Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)

Towards Certified Unsolvability in Classical Planning

Gabriele Röger
University of Basel, Switzerland
gabriele.roeger@unibas.ch

Abstract

While it is easy to verify that an action sequence is a solution for a classical planning task, there is no such verification capability if a task is reported unsolvable. We are therefore interested in certificates that allow an independent verification of the absence of solutions. We identify promising concepts for certificates that can be generated by a wide range of planning approaches. We present a first proposal of unsolvability certificates and sketch ideas how the underlying concepts can be used as part of a more flexible unsolvability proof system.

1 Introduction

The aim of classical planning is to find a sequence of actions (a *plan*) that transforms the current world situation into a desirable state or to prove that no such plan exists.

As with all software systems, the output of a planning system can be wrong, due to software bugs, hardware faults or even malicious reasons. This is no problem in academia but as intelligent problem-solving techniques are more and more commonly applied in the real world, it gets increasingly important that we are able to verify the correctness of the output.

Traditionally, the planning community placed most emphasis on actually finding plans and there are several tools available that can efficiently check whether an action sequence is a valid plan (e. g. VAL [Howey and Long, 2003], INVAL[1]). Recently, the problem of identifying *unsolvable* planning tasks has attracted increasing interest, not least due to the first International Planning Competition on proving unsolvability (UIPC). As there have been no analogous validation capabilities for unsolvability, participating planning systems were discouraged from "guessing" by disqualifying them from a domain when they falsely reported a task unsolvable. While this might be acceptable in a competition, applying such systems reliably in real-world scenarios requires the capability to verify the output also for unsolvable tasks.

We are therefore interested in the development of *certifying* planning systems. Besides its usual output, a *certifying algorithm* [McConnell *et al.*, 2011] generates a *certificate* that justifies the result and is sufficient for verifying its correctness independently of the algorithm. To be practically useful, certificates should satisfy four criteria [Eriksson *et al.*, 2017]:

- *completeness:* There should be a certificate for every unsolvable planning task.
- *efficient generation:* Transforming a non-certifying algorithm into a certifying one should be possible with reasonable (polynomial, ideally linear) overhead.
- *efficient verification:* Verifying the correctness of a certificate should be possible with reasonable effort (e.g., polynomial in the size of the certificate).
- *generality:* A single class of certificates (and hence verifier) should be useful for a wide range of algorithms.

In this paper we describe first steps towards such certified unsolvability. After a high-level introduction of the planning problem, we give an overview of recent approaches for proving unsolvability and identify some key concepts that might be suitable to cover a broad spectrum of algorithms. We briefly describe a recent proposal for general unsolvability certificates and sketch ideas how the underlying concepts can be used as part of a more flexible unsolvability proof system.

2 Background

A *planning task* is typically defined over a set of state variables that can take on values from a finite domain. A *state* is given by a total assignment to the state variables, and we denote the set of all states by $\mathcal{S}$. An *action* is specified by a *precondition*, which defines in which states it is applicable, and an *effect* that specifies how the action application changes the world. In classical planning actions are deterministic, i. e. there is always a single, fully determined successor state. The successor relation described by the actions induces a *state space*, which is a directed graph where the vertices correspond to all possible states and the edges to the action applications. The size of the state space is generally exponential in the size of the task specification. In addition to the action specification, the definition of a planning task identifies an *initial state* and specifies a *goal* condition. We say that a state s is *solvable* if there is a path in the state space from s to a state that satisfies the goal condition. An s-plan is a sequence of actions that induces such a path, and it is a *plan* for the task if s is the initial state. A task is *unsolvable* if there exists no plan.

[1] `http://users.cecs.anu.edu.au/~patrik/`

3 Related Work in Planning

The predominant approach to classical planning is informed state-space search, where the search is guided by a heuristic function that estimates the cost to reach the goal from a given state. An estimate ∞ for a state s from a *safe* heuristic indicates that s is a *dead end*, i. e. it is impossible to reach the goal from s. During the search, infinite heuristic estimates are used to prune the search space. If a complete search algorithm such as A* or greedy best-first search exhausts the search space with a safe heuristic, then the task is unsolvable. This is also true for the extreme case, where the heuristic identifies the *initial* state of the task as a dead end.

