called the *ordering predicate*, with special semantics. Similarly to the equality predicate $=$, the interpretation of the ordering predicate $<$ may not be altered by the operators. It is instead assumed to correspond to an arbitrary linear order over all constants (domain-wide and instance-specific). The added axioms may use this ordering predicate in the body but must be *order-invariant*, which means that for a fixed basic state, all linear orders must induce the same truth value for the legality predicate in the extended state.

We refer to such augmented domains as *formalized*:

Definition 3 (Formalized domain; adapted from Grundke, Röger, and Helmert (2024))**.** *Let $\mathbf{D}_{\text{PDDL}} = \langle \mathcal{P}, \mathcal{C}, \mathcal{O}, \mathcal{A} \rangle$ be a PDDL domain. A* formalized domain *is a tuple $\langle \mathbf{D}_{\text{PDDL}}, \mathcal{P}_{\text{legal}}, \mathcal{A}_{\text{legal}}, \mathit{legal}, \mathcal{G} \rangle$, where*

- *$\mathcal{P}_{\text{legal}}$ with $\mathcal{P}_{\text{legal}} \cap \mathcal{P} = \emptyset$ is a finite set of predicate symbols that can be partitioned into a set of derived predicates $\mathcal{D}_{\text{legal}}$ and $\{<\}$,*
- *$\mathcal{A}_{\text{legal}}$ is a stratifiable finite set of axioms, the* legality axioms*, using predicates from both $\mathcal{P}$ and $\mathcal{P}_{\text{legal}}$ in the bodies but only from $\mathcal{D}_{\text{legal}}$ in the heads,*
- *legal is the nullary* legality predicate *with $\mathit{legal} \in \mathcal{D}_{\text{legal}}$ whose interpretation is* order-invariant *wrt. $<$,*
- *$\mathcal{G}$ is a first-order sentence using predicates only from $\mathcal{P}$.*

An initial state is legal for the domain if its extension satisfies the legality predicate:

Definition 4 (Legality; adapted from Grundke, Röger, and Helmert (2024))**.** *Let $\mathbf{D}$ be a formalized domain with legality predicate legal. Let $\mathcal{A}$ be the axioms from its PDDL domain and $\mathcal{A}_{\text{legal}}$ be its legality axioms.*

An initial state $\mathcal{I}$ is legal *for $\mathbf{D}$ if $\mathit{legal}()$ is true in the extension of $\mathcal{I}$ with the axioms from $\mathcal{A} \cup \mathcal{A}_{\text{legal}}$ for any interpretation of $<$ as a linear order on the universe (consisting of the domain-wide and instance-specific constants).*

Formalized domains can express exactly the properties of initial states that can be decided in polynomial time, and for a fixed domain it can be tested in polynomial time whether $\mathit{legal}()$ is true in the extension of a state (Grundke, Röger, and Helmert 2024).

3 Verifying Instances

We implemented a tool in Python to verify instances against formalized domains. We call this tool verifinsta as it allows **ver**ifying **if** an **insta**nce is legal for a formalized domain.

A given instance $\langle \mathcal{C}', \mathcal{I}, \mathcal{G} \rangle$ *belongs* to a formalized domain $\mathbf{D}$ for PDDL domain $\mathbf{D}_{\text{PDDL}}$, if it has the following three properties:

Property 1 The instance fits the PDDL domain $\mathbf{D}_{\text{PDDL}}$,

Property 2 $\mathcal{G}$ matches the domain-wide goal, and

Property 3 $\mathcal{I}$ is legal for $\mathbf{D}$.

[1] The full definition of operators is not relevant for this work.

These properties can be verified independently and for properties 1 and 3 we can build on existing tools.

For the first property, there are established tools for exactly this task, so we do not discuss it more deeply. For example, the plan validator VAL (Howey, Long, and Fox 2004) can not only validate plans but also check whether a given (PDDL) domain and instance encode a valid PDDL task. The second property is the only aspect verified fully by verifinsta. For the third property, we construct a new verification task that is solvable iff the initial state is legal for the formalized domain. If the verification task is solvable then it is already solved by the empty plan. We can thus rely on any (complete) planner or plan validator that supports axioms to verify that the initial state is legal.

Legality of the initial state

Our tool verifinsta compiles the given instance and formalized domain into a *verification task* with a *verification instance* $\mathbf{P}_{\text{ver}}$ and *verification domain* $\mathbf{D}_{\text{ver}}$.

