A Proof System for Unsolvable Planning Tasks

Salomé Eriksson and **Gabriele Röger** and **Malte Helmert**
University of Basel, Switzerland
{salome.eriksson,gabriele.roeger,malte.helmert}@unibas.ch

Abstract

While traditionally classical planning concentrated on finding plans for solvable tasks, detecting unsolvable instances has recently attracted increasing interest. To preclude wrong results, it is desirable that the planning system provides a certificate of unsolvability that can be independently verified. We propose a rule-based proof system for unsolvability where a proof establishes a knowledge base of verifiable basic statements and applies a set of derivation rules to infer the unsolvability of the task from these statements. We argue that this approach is more flexible than a recent proposal of inductive certificates of unsolvability and show how our proof system can be used for a wide range of planning techniques.

Introduction

The classical planning problem is defined as finding a sequence of actions which reaches a goal state from a given initial state, or to prove that such a sequence does not exist. Most current planning systems focus on how to find plans as fast as possible, with little attention paid to proving the non-existence of plans until recently. The Unsolvability IPC (Muise and Lipovetzky 2016) was held in an effort to remedy this and resulted in many interesting new techniques such as dead-end potentials (Seipp et al. 2016) or an incremental learning algorithm inspired by clause learning for satisfiability problems (Steinmetz and Hoffmann 2017).

In the SAT community it is common practice that a solver that identifies a formula as unsolvable can generate a proof of unsolvability that can be verified independently by another program. In our previous work (Eriksson, Röger, and Helmert 2017) we introduced a similar notion of unsolvability certificates for planning problems. These so-called *inductive certificates* have several desirable properties: they are complete in the sense that for every unsolvable task there exists a certificate that certifies unsolvability; they are efficiently verifiable in the sense that an independent verifier can validate certificates in polynomial time in the certificate size; and they are fairly general in the sense that a number of commonly used planning techniques can be modified to generate certificates of the required form with polynomial overhead. The list of supported planning algorithms includes blind search, symbolic search and heuristic search with certain heuristics.

However, these certificates also have undesirable properties. In particular, they are quite *monolithic* in the sense that they contain no or little substructure. This makes them complex to understand and reason about. For example, debugging an algorithm that produces invalid certificates of this form is quite difficult because it is hard to pinpoint a human-understandable (in particular, reasonably-sized) reason for invalidity. Most importantly, the certificates are not *compositional*. For example, while our previous approach could create certificates of unsolvability for A^* with merge-and-shrink heuristics and for A^* with the h^2 heuristic, it could not create certificates of unsolvability for A^* with *both* of these heuristics because the two heuristics require incompatible forms of representation.

In this paper we introduce computer-verifiable *proofs of unsolvability* as a new, compositional form of unsolvability certificates which addresses these shortcomings. Proofs consist of *basic statements* that can be verified immediately and *derivation steps* that use knowledge derived in previous proof steps to generate new knowledge. The approach is compositional in the sense that different subproofs can be easily combined, and it is easy to point out at which exact step an invalid proof fails. Apart from theory, we also provide an implementation of a proof verifier and proof-generating variants of several planning algorithms from the literature, including ones our previous approach cannot deal with.

Background

We consider propositional *STRIPS planning tasks* (Fikes and Nilsson 1971), which are given by a tuple $\Pi = \langle V, A, I, G\rangle$ consisting of a finite set of *state variables* V, a finite set of *actions* A, the *initial state* $I \subseteq V$ and the *goal specification* $G \subseteq V$. We write the size of the representation of Π as $\|\Pi\|$.

A *state* $s \subseteq V$ is given by the variables that are *true* in s. The set of all states is $S(\Pi) = 2^V$. The set of *goal states* is $S_G(\Pi) = \{s \in S(\Pi) \mid G \subseteq s\}$.

An action $a \in A$ is represented by a triple $a = \langle \mathit{pre}(a), \mathit{add}(a), \mathit{del}(a)\rangle$ with $\mathit{pre}(a), \mathit{add}(a), \mathit{del}(a) \subseteq V$. It is *applicable* in state s if $\mathit{pre}(a) \subseteq s$, in which case the resulting *successor state* is $s[a] = (s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.

For state s and action sequence π, applying π in s results

in state $s[\pi]$ defined by $s[\langle\rangle] = s$ and $s[\pi'\langle a\rangle] = s[\pi'][a]$. (This requires that a is applicable in $s[\pi']$.) An *s-plan* for state s is an action sequence π such that $s[\pi] \in S_G(\Pi)$. We say that s-plan π *traverses* state s' if $s' = s[\pi']$ for some prefix π' of π. An I-plan is also just called a *plan* for Π. The task is *unsolvable* if there is no I-plan.