Heuristic estimates are typically computed from a simplification of the planning task and most of the heuristics that are relevant for this work fall into one of the following classes:

Abstraction heuristics define an equivalence relation over the states and do not distinguish equivalent states. The induced *abstract state space* contains a state for each equivalence class and has the property that every solution in the original state space is also a solution in the abstract space, so that abstract goal distances are admissible heuristic estimates. In particular, if an abstract state is unsolvable then the corresponding original states are unsolvable. Typical representatives of abstraction heuristics are *PDB heuristics* [Edelkamp, 2001] or *merge-and-shrink heuristics* [Helmert *et al.*, 2014].

Delete-relaxation heuristics [Bonet and Geffner, 2001; Hoffmann and Nebel, 2001] ignore “negative” effects of actions. For example, moving a truck from A to B has the disadvantage that it is no longer at A. Delete relaxation ignores this, intuitively accumulating the values of state variables instead of switching between them. As a result, the truck in the example would not only be at B but also still at A, so we could transport a package from B to A by loading it at B and unloading it at A without another movement of the truck. *Partial delete-relaxation* approaches [Katz *et al.*, 2013; Domshlak *et al.*, 2015; Keyder *et al.*, 2014] allow a more fine-grained selection of what deletes may be ignored.

Critical-path heuristics [Haslum and Geffner, 2000; Hoffmann and Fickert, 2015] are defined in terms of a set C of partial variable assignments. The heuristic computation approximates the cost of achieving action preconditions and the goal condition by the cost of achieving the most expensive element of C that is implied by the condition.

Potential heuristics [Pommerening *et al.*, 2015] assign a potential to each value of a state variable and simply sum up these potentials for the values in a state. The potentials are typically determined with linear programming.

These heuristics focus on computing good heuristic estimates, and the detection of dead ends is rather a side product of the computation. However, for proving unsolvability, the goal distances of solvable states are irrelevant. Hoffmann *et al.* [2014] thus introduced the concept of *unsolvability heuristics* as functions $u : \mathcal{S} \to \{0, \infty\}$ where $u(s) = \infty$ implies that s is a dead end and $u(s) = 0$ means that we do not know.

Most approaches for detecting unsolvable planning tasks use an unsolvability heuristic that is based on an existing heuristic but optimized towards a specifically good dead-end detection or faster computation. There are unsolvability heuristics based on PDBs [Bäckström *et al.*, 2013; Pommerening and Seipp, 2016; Seipp *et al.*, 2016; Torralba, 2016], merge-and-shrink abstractions [Hoffmann *et al.*, 2014; Torralba *et al.*, 2016], partial delete-relaxation [Gnad *et al.*, 2016a], critical paths [Steinmetz and Hoffmann, 2017b; Haslum, 2016] and potential heuristics [Seipp *et al.*, 2016]. While most of them are used for pruning within a search, some of the unsolvability approaches try to strengthen the heuristic computation until the initial state is detected as a dead end [Bäckström *et al.*, 2013; Hoffmann *et al.*, 2014; Torralba *et al.*, 2016; Haslum, 2016].

To avoid expensive heuristic computations, some approaches derive logical formulas with the property that if the formula is true under the variable assignment of a state then the state is a dead end. As this can be evaluated very quickly, such dead-end detectors can always be consulted before the more expensive heuristic evaluation. Such formulas can be computed in a preprocessing step [Lipovetzky *et al.*, 2016; Steinmetz and Hoffmann, 2017a] or be derived online during the search [Steinmetz and Hoffmann, 2017b; 2017a].

Many systems use irrelevance pruning, most notably the preprocessing technique by Alcázar and Torralba [2015] that alternates between a forward and backward analysis to derive formulas that recognize states that can never be part of a plan.

Besides heuristic-based methods, there are also search-based approaches that explore a very different search space [Gnad *et al.*, 2016b] and satisfiability-based approaches using model checking [Korovin and Suda, 2016] or property directed reachability [Balyo and Suda, 2016].