The main idea is to replace the goal with *legal*() and to remove all actions, so that the legality atom must already be derivable from the initial state. This compilation is mostly straight-forward, with the ordering predicate $<$ being the only aspect that is not directly covered by the standard PDDL semantics. While the formalized domains treat it as a built-in predicate, we need to make its interpretation explicit in the verification task (which is a standard PDDL task).

Since the legality axioms of a formalized domain are order-invariant, we can fix an arbitrary linear order over the constants for the interpretation of $<$ and add it to the initial state. By the definition of order invariance, the choice of the order will not affect the solvability of the task. In the implementation, verifinsta simply appends the instance-specific constants to the domain-wide constants, preserving the order given in the respective files.

Definition 5 (Verification task)**.** *For a given formalized domain* $\mathbf{D} = \langle \mathbf{D}_{\text{PDDL}}, \mathcal{P}_{\text{legal}}, \mathcal{A}_{\text{legal}}, \mathit{legal}, \mathcal{G} \rangle$ *with* $\mathbf{D}_{\text{PDDL}} = \langle \mathcal{P}, \mathcal{C}, \mathcal{O}, \mathcal{A} \rangle$ *and instance* $\mathbf{P} = \langle \mathcal{C}', \mathcal{I}, \mathcal{G}' \rangle$ *that fits* $\mathbf{D}_{\text{PDDL}}$*, a* verification task *consists of the PDDL domain* $\mathbf{D}_{\text{ver}}$ *and a verification instance* $\mathbf{P}_{\text{ver}}$ *for a linear order* $\prec$ *as follows:*

- $\mathbf{D}_{\text{ver}} = \langle \mathcal{P} \cup \mathcal{P}_{\text{legal}}, \mathcal{C}, \{\}, \mathcal{A} \cup \mathcal{A}_{\text{legal}} \rangle$*,*
- $\prec$ *is a linear order on* $\mathcal{C} \cup \mathcal{C}'$*, and*
- $\mathbf{P}_{\text{ver}} = \langle \mathcal{C}', \mathcal{I}', \mathit{legal}() \rangle$ *with* $\mathcal{I}' = \mathcal{I} \cup \{ \texttt{(<\ a\ b)} \mid a, b \in \mathcal{C} \cup \mathcal{C}' \text{ with } a \prec b \}$.

Note that the union $\mathcal{A} \cup \mathcal{A}_{\text{legal}}$ of the axioms is stratifiable, e.g., by the stratification that appends a stratification of $\mathbf{D}$'s axioms to a stratification of $\mathbf{D}_{\text{PDDL}}$'s axioms, because no axiom of $\mathbf{D}$ mentions a predicate from $\mathbf{D}_{\text{PDDL}}$ in the head.

Proposition 1. *A verification task* $\langle \mathbf{D}_{\text{ver}}, \mathbf{P}_{\text{ver}} \rangle$ *for a formalized domain* $\mathbf{D}$ *and an instance* $\mathbf{P}$ *that fits its PDDL domain is solvable iff the initial state of* $\mathcal{I}$ *is legal for* $\mathbf{D}$*.*

Based on this proposition, any complete planner that supports axioms can verify whether the initial state is legal for the formalized domain. Moreover, since the verification task contains no actions, it is either already solved by the empty plan or not solvable at all, so we can alternatively also use a plan validation tool such as VAL on the verification task and the empty plan to decide whether the initial state is legal.

Matching Goals

The original work by Grundke, Röger, and Helmert (2024) does not consider the goal a part of the instance. Instead they simply enforce the domain-wide goal for all of its tasks.

They argue that a shared first-order goal is natural for many application domains. For instance, in a logistics scenario where all packages need to be delivered one can use the same goal $\forall p\,(\mathit{package}(p) \rightarrow \mathit{delivered}(p))$ for all tasks.

For other cases, they suggest a transformation that moves some aspects of the goal specification into the initial state, to make a uniform goal usable across instances. They used this to cover existing benchmark domains in the more restricted STRIPS formalism. Without a transformation, STRIPS instances can in general not be used for formalized domains. Their goals are conjunctions of ground atoms using instance-specific constants and can thus not be shared across instances.