For a set $S' \subseteq S(\Pi)$ of states and a set $A' \subseteq A$ of actions, the *progression* of S' w.r.t. A' is $S'[A'] = \{s[a] \mid s \in S', a \in A', a \text{ applicable in } s\}$, containing all successors of state in S' via actions in A'. The *regression* of S' w.r.t. A' is $[A']S' = \{s \mid s[a] \in S', a \in A', a \text{ applicable in } s\}$, containing all predecessors of S' via A'.

Proof System

A *proof* in the proposed proof system consists of two parts: a collection of *basic statements* that need to be verified individually and a *derivation* based on these statements that shows that the task is unsolvable. The derivation consists of a sequence of *derivation steps*, which instantiate a prespecified set of *derivation rules* that are universally true and hence do not need to be verified again. The verifier only needs to confirm that the premises of all derivation steps have been established and that the derivation rules have been instantiated correctly.

We first introduce a useful set of derivation rules and the form of basic statements they require. In the following section, we discuss how to efficiently verify these statements.

Derivation Rules

Our proof system is based on the notion of *dead* states. These are states that cannot be traversed by any plan.

Definition 1 (Dead state)**.** *A state s is dead if no plan traverses s. A set of states is dead if all its elements are dead.*

There are two reasons why a state might be dead: there is no s-plan, or s is unreachable from the initial state I. If a state s is both reachable from I and there exists an s-plan, then it is possible to construct a plan that traverses s.

In general, it is PSPACE-complete to decide whether a given state is dead because a planning task is solvable iff I is not dead. But in principle, all planning systems that prove a task unsolvable use some techniques that can be used to argue that the initial state or the set of goal states are dead. In the following, we identify rules that allow making this underlying argumentation explicit for a wide range of techniques. We first discuss and prove correctness of these rules separately and then give an overview of all derivation rules.

The first three rules follow directly from the definition of dead state sets as sets of dead states.

- $\emptyset$ is dead
- The union of dead state sets is dead.
- Subsets of dead state sets are dead.

The next rules define sufficient criteria for unsolvability and will form the end of each derivation:

Theorem 1. *If the initial state is dead or all goal states are dead, then the task is unsolvable.*

Proof. If the initial state is dead, no action sequence can be a plan because every plan must traverse the initial state.

If all goal states are dead, no action sequence can be a plan because every plan must traverse a goal state. □

The following rules build the core of the proof system. They identify state sets as dead that cannot be left except via dead states or that cannot be reached from the outside, except from dead states. For progression, we can state the following rules:

Theorem 2. *Let S be a set of states and S' a dead set of states such that all successors of states $s \in S$ are in S or S', i.e., $S[A] \subseteq S \cup S'$.*

(i) If $S \cap S_G(\Pi)$ is dead, then S is dead.

(ii) If $I \in S$, then $\overline{S}$ is dead.

Proof. For any plan $\pi = \langle a_1, \ldots, a_n\rangle$ traversing a state from S we can show that the plan will never leave S again. More formally, if $I[\langle a_1, \ldots, a_i\rangle] \in S$ for some $i \leq n$, then $I[\langle a_1, \ldots, a_j\rangle] \in S$ for all $i \leq j \leq n$.

For $i = n$ we have nothing to show. Otherwise, consider $s = I[\langle a_1, \ldots, a_{i+1}\rangle]$. From $S[A] \subseteq S \cup S'$ we know that $s \in S \cup S'$. But since S' is dead and s is part of a plan and thus cannot be dead, we conclude that $s \in S$. Applying this argument iteratively proves the claim.

(i) By contradiction: assume that $S \cap S_G(\Pi)$ is dead and S is not dead. Then a plan π traversing a state in S exists. We have shown that $I[\pi] \in S$. Because π is a plan, we also get that $I[\pi] \in S_G(\Pi)$ and that $I[\pi]$ is not dead. This contradicts the fact that $S \cap S_G(\Pi)$ is dead.

(ii) From $I \in S$ we know that no plan leaves S, and hence $s \notin S$ cannot be traversed by a plan. Hence $\overline{S}$ is dead. □

For regression, we can formulate similar rules:

Theorem 3. *Let S be a set of states and S' a dead set of states such that all predecessors of states $s \in S$ are in S or S', i.e., $[A]S \subseteq S \cup S'$.*

(i) If $\overline{S} \cap S_G(\Pi)$ is dead, then $\overline{S}$ is dead.

(ii) If $I \notin S$, then S is dead.

Proof. Similarly to progression, we can argue that if a plan $\pi = \langle a_1, \ldots, a_n\rangle$ traverses a state from S, it cannot have traversed a state from $\overline{S}$ earlier, or more formally: if $I[\langle a_1, \ldots, a_i\rangle] \in S$ for some $i \leq n$, then for all $j \leq i$ we get $I[\langle a_1, \ldots, a_j\rangle] \in S$ and in particular $I[\langle\rangle] = I \in S$.