Despite the large number of systems for detecting unsolvable planning tasks, there is only one that can certify its output [Steinmetz and Hoffmann, 2017b], and its certificate generation is limited to this specific approach.

4 Key Concepts

One challenge in defining general unsolvability certificates is to identify concepts that cover a wide range of the previously mentioned techniques. As the question whether a planning task is solvable is inherently a reachability question in a directed graph, *reachability*-based concepts look most promising for this endeavor. In the following we introduce two such concepts and briefly discuss representational issues.

Inductive Sets and Dead States

An *inductive set* S is a set of states that is closed under action application, i. e. all action applications in a state in S lead to a state in S. Once entered, such a set cannot be left again, and if an inductive set does not contain a goal state of the task then all states in the inductive set are unsolvable.

Definition 1 (Eriksson *et al.*, 2017)**.** *An* inductive certificate *for state s consists of a set S of states such that (1) S contains s (2) S contains no goal state and (3) S is inductive.*

Clearly, if there is an inductive certificate for a state then the state is unsolvable. Moreover, there is an inductive certificate for every unsolvable state (e. g. the set consisting of all states reachable from this state).

The definition of inductive sets stems from a forward-search perspective and there is a natural translation to regression search (backwards from the goal): A *backwards-inductive set* is a set of states S that cannot be entered from the outside, i. e. *no* action application in a state that is not in S leads to a state in S. If a backwards-inductive set contains *all* goal states but not state s then state s is unsolvable. This concept would also be sufficient to define a certificate for every unsolvable task (e. g. using the set of all solvable states).

While inductive sets consider reachability *from* a set, backwards-inductive sets capture reachability *of* a set. The notion of dead states allows combining both concepts relative to the initial state and the goal: a state is *dead* if it is not reachable from the initial state or if it is unsolvable. This simple generalizing property can be very helpful if we need to integrate information from different sources of information.

Sets of States as Logical Formulas

If we want to use sets of states as basis for unsolvability certificates, we cannot represent them explicitly because this would in most cases prohibit an efficient certificate generation. Instead, we represent sets of states by logical formulas over the state variables: a state is included in the set iff its variable assignment satisfies the formula. This is a common concept also underlying the specification of action preconditions, or for representing sets of states in a regression search. In the context of unsolvability, inductive sets described by a formula are also called *traps* [Lipovetzky *et al.*, 2016].

There are many formalism for representing formulas such as reduced ordered binary decision diagrams (BDDs) or 2-CNF formulas. As different formalisms support different operations efficiently, the choice of the formalisms is crucial for an efficient certificate generation and verification.

5 Unsolvability Certificates

Eriksson *et al.* [2017] proposed unsolvability certificates based on inductive sets. In the simplest case, such a certificate consists of a single set, given as a formula in some representation formalism. The proposal does not require a specific formalism but instead identifies sufficient conditions for an efficient certificate verification. The conditions are specified as sets of operations that must be possible in polynomial time, for example constructing the conjunction of two given formulas or deciding whether a formula logically implies a given clause. The results establish for three particularly interesting formalisms, namely BDDs, Horn formulas and 2-CNF formulas, that they all are suitable for this kind of certificate.

The core of the verification step is a formulation of the successor relation induced by the actions in propositional logic. Encoding action applications as logical formulas over two sets of state variables – one for the source and one for the successor state – is not new but is the core of planning as satisfiability [Kautz and Selman, 1992] or with symbolic search [Edelkamp and Helmert, 2001].

Even if an efficient verification is possible, constructing a certificate as a single monolithic set can be computationally challenging. Consider for example a set of BDDs, where each BDD represents an inductive set. The union of all these sets is again an inductive set but representing this set as a BDD can require exponential space in the size of the ingredient BDDs.

Conjunctive and *disjunctive* certificates mitigate these problems by “factorizing” the certificate into several component sets with the property that their intersection and union, respectively, form an inductive certificate. The criteria for an efficient verification of such certificates are slightly more complicated. The certificates can be parameterized with some number r that determines how many component sets must be considered at the same time. Then r-conjunctive certificates can be verified efficiently if the components are represented as BDDs, Horn formulas or 2-CNF formulas, where the latter two formalisms do not require the restriction on a fixed r. General r-disjunctive certificates can be verified efficiently when given as BDDs. In the special case $r = 1$ also a representation as Horn formulas or 2-CNF formulas is suitable.