The suggested transformation works as follows. Let $\mathcal{P}_{\text{goal}}$ be the set of predicates that occur in a goal of some instance of the (existing benchmark) domain. For each $P \in \mathcal{P}_{\text{goal}}$, add a new domain-wide (basic) predicate P^g of the same arity, and use the formula

$$\bigwedge_{P \in \mathcal{P}_{\text{goal}}} \forall \bar{x}(P^g(\bar{x}) \rightarrow P(\bar{x})). \tag{1}$$

for the domain-wide goal. Transform every instance by adding for each atom $P(\bar{c})$ of its goal conjunction an atom $P^g(\bar{c})$ to the initial state. Then the domain-wide goal characterizes the same goal states as the original STRIPS goal.

This idea can be extended to conjunctions of *literals* by introducing further predicates for negative literals, but we only support conjunction of atoms.

In addition to enabling a single domain-wide goal, the transformation allows conceptual goals to be further refined through legality axioms. Consider, for example, a Blocksworld domain with the (basic) predicate *on*, in which each instance goal is a conjunction of atoms representing a single tower of blocks. After introducing predicate on^g, one can use legality axioms to enforce goals of this form:

$$\begin{aligned} \mathit{topBlock}(x) &\leftarrow \forall y\, \neg \mathit{on}^g(x, y) \\ \mathit{legal}() &\leftarrow \neg \mathit{illegal}() \\ \mathit{illegal}() &\leftarrow \exists x, y\, (\mathit{topBlock}(x) \wedge \mathit{topBlock}(y) \wedge x \neq y) \end{aligned}$$

A top block is a block for which the goal does not specify a block that is on top of it. A legal instance may not have multiple top blocks. (For enforcing a single tower, we would need further axioms to prevent physically impossible configurations such as cyclic "towers" with a on b and b on a).

Since Grundke, Helmert, and Röger (2025) built on this transformation in their formalized domains and we want to be compatible with the existing STRIPS benchmarks, we added a STRIPS goal option to verifinsta that switches between two behaviors:

- In the default setting, verifinsta verifies that the given instance goal is literally the domain-wide goal.

- With the option enabled, it checks that the goal is indeed a STRIPS goal and that the domain-wide goal corresponds to the one from Eq. (1). It then compiles the goal information into the initial state as described above and uses the augmented state in the subsequent construction of the verification task.

To summarize, verifying instances for formalized domains proceeds in three steps:

0. (Optionally) an external tool verifies that the instance and underlying PDDL domain are syntactically valid.
1. Verifinsta checks that the instance goal matches the domain goal and produces the verification task, optionally handling STRIPS goals as described above.
2. A complete planner or plan validation tool supporting axioms verifies that the verification task is solvable (by the empty plan).

The instance belongs to the formalized domain iff all three steps succeed (assuming correctness of the external tools).

In the command line interface of verifinsta, we left step 0 optional because in many applications the compliance with the PDDL standard is taken as granted. For step 2, the user can choose between using Fast Downward (Helmert 2006) or VAL (Howey, Long, and Fox 2004) as the external tool. Alternatively, one can only call step 1 to obtain the verification task and check its solvability externally.

4 Experimental Evaluation

To evaluate verifinsta, we verified instances against the domains formalized by Grundke, Helmert, and Röger (2025). These formalized domains are based on nine out of ten PDDL domains from the IPC 2023 learning track (Taitler et al. 2024) and capture instance properties implied by the domain-specific (procedural) instance generators used in the competition. In our evaluation, we use the instances from the corresponding competition benchmark set.

Since all these instances have conjunctions of ground atoms as goals (satisfying the STRIPS requirement of PDDL) we enabled verifinsta's STRIPS goal option.

We also had to adjust some formalized domains because the domain-wide goal did not have the required form. For example, in the childsnack domain we changed the domain goal from `(forall (?c - child) (served ?c))` to `(forall (?c - child) (imply (served_g ?c) (served ?c)))` and added a legality axiom ensuring that `(served_g ?c)` holds for all children (in the initial state).

Preliminary experiments revealed small mistakes in the legality axioms of some domains, e.g., using non-existing variables. We fixed those for our later experiments.