(i) If $\overline{S} \cap S_G(\Pi)$ is dead, then all plans must end in a goal state in S. With the above argument the plans traverse only states from S, so $\overline{S}$ is dead.

(ii) If a plan traverses a state from S, then I is in S. As $I \notin S$, there cannot be such a plan, and hence S is dead. □

The next rules describes a connection between progression and regression.

Theorem 4. *$S[A] \subseteq S'$ iff $[A]\overline{S'} \subseteq \overline{S}$.*

Proof. If $S[A] \subseteq S'$, then all transitions originating from a state in S lead to a state in S'. This means that for all states not in S' (i.e., $s \in \overline{S'}$) their predecessors cannot lie in S and hence $[A]\overline{S'} \subseteq \overline{S}$.

If $[A]\overline{S'} \subseteq \overline{S}$, then all transitions leading to a state in $\overline{S'}$ originate from a state in $\overline{S}$. This means that for all states in S, their successors must lie in S' and hence $S[A] \subseteq S'$. □

We are now ready to list all derivation rules. They include the basic properties of dead sets (**D1**–**D3**) and the rules proved in Theorems 1–4 (**D4**–**D11**).

Definition 2. *Proof system – derivation rules*

D1 $\emptyset$ *is dead.*

D2 *If* S *is dead and* S' *is dead, then* $S \cup S'$ *is dead.*

D3 *If* $S \subseteq S'$ *and* S' *is dead, then* S *is dead.*

D4 *If* $\{I\}$ *is dead, then the task is unsolvable.*

D5 *If* $S_G(\Pi)$ *is dead, then the task is unsolvable.*

D6 *If* $S[A] \subseteq S \cup S'$, S' *is dead and* $S \cap S_G(\Pi)$ *is dead, then* S *is dead.*

D7 *If* $S[A] \subseteq S \cup S'$, S' *is dead and* $\{I\} \subseteq S$, *then* $\overline{S}$ *is dead.*

D8 *If* $[A]S \subseteq S \cup S'$, S' *is dead and* $\overline{S} \cap S_G(\Pi)$ *is dead, then* $\overline{S}$ *is dead.*

D9 *If* $[A]S \subseteq S \cup S'$, S' *is dead and* $\{I\} \subseteq \overline{S}$, *then* S *is dead.*

D10 *If* $[A]S \subseteq S'$, *then* $\overline{S'}[A] \subseteq \overline{S}$.

D11 *If* $S[A] \subseteq S'$, *then* $[A]\overline{S'} \subseteq \overline{S}$.

The derivation rules serve as templates for derivation steps used in a proof. Derivation steps are obtained by replacing the set placeholders (S, S', S'') in a rule by set expressions. A set expression can be a set variable (X, X' etc.), the constant sets $\emptyset$, $\{I\}$ or $S_G(\Pi)$, or formed from set expressions S and S' by union ($S \cup S'$), intersection ($S \cap S'$), complement ($\overline{S}$), progression ($S[A]$) or regression ($[A]S$).

The knowledge base on which the rules operate consists of atomic statements of the form "$S \subseteq S'$" or "S is dead", where S and S' are set expressions. (We use $\{I\} \subseteq S$ and $\{I\} \subseteq \overline{S}$ in the rules above instead of the more natural $I \in S$ and $I \notin S$ to not require another form of atomic statement.) A derivation step may be applied if all its premises have been previously derived in the knowledge base. Applying it adds its consequence to the knowledge base. Deriving the special statement "the task is unsolvable" concludes the proof.

As is usual in proof calculi, applying a derivation rule is a purely syntactic operation where the set expressions and statements are *not* interpreted. For example, if a rule premise requires the statement $S[A] \subseteq S \cup S'$, it is not sufficient to know $S[A] \subseteq S''$ for some set S'' that happens to contain exactly the elements of $S \cup S'$. The premise needs to be derived precisely for the set expression $S \cup S'$. This way the verification of derivation rules is a simple lookup with no added complexity which might arise when interpreting set expressions.

Basic Statements

In addition to applying derivation rules, a proof of unsolvability needs some kind of initial knowledge base as a starting point. (After all, none of the derivation rules depend on the specific planning task in question, and of course unsolvability cannot be proven without appealing to any properties of the planning task.)

In the proposed proof system the initial knowledge base is specified by introducing *set variables* to which specific sets of states are assigned (using representations discussed in the next section) and *basic statements* about these set variables that need to be verified.