With these certificates, we can already transform a wide range of planning techniques into certifying algorithms.

The simplest case is blind explicit-state forward search, which must explore all states that are reachable from the initial state. If the task is unsolvable, this set is an inductive certificate that can be efficiently converted into a BDD.

Complete *heuristic search* algorithms such as A*, IDA*, or greedy best-first search exploit (unsolvability) heuristics to prune the search space. Consider the set S_{exp} of states expanded by the algorithm. Each successor state of a state in S_{exp} is either included in S_{exp} or it has been detected unsolvable by a heuristic. If the heuristics provide an inductive certificate for each such state, then the union of $\{\{s\} \mid s \in S_{\text{exp}}\}$ and all heuristic certificates is an inductive certificate, which can be efficiently expressed as a (1-)disjunctive certificate.

These certificates require heuristics that efficiently certify infinite heuristic estimates, which is possible for a large spectrum of common heuristics [Eriksson *et al.*, 2017].

For example, for the most common class of merge-and-shrink abstractions (using *linear* merge strategies), the internal representation of the abstraction can be transformed into a BDD that serves as certificate for *all* states detected unsolvable by the heuristic. However, since the construction adopts the variable order used by the merge strategy, the variable order of the certificate BDD cannot be chosen arbitrarily.

For the h^m family of critical-path heuristics, it is possible to efficiently compute inductive Horn-formula certificates, which in the common case of $m \leq 2$ are also inductive 2-SAT certificates. A compact representation as a single BDD is not possible but the individual clauses of the Horn certificate can be efficiently converted into BDD representation. The collection of all these BDDs forms a 1-conjunctive certificate.

While the proposed certificates already cover a wide range of planning algorithms, they do not allow an arbitrary combination of techniques. One reason is that all component sets of a conjunctive or disjunctive certificate must be given in the same representation formalism. For example, heuristic search with several merge-and-shrink heuristics that use different variable orderings is not expressible with the existing results. Moreover, some of the results require a specific form of the components. Consider for example heuristic search with a merge-and-shrink heuristic and h^2 from the h^m heuristic family. The result for the merge-and-shrink heuristic implies a BDD representation but h^2 can only provide *conjunctive* BDD certificates. These are not suitable for the disjunctive

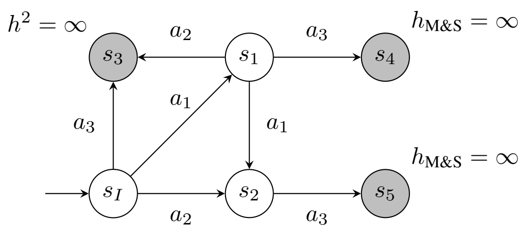

Figure 1: Explored part of the state space (Example 2)

certificate created by the search algorithm, which requires (non-conjunctive) inductive certificates from the heuristics.

6 Proof System

We suggest to use a more flexible proof system to overcome the limitations of the previously proposed unsolvability certificates. Such a system would not only allow to bridge the gap between different representation formalisms, but also would easily integrate the forward and backward reachability perspectives. This section presents a few ideas on work that is still very much in progress. We would like to give an impression how a proof system for unsolvable planning tasks could look like, but do not want to go into the formal details.

The core of the proof system will be a set of rules that we prove correct so that they can be applied without further justification in the proofs. For illustration, the following statements could occur as rules in a proof system:

(a) If the initial state is dead then the task is unsolvable.

(b) Every state in a dead set is dead.

(c) An inductive set that contains no goal state is dead.

(d) If every action application in a state s leads to a dead state then state s is dead.

The basis of each proof would be formed by a collection of statements that require verification. These can then be used as premises for rule applications. The basic statements must be separately verifiable. In the following we present two case studies to illustrate the overall concept.

Example 1

The first example shows how inductive certificates could be covered, demonstrated for a merge-and-shrink BDD certificate B.