With the fixes all instances where verifinsta did not run out of time or memory were found to be legal for all domains except for floortile, rovers, and childsnack. The single floortile instance that was found to be illegal seems to mistakenly mark a tile as *clear* in the initial state although it is blocked by a robot. The five rovers instances that were found to be illegal are not consistent with the rovers instance generator of the IPC. For example, the generator produces only instances where at least one rover has the equipment for each required image and rock or soil analysis. Four of the five illegal rovers instances do not satisfy this. Thus, verifinsta correctly detected these instances as illegal, as the legality axioms capture the structural constraints enforced by the instance generator.

	Result	Plan Time	Total Time	Mem
Blocksworld (189)	189/0/0	1.28	1.32	42.23
Childsnack (188)	149/7/32	26.12	26.19	276.61
Ferry (189)	189/0/0	2.19	2.38	82.67
Floortile (187)	169/1/17	29.09	29.24	316.08
Miconic (189)	189/0/0	2.36	2.42	75.12
Rovers (189)	155/5/29	16.44	16.54	162.96
Satellite (189)	189/0/0	1.19	1.22	39.06
Spanner (179)	142/0/37	25.57	25.61	249.24
Transport (189)	189/0/0	2.99	3.02	72.19
Total (1688)	1560/13/115	11.91	11.99	146.24

Table 1: Statistics on running verifinsta on the IPC 2023 learning track instances. Numbers in parentheses denote the total number of instances of the domain. Result counts the legal/illegal/unsolved instances, where unsolved means that verifinsta did not terminate with the given time and memory bound. Plan Time and Total Time show, respectively, the average CPU time in seconds for solving the verification tasks and for building and solving them. Mem shows the average maximal memory in MB.

For childsnack, 127 / 188 instances were found to be illegal initially. Most of these instances are illegal because the formalized domain requires the number of bread and content portions, respectively, to equal the number of *sandwiches* while the childsnack instance generator of the IPC sets the portion numbers equal to the number of *children*. We changed the formalized childsnack domain to be consistent with the IPC instance generator and used this modified domain in all following experiments.

With the modified childsnack domain seven instances were still found to be illegal. Four of those are illegal because they have more bread or content portions than children. Their portion numbers equal the number of sandwiches making them inconsistent with the childsnack instance generator but legal for the original formalized childsnack domain. The remaining three instances are illegal because they are also not consistent with the instance generator, however for a different reason. The generator produces only instances where the numbers of gluten-free bread and content portions, respectively, equal the number of allergic children. The three illegal instances do not satisfy this because they contain more gluten-free portions than allergic children.

All instances that verifinsta detected as illegal (after modifying the childsnack domain) are so-called base cases of the IPC benchmarks. These instances are very small and appear to be (at least partially) hand-crafted. This demonstrates that verifinsta can uncover mistakes and inconsistencies in benchmarks given suitable domain formalizations.

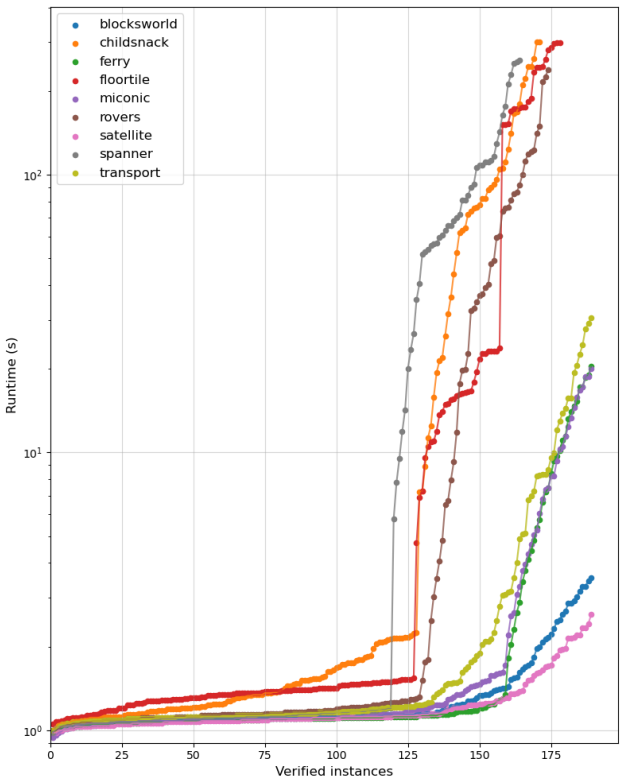

Figure 2: Cactus plot showing the number of verified instances against runtime by domain.