A proof then consists of two parts. The first part is an initial knowledge base of basic statements, where set variables are interpreted according to their definition. The second part is a derivation based solely on syntactic application of derivation rules. In the derivation, set expressions are seen as abstract expressions rather than concrete sets, and the derivation steps must be ordered such that for each rule application the premise has already been established before (either by basic statements or earlier rule applications).

As mentioned before, our knowledge bases consist of statements of the form "$S \subseteq S'$" and "S is dead", where S and S' are set expressions. Because proofs should be efficiently verifiable, we only allow an efficiently verifiable subset of these statements as basic statements. Our intent here is to be minimalist and not include basic statements that can easily be derived from other, "more basic" statements. Specifically, we only allow basic statements of the form $S \subseteq S'$, and these only for limited set expressions S and S'. The following definition lists the permitted basic statements, where X, X' and X'' are set variables and L and L' are atomic set term literals (set variables X, set constants $\emptyset$, $\{I\}$ or $S_G(\Pi)$, or the complements of set variables or constants).

Definition 3. *Proof system – basic statements*

B1 $L \subseteq L'$

B2 $X \subseteq X' \cup X''$

B3 $L \cap S_G(\Pi) \subseteq L'$

B4 $X[A] \subseteq X \cup L$

B5 $[A]X \subseteq X \cup L$

Representation of State Sets

The actual representation of the sets that the set variables refer to is important for two reasons. Firstly, we need to be able to represent large sets compactly; otherwise, polynomial-sized unsolvability proofs can only exist for planning tasks that can be proven unsolvable by a form of blind explicit-state search (progression or regression).

Secondly, we must be able to verify the basic statements **B1**–**B5** in polynomial time in the representation size of the sets involved and the planning task, or else the overall verification algorithm will not be polynomial in its input.

Following our model from Eriksson et al. (2017), we use restricted classes of logical formulas and formula-like structures to obtain compact set representations, where a formula

φ over the state variables represents all states that are satisfying assignments for φ. Specifically, we consider the following four representations for state sets:

- explicit sets of states
- (reduced ordered) BDDs (Bryant 1985)
- Horn formulas (conjunctions of Horn clauses)
- 2CNF formulas (conjunctions of unary or binary clauses)

For example, to show that BDDs are a suitable representation for basic statements **B2** ($X \subseteq X' \cup X''$), we must show that if state sets X, X' and X'' are represented as BDDs, then the test $X \subseteq X \cup X''$ can be performed in polynomial time in the representation size (number of nodes) of the BDDs involved.

The results in this section can be generalized to other forms of representations by making use of concepts from the area of knowledge compilation (Darwiche and Marquis 2002), but for space reasons we focus on the four representations above in the following.

Homogeneous Representations

For statements **B2**–**B5**, we only permit homogeneous representations, i.e., the different set variables involved in these statements must be of the same representation. (For BDDs, this also means that they must use the same variable order.) To bridge between different representations, we permit mixing different representations for **B1**, which we discuss later.

We first prove the following lemma:

Lemma 1. *The following entailment queries for CNF formulas can be answered in polynomial time (unless stated otherwise, all formulas may be general CNF formulas):*

(a) $\varphi \models \varphi'$*, where* φ *is Horn or 2CNF*

(b) $\varphi \models \varphi' \vee \varphi''$*, where* φ *is Horn or 2CNF*

(c) $\varphi \models \varphi' \vee \neg\varphi''$*, where* φ *and* φ'' *are Horn*

(d) $\varphi \models \varphi' \vee \neg\varphi''$*, where* φ *and* φ'' *are 2CNF*

Proof. (a): We have $\varphi \models \varphi'$ iff $\varphi \models c$ for all clauses c of φ' iff $\varphi \wedge \neg c$ is unsatisfiable for all clauses c of φ'. This can be tested in polynomial time because $\varphi \wedge \neg c$ is a Horn/2CNF formula if φ is a Horn/2CNF formula. (Note that $\neg c$ is a conjunction of literals and hence both Horn and 2CNF.)

(b): Transform $\varphi' \vee \varphi''$ into a CNF formula. This can be done in time $O(\|\varphi'\| \cdot \|\varphi''\|)$ because $\varphi' \vee \varphi'' = \bigwedge_i c'_i \vee \bigwedge_j c''_j \equiv \bigwedge_i \bigwedge_j (c'_i \vee c''_j)$. Then apply (a).

(c) + (d): We have $\varphi \models \varphi' \vee \neg\varphi''$ iff $\varphi \wedge \varphi'' \models \varphi'$. The formula $\varphi \wedge \varphi''$ is Horn/2CNF, so apply (a). □

We are now ready to prove our results. We use symbols like φ to refer to the set representations (explicit sets, BDDs, Horn/2CNF formulas), which we also treat as logical formulas, and we use $[\![\varphi]\!]$ to refer to the represented set of states.