Statements:

1. BDD set B is inductive.
2. no goal state is included in BDD B.
3. the initial state is included in BDD B.

Derivation:

4. rule (c), (1), (2) $\Rightarrow$ B is dead.
5. rule (b), (3), (4) $\Rightarrow$ the initial state is dead.
6. rule (a), (5) $\Rightarrow$ the task is unsolvable.

Example 2

The second example illustrates what a proof for heuristic search with a merge-and-shrink heuristic and the h^2 heuristic could look like. Figure 6 depicts the explored part of the search space. The search expands states s_I, s_1 and s_2. Heuristic h^2 detects s_3 as a dead end and the merge-and-shrink heuristic detects states s_4 and s_5 as a dead end.

The creation of the proof would be distributed among the different components. In the following proof, heuristic h^2 would add line (1)–(3) and (11)–(12), the merge-and-shrink heuristic would add lines (5)–(7), (9), (13)–(15) and the search the remaining lines. Note that in the example the merge-and-shrink heuristic "knows" that the BDD B suffices for all its detected dead ends, so it reuses it for state s_5 without establishing again that B is dead.

Statements:

1. Horn formula set H_{s_3} is inductive.
2. no goal state is included in Horn formula set H_{s_3}.
3. s_3 is included in H_{s_3}.
4. all action applications in state s_I lead to s_1, s_2 or s_3.
5. BDD set B is inductive.
6. no goal state is included in BDD B.
7. s_4 is included in B.
8. all action applications in state s_1 lead to s_2, s_3 or s_4.
9. s_5 is included in B.
10. all action applications in state s_2 lead to s_5.

Derivation:

11. rule (c), (1), (2) $\Rightarrow$ H_{s_3} is dead.
12. rule (b), (3), (11) $\Rightarrow$ s_3 is dead.
13. rule (c), (1), (2) $\Rightarrow$ B is dead.
14. rule (b), (7), (13) $\Rightarrow$ s_4 is dead.
15. rule (b), (9), (13) $\Rightarrow$ s_5 is dead.
16. rule (d), (10), (15) $\Rightarrow$ s_2 is dead.
17. rule (d), (8), (16), (12), (14) $\Rightarrow$ s_1 is dead.
18. rule (d), (4), (17), (16), (12) $\Rightarrow$ s_I is dead.
19. rule (a), (18) $\Rightarrow$ the task is unsolvable.

Every line specifies not only the rule, but also states where its premises have been established. Alternatively, we could require that the premises are spelled out but no references are necessary. This would move some overhead from the certifying algorithm to the verifier.

The challenge in the definition of a proof system will be the identification of a suitable set of rules and possible statements (including the representation formalisms) that allows for an efficient generation and verification but also conforms to the requirement of a complete and general approach.

7 Conclusion

Unsolvability is a very active research topic in classical planning, and the special focus of the recent planning competition has stimulated the development of a number of systems for detecting unsolvable tasks. However, if these systems report an input task to be unsolvable, we cannot easily verify this result but have to believe that it has for example not been compromised by a bug. We are therefore interested in certifying planning systems that provide additional information, allowing an independent validation of the claim. Unsatisfiability certificates that are based on inductive sets can serve for this purpose for a wide range of planning approaches. Still, they have some limitations when it comes to arbitrary combinations of techniques. We therefore suggest to generalize them to a more flexible rule-based proof system.

References

[Alcázar and Torralba, 2015] Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS 2015*, pages 2–6, 2015.

[Bäckström *et al.*, 2013] Christer Bäckström, Peter Jonsson, and Simon Ståhlberg. Fast detection of unsolvable planning instances using local consistency. In *Proc. SoCS 2013*, pages 29–37, 2013.

[Balyo and Suda, 2016] Tomáš Balyo and Martin Suda. Reachlunch entering the Unsolvability IPC 2016. In *Unsolvability IPC: planner abstracts*, pages 3–5, 2016.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *AIJ*, 129(1):5–33, 2001.