We ran verifinsta on each instance of the IPC domains on a cluster with Intel Xeon Silver 4114 CPUs, using a time limit of 5 minutes (both CPU and wall-clock time) and a memory limit of 4 GiB. The code of verifinsta can be found online[2] and the raw data of the evaluation will be published with the camera-ready version of this paper.

Table 1 shows time and memory measurements accumulated by domain, using the Apptainer version of Fast Downward[3] 24.06 to solve the verification tasks. For solving, verifinsta runs a blind search with Fast Downward where the invariant synthesis of Fast Downward's translator component is deactivated.

For five of the nine IPC domains, all instances could be verified within the given resource limits. For the remaining domains, however, verifinsta exceeded the resource limits on 115 instances. In all cases, this occurred while solving the verification task. The average runtimes for successful runs confirm that verifinsta spends nearly all of its runtime solving the verification tasks, while the time required to construct them is negligible. Fast Downward timed out on 56 instances and exhausted the memory limit on 59 instances. All of these instances are among the largest in their respective domains, some of which scale to very large sizes; for example, the largest instance of the Spanner domain has more than 480 spanners. The smallest unsolved instance in this domain contained 76 spanners, 42 nuts, and 38 locations.

The cactus plot in Figure 2 shows that running times remain within a few seconds for many instances before increasing rapidly. We observed that this behavior correlates with the different test sets (simple/medium/hard) of the learning competition, which increasingly stress Fast Downward, in particular during its grounding step.

To assess whether the 5-minute time limit is too restrictive when using Fast Downward, we conducted an additional experiment with a 30 minutes time limit, maintaining the memory limit. In this setting, only 8 additional instances could be verified. Compared to the experiment with a 5-minute time limit, verifinsta timed out on 12 fewer instances but ran out of memory on 4 additional instances. This observation suggests that, in some domains, verification does not scale beyond a certain instance size when using a non-lifted planner.

We also ran an experiment (with a 5 minutes time limit) where verifinsta used the plan validator VAL[4] for checking the solvability of the verification tasks. With VAL, verifinsta verified 249 fewer instances than with Fast Downward. It ran out of time on 72 instances (compared to 56 with Fast Downward) and out of memory on 292 (compared to 59) instances. This shows that memory is much more important when running verifinsta with VAL instead of Fast Downward. As expected, both variants yield the same legality results for all instances on which both terminated.

5 Conclusion

We presented an approach for verifying whether a given instance belongs to a domain, building on a recently proposed extension of PDDL domains with additional information about legal initial states and a domain-wide goal. In particular, we showed how the core verification problem, namely the legality of the initial state, can be reduced to determining whether a corresponding verification task is solvable by the empty plan, and described the construction of this task.

In addition, our tool incorporates a known transformation of STRIPS instances that broadens the applicability of formalized domains to additional benchmark domains.

Our experiments show that verification with verifinsta is feasible, while also highlighting the limitations of non-lifted approaches under the extreme scaling of the learning competition. Moreover, verifinsta proved useful for debugging purposes. During our experiments, we discovered multiple subtle mistakes and inconsistencies in both the domain formalizations and the benchmark instances.

References

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In Lang, J., ed., *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 4667–4673. IJCAI.

Drexler, D.; Seipp, J.; and Geffner, H. 2021. Expressing and Exploiting the Common Subgoal Structure of Classical

[2]https://github.com/grucla/verifinsta

[3]https://www.fast-downward.org

[4]https://github.com/KCL-Planning/VAL

Planning Domains Using Sketches. In Erdem, E.; Bienvenu, M.; and Lakemeyer, G., eds., *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 258–268. IJCAI Organization.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5554–5561. IJCAI.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *Artificial Intelligence*, 173(5–6): 619–668.

Grundke, C.; Helmert, M.; and Röger, G. 2025. Domain-Independent Instance Generation for Classical Planning. 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)*, 805–809. IJCAI Organization.

Grundke, C.; Röger, G.; and Helmert, M. 2024. Formal Representations of Classical Planning Domains. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 239–248. AAAI Press.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *Proceedings of the 16th International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 294–301. IEEE.

Illanes, L.; and McIlraith, S. A. 2019. Generalized Planning via Abstraction: Arbitrary Numbers of Objects. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7610–7618. AAAI Press.

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.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2): 393–401.

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.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In Defense of PDDL Axioms. *Artificial Intelligence*, 168(1–2): 38–69.