Explicit sets can be converted into BDDs in polynomial time, so whenever an operation is efficiently possible with BDDs, it is also efficiently possible with explicit sets.

Theorem 5. *For homogeneous representations as explicit sets, BDDs, Horn or 2CNF formulas, the statement*

$$\textbf{B2} \quad [\![\varphi]\!] \subseteq [\![\varphi']\!] \cup [\![\varphi'']\!]$$

can be verified in polynomial time.

Proof. We need to test $\varphi \models \varphi' \vee \varphi''$. For BDDs, this is equivalent to testing that $\varphi \wedge \neg(\varphi' \vee \varphi'')$ is the empty BDD, and all logical connectives and emptiness tests are polynomial operations for BDDs. This also covers explicit sets. For Horn and 2CNF, we apply Lemma 1. □

The remaining statements are more complex because they involve not just set variables but more general set term literals, which can include set constants ($\emptyset$, $\{I\}$, $S_G(\Pi)$) and complements. However, all set constants can be represented compactly as BDDs, Horn and 2CNF formulas, so it is sufficient to cover set variables and complemented set variables.

Theorem 6. *For homogeneous representations as explicit sets, BDDs, Horn or 2CNF formulas, the statements*

$$\textbf{B3a} \quad [\![\varphi]\!] \cap S_G(\Pi) \subseteq [\![\varphi']\!]$$
$$\textbf{B3b} \quad \overline{[\![\varphi]\!]} \cap S_G(\Pi) \subseteq [\![\varphi']\!]$$
$$\textbf{B3c} \quad [\![\varphi]\!] \cap S_G(\Pi) \subseteq \overline{[\![\varphi']\!]}$$
$$\textbf{B3d} \quad \overline{[\![\varphi]\!]} \cap S_G(\Pi) \subseteq \overline{[\![\varphi']\!]}$$

can be verified in polynomial time.

Proof. For BDDs (and hence also for explicit sets), all cases are polynomial because they involve a bounded number of Boolean operations and an emptiness test.

For Horn/2CNF, let γ be a formula describing the goal: $[\![\gamma]\!] = S_G(\Pi)$. Note that γ is Horn and 2CNF (it is a conjunction of atoms). We must test $\varphi \wedge \gamma \models \varphi'$ and the related queries with $\neg\varphi$ instead of φ and/or $\neg\varphi'$ instead of φ'.

(a): $\varphi \wedge \gamma \models \varphi'$ is an entailment test for a Horn/2CNF formula ($\varphi \wedge \gamma$ is Horn/2CNF) and hence polynomial.

(b): We have $\neg\varphi \wedge \gamma \models \varphi'$ iff $\neg\varphi \wedge \gamma \wedge \neg\varphi'$ is unsatisfiable. We can rewrite this formula as $\neg\varphi \wedge \neg\varphi' \wedge \gamma \equiv \neg(\varphi \vee \varphi') \wedge \gamma$, which is unsatisfiable iff $\gamma \models \varphi \vee \varphi'$. This can be tested in polynomial time by Lemma 1(b).

(c): $\varphi \wedge \gamma \models \neg\varphi'$ holds iff $\varphi \wedge \gamma \wedge \varphi'$ is unsatisfiable. This is a Horn/2CNF formula, so polynomially testable.

(d): $\neg\varphi \wedge \gamma \models \neg\varphi'$ holds iff $\varphi' \wedge \gamma \models \varphi$ (contraposition with background knowledge γ), so this reduces to (a). □

We continue with the results for **B4** and **B5**.

Theorem 7. *For homogeneous representations as explicit sets, BDDs, Horn or 2CNF formulas, the statements*

$$\textbf{B4a} \quad [\![\varphi]\!][A] \subseteq [\![\varphi]\!] \cup [\![\varphi']\!]$$
$$\textbf{B4b} \quad [\![\varphi]\!][A] \subseteq [\![\varphi]\!] \cup \overline{[\![\varphi']\!]}$$
$$\textbf{B5a} \quad [A][\![\varphi]\!] \subseteq [\![\varphi]\!] \cup [\![\varphi']\!]$$
$$\textbf{B5b} \quad [A][\![\varphi]\!] \subseteq [\![\varphi]\!] \cup \overline{[\![\varphi']\!]}$$

can be verified in polynomial time.

Proof. We can test the statements by individually testing each action. For example, for (4a) we test $[\![\varphi]\!][\{a\}] \subseteq [\![\varphi]\!] \cup [\![\varphi']\!]$ for all $a \in A$.

For (4a) and (4b), Theorem 4 of Eriksson et al. (2017) describes how to express the test $[\![\varphi]\!][\{a\}] \subseteq [\![\psi]\!]$ as an entailment test of the form $\alpha(\varphi, a) \models \psi$, where $\alpha(\varphi, a)$ is a BDD/Horn formula/2CNF formula if φ is and can be computed in polynomial time. They do not consider an arbitrary formula ψ, but the proof generalizes directly to this case.