[Domshlak *et al.*, 2015] Carmel Domshlak, Jörg Hoffmann, and Michael Katz. Red-black planning: A new systematic approach to partial delete relaxation. *AIJ*, 221:73–114, 2015.

[Edelkamp and Helmert, 2001] Stefan Edelkamp and Malte Helmert. The model checking integrated planning system (MIPS). *AI Magazine*, 22(3):67–71, 2001.

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, pages 84–90, 2001.

[Eriksson *et al.*, 2017] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *Proc. ICAPS 2017*, 2017.

[Gnad *et al.*, 2016a] Daniel Gnad, Marcel Steinmetz, and Jörg Hoffmann. Django: Unchaining the power of red-black planning. In *Unsolvability IPC: planner abstracts*, pages 19–22, 2016.

[Gnad *et al.*, 2016b] Daniel Gnad, Álvaro Torralba, Jörg Hoffmann, and Martin Wehrle. Decoupled search for proving unsolvability. In *Unsolvability IPC: planner abstracts*, pages 16–18, 2016.

[Haslum and Geffner, 2000] Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS 2000*, pages 140–149, 2000.

[Haslum, 2016] Patrik Haslum. Adapting h^{++} for proving plan non-existence. In *Unsolvability IPC: planner abstracts*, pages 1–1, 2016.

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *JACM*, 61(3):16:1–63, 2014.

[Hoffmann and Fickert, 2015] Jörg Hoffmann and Maximilian Fickert. Explicit conjunctions w/o compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in polynomial time. In *Proc. ICAPS 2015*, pages 115–119, 2015.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Hoffmann *et al.*, 2014] Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In *Proc. ECAI 2014*, pages 441–446, 2014.

[Howey and Long, 2003] Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the International Planning Competition. In *ICAPS 2003 Workshop on the Competition*, 2003.

[Katz *et al.*, 2013] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Who said we need to relax *all* variables? In *Proc. ICAPS 2013*, pages 126–134, 2013.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI 1992*, pages 359–363, 1992.

[Keyder *et al.*, 2014] Emil Keyder, Jörg Hoffmann, and Patrik Haslum. Improving delete relaxation heuristics through explicitly represented conjunctions. *JAIR*, 50:487–533, 2014.

[Korovin and Suda, 2016] Konstantin Korovin and Martin Suda. iProverPlan: a system description. In *Unsolvability IPC: planner abstracts*, pages 6–7, 2016.

[Lipovetzky *et al.*, 2016] Nir Lipovetzky, Christian Muise, and Hector Geffner. Traps, invariants, and dead-ends. In *Proc. ICAPS 2016*, pages 211–215, 2016.

[McConnell *et al.*, 2011] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–162, 2011.

[Pommerening and Seipp, 2016] Florian Pommerening and Jendrik Seipp. Fast Downward dead-end pattern database. In *Unsolvability IPC: planner abstracts*, pages 2–2, 2016.

[Pommerening *et al.*, 2015] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proc. AAAI 2015*, pages 3335–3341, 2015.

[Seipp *et al.*, 2016] Jendrik Seipp, Florian Pommerening, Silvan Sievers, Martin Wehrle, Chris Fawcett, and Yusra Alkhazraji. Fast Downward Aidos. In *Unsolvability IPC: planner abstracts*, pages 28–38, 2016.

[Steinmetz and Hoffmann, 2017a] Marcel Steinmetz and Jörg Hoffmann. Search and learn: On dead-end detectors, the traps they set, and trap learning. In *Proc. IJCAI 2017*, 2017.

[Steinmetz and Hoffmann, 2017b] Marcel Steinmetz and Jörg Hoffmann. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *AIJ*, 245:1–37, 2017.

[Torralba *et al.*, 2016] Álvaro Torralba, Jörg Hoffmann, and Peter Kissmann. MS-Unsat and SimulationDominance: Merge-and-Shrink and dominance pruning for proving unsolvability. In *Unsolvability IPC: planner abstracts*, pages 12–15, 2016.

[Torralba, 2016] Álvaro Torralba. Sympa: Symbolic perimeter abstractions for proving unsolvability. In *Unsolvability IPC: planner abstracts*, pages 8–11, 2016.