It remains to test $\alpha(\varphi, a) \models \psi$, where $\psi = \varphi \vee \varphi'$ for part (a) and $\psi = \varphi \vee \neg\varphi'$ for part (b). This is possible in polynomial time for Horn/2CNF formulas (Lemma 1) and also for BDDs.

The same approach works for (5a) and (5b) with trivial changes to the argument. □

Bridging Representation Formalisms

To conclude this section, we consider the remaining basic statement **B1** of the form $L \subseteq L'$. This statement is somewhat different in that we allow to use *different* representations for L and L' – for example, BDDs for L and Horn formulas for L'. This freedom is important to permit proofs where different subproofs need different representation formalisms for efficient reasoning.

Like for the previous rules, it is sufficient to consider the cases where L and L' are set variables or complemented set variables (rather than constants or their complements).

Theorem 8. *For homogeneous representations as explicit sets, BDDs, Horn or 2CNF formulas, the statements*

$$\textbf{B1a} \quad [\![\varphi]\!] \subseteq [\![\varphi']\!]$$
$$\textbf{B1b} \quad \overline{[\![\varphi]\!]} \subseteq [\![\varphi']\!]$$
$$\textbf{B1c} \quad [\![\varphi]\!] \subseteq \overline{[\![\varphi']\!]}$$
$$\textbf{B1d} \quad \overline{[\![\varphi]\!]} \subseteq \overline{[\![\varphi']\!]}$$

can be verified in polynomial time.

Moreover, **B1a** *can also be verified in polynomial time for all combinations of representations except those where φ' is represented as a BDD and φ is represented as a BDD with another variable order, a Horn formula, or a 2CNF formula.*

Proof. The homogeneous representation case is covered by the special case of Theorem 6 where all states are goal states.

For the mixed cases of **B1a**, all cases where φ' is a Horn or 2CNF formula can be tested by testing separately for each clause c of φ' that $\varphi \models c$. All considered formalisms support clausal entailment in polynomial time.

All cases where φ' is an explicit set can be tested by using efficient incremental model enumeration algorithms: see Bryant (1986) for BDDs and Dechter and Itai (1992) for Horn and 2CNF.

The remaining permitted case is where φ is an explicit set and φ' is a BDD. For this it is sufficient to explicitly test that all elements of $[\![\varphi]\!]$ are models of φ'. □

We remark that it is mainly for simplicity and brevity that we only allow mixed representations in the case **B1a** (i.e., without complemented variables). Mixed representations could also be allowed in certain other cases. For example, all cases of **B1** where the right-hand side is the complement of an explicitly represented set are easy to verify.

Relationship to Inductive Certificates

Having completed the presentation of the proposed proof system, we now turn to its practical uses. Firstly, we compare it to our previous approach of unsolvability certificates.

All unsolvability certificates in our previous approach are based on *inductive sets*, which are sets S of states that are closed under progression, i.e., $S[A] \subseteq S$. The most basic version are *inductive certificates*.

Definition 4 (Inductive Certificate, Eriksson, Röger, and Helmert 2017)**.** *An* inductive certificate for a state s *of task* Π *is given by a set* $S \subseteq S(\Pi)$ *of states such that*

1. $s \in S$,
2. $S \cap S_G(\Pi) = \emptyset$, *and*
3. S *is an inductive set.*

An inductive certificate for the initial state I is also called an inductive certificate for Π.

Inductive certificates for s prove that s is dead, and inductive certificates for Π prove that Π is unsolvable. We now show that these certificates can easily be transformed into proofs in our proof system.

Theorem 9. *Given an inductive certificate for state s, we can construct a proof in the proof system that $\{s\}$ is dead in linear time. Given an inductive certificate for* Π*, we can construct a proof that* Π *is unsolvable in linear time.*

Proof. Let S be an inductive certificate for state s.

We define the sets $X_s = \{s\}$ and $X_S = S$ and use the basic statements (1): $X_s \subseteq X_S$ (**B1**), (2): $X_S \cap S_G(\Pi) \subseteq \emptyset$ (**B3**) and (3): $X_S[A] \subseteq X_S \cup \emptyset$ (**B4**).

Using **D1** we derive (4): $\emptyset$ is dead. Using **D3** with (2), (4) we derive (5): $X_S \cap S_G(\Pi)$ is dead. Using **D6** with (3), (4), (5) we derive (6): X_S is dead. Finally, using **D3** with (1), (6) we derive (7): X_s is dead, establishing that $\{s\}$ is dead.

If $s = I$, we can use $\{I\}$ in place of X_s in (1) and (7) and use **D4** with (7) to show that the task is unsolvable. □

We remark that our previous work showed that inductive certificates are *complete* (i.e., exist for every unsolvable task), which also translates to our proof system:

Corollary 1. *The proposed proof system is complete.*

However, this does not mean that *small* certificates (or short proofs) exist in all cases. Because "regular" inductive certificates are often not compact, we previously also introduced so-called *r-disjunctive* and *r-conjunctive* certificates, where $r \in \mathbb{N}$ is a parameter. These cannot be (easily) converted into proofs in our proof system. For example, one of the mismatches is that such certificates allow case distinctions for different actions where our proof system always considers all actions together. In future work, it would be interesting to refine our proof system to fully cover all these certificates.

Note, though, that even without having a direct compilation from r-disjunctive/r-conjunctive certificates, our proof

system covers all planning approaches covered in our previous work and more, as we will see in the following section. This is because the main reason why the advanced certificates are needed is due to a lack of compositionality in their approach, a weakness that our proof system does not share.

Applications to Current Planning Systems

In this section we present how a variety of techniques for proving unsolvability can be augmented to produce a proof of unsolvability in our proof system with only polynomial overhead. We focus on approaches that are not already covered by inductive certificates.

Heuristic Search with Multiple Heuristics

Our previous approach covers heuristic search with certain heuristics, such as A* with the h^2 heuristic (Haslum and Geffner 2000) or A* with linear merge-and-shrink heuristics (Helmert et al. 2014). However, it does not cover A* using *both* heuristics, as h^2 requires Horn or 2CNF certificates and merge-and-shrink requires BDD certificates, and the two cannot be combined. For similar reasons, the approach does not cover A* with two merge-and-shrink heuristics based on different variable orders.

With the proposed proof system, we can combine information from an arbitrary set of heuristics as long as for each state where $h(s) = \infty$ according to some heuristic h, we can produce a proof that $\{s\}$ is dead. As seen in the previous section, this is (for example) the case whenever there exists an inductive certificate for s. We showed in our previous work that compact certificates can be generated for linear merge-and-shrink heuristics (Helmert et al. 2014), PDB heuristics (Edelkamp 2001), delete relaxation heuristics (Bonet and Geffner 2001; Hoffmann and Nebel 2001), the h^m critical-path heuristics (Haslum and Geffner 2000) and landmark heuristics based on delete relaxation (Helmert and Domshlak 2009) or the Π^m compilation (Keyder, Richter, and Helmert 2010). Below, we extend this result to the h^C critical-path heuristics that generalize h^m (Keyder, Hoffmann, and Haslum 2014). Here, we show how information from multiple heuristics can be combined to form an overall proof of unsolvability.

We describe a simple polynomial algorithm that produces a proof. More efficient algorithms are possible. Let E be the set of states expanded during search, which implies $I \in E$ (except in the easy case where the initial state is already pruned by a heuristic). Let $D = \{d_1, \dots, d_n\}$ be the set of successors of expanded states that were detected as dead-ends by some heuristic. For the search algorithm to have terminated without finding a solution, we must have $E[A] \subseteq E \cup D$ and $E \cap S_G(\Pi) = \emptyset$. We assume that we have already proved that the singleton sets $D_1 = \{d_1\}, \dots, D_n = \{d_n\}$ are dead, e.g., using inductive certificates as described above. We use an explicit set representation for the sets D_i and for the set E.

Next, we define explicit sets $C_1 = \{d_1\}$, $C_2 = \{d_1, d_2\}$, $\dots$, $C_n = \{d_1, \dots, d_n\}$ and add the basic statements $C_1 \subseteq D_1$ (**B1**), $C_2 \subseteq C_1 \cup D_2$ (**B2**), $\dots$, $C_n \subseteq C_{n-1} \cup D_n$ (**B2**). We prove that all sets C_i are dead by using **D3** for C_1 and **D2** for all other C_i. In particular, we obtain (A) "C_n is dead".

We prove (B) "$E \cap S_G(\Pi)$ is dead" via the basic statement $E \cap S_G(\Pi) \subseteq \emptyset$ (**B3**) and a derivation using **D1** and **D3**.

We add (C) "$E[A] \subseteq E \cup C_n$" as a basic statement (**B4**).

Using (C), (A), (B) in **D6**, we obtain that E is dead.

From there, using a basic statement $\{I\} \subseteq E$, we derive that $\{I\}$ is dead using **D3** and conclude that the task is unsolvable using **D4**.

Clause-Learning State-Space Search

Steinmetz and Hoffmann (2016; 2017) describe a search algorithm built upon the h^C heuristic family where the heuristic is incrementally improved to detect more dead-ends. We focus on the use of the algorithm for proving unsolvability.

The algorithm can be run with or without an *early termination* option. Without this option, the algorithm guarantees $h^C(I) = \infty$ for the final heuristic. With the option, the algorithm generates a depth-first search space where states are pruned by different (increasingly stronger) variants of the h^C heuristic. Both cases are covered by our proof system via our general argument for heuristic search with (one or multiple) heuristics provided that we can prove that states with $h^C(s) = \infty$ are dead. We show that such states have an inductive certificate that can be represented as a Horn formula, from which the result follows with Theorem 9.

The heuristic h^C is parameterized by a set C of *conjunctions*, where each conjunction is a set of state variables (Keyder, Hoffmann, and Haslum 2014; Steinmetz and Hoffmann 2017). For the following argument, it is useful to consider h^C as a heuristic with two arguments, where $h^C(s, G')$ estimates the distance from state s to (sub-)goal G'. When viewed in this way, h^C is admissible and consistent in both arguments, w.r.t. progression for s and regression for G'.

For a given state s, h^C is based on admissible and consistent distance estimates $d(s, c)$ for reaching each conjunction $c \in C$, setting $h^C(s, G') = \max_{c \in C, c \subseteq G'} d(s, c)$.

Consider a state s with $h^C(s) = h^C(s, G) = \infty$. Let $C_\infty(s) = \{c \in C \mid d(s, c) = \infty\}$. Let $S = \{s' \in S(\Pi) \mid c \not\subseteq s' \text{ for all } c \in C_\infty(s)\}$, i.e., S is the set of states that h^C *considers reachable from* s. S can be represented by the Horn formula $\bigwedge_{c \in C_\infty(s)} \bigvee_{v \in c} \neg v$.

We show that S is an inductive certificate for s. Firstly, we have $s \in S$ because conjunctions considered unreachable from s cannot be true in s (due to the admissibility of h^C). Secondly, we have $S \cap S_G(\Pi) = \emptyset$: because $h^C(s) = \infty$, there is a goal conjunction $c \subseteq G$ in $C_\infty(s)$, and this conjunction must be present in all goal states. Finally, we have $S[A] \subseteq S$ by contradiction: otherwise we could transition from a state considered reachable from s to a state considered unreachable from s, which violates the consistency of h^C in the regression space.

Iterative Dead Pairs Calculation

Alcázar and Torralba (2015) describe an algorithm which finds pairs of SAS$^+$ facts $\{p, q\}$ such that any state satisfying $p \wedge q$ is dead. In the STRIPS setting, the most faithful adaptation of the algorithm considers pairs of *literals*, i.e., p and q can be state variables or negated state variables. Such fact pairs are sometimes called "forward/backward mutexes",

but since they are not mutexes in the strict sense, we call them *dead pairs* instead. A *dead pair set* D is a set of dead pairs. A state is *consistent with* D if it contains no pair in D; otherwise it is *pruned* by D. Pruned states are dead.

The algorithm by Alcázar and Torralba alternatingly performs forward and backward steps. The k-th step computes a dead pair set D_k, exploiting that D_{k-1} is already a known dead pair set (beginning with $D_0 = \emptyset$). If the k-th iteration is forward, the algorithm performs an h^2-style reachability analysis using D_{k-1} as background knowledge, i.e., ignoring states pruned by D_{k-1}. The backward iterations are similar, but using a backward h^2-style analysis.

In a *forward* step, the new dead pair set D_k is the (uniquely defined) maximal set of pairs such that: (a) if I is consistent with D_{k-1}, then I is consistent with D_k, and (b) for all transitions $s \to s'$ where s is consistent with D_k and s' is consistent with D_{k-1}, s' is consistent with D_k. This characterization is not apparent from the description by Alcázar and Torralba, but it follows directly from the description of Rintanen's (2008) invariant synthesis algorithm, which is equivalent to h^2. Note that in a STRIPS setting the first forward iteration will find all mutex information encoded in the multivalued variables of a SAS$^+$ task.

Similarly, in a *backward* step, the new dead pair set D_k is the (uniquely defined) maximal set of pairs such that: (a') all goal states consistent with D_{k-1} are consistent with D_k, and (b') for all transitions $s \to s'$ where s' is consistent with D_k and s is consistent with D_{k-1}, s is consistent with D_k.

Based on these characterizations, we can construct a proof in our proof system that all states pruned by D_k are dead. Let S_k be the set of states consistent with D_k, which can be described by the 2CNF formula $\bigwedge_{\{\ell_1,\ell_2\}\in D_k}(\overline{\ell_1} \vee \overline{\ell_2})$, where $\overline{\ell}$ denotes the complement of the literal ℓ. We prove that all sets $\overline{S_k}$ are dead. For $k = 0$, this is easily proved from $\overline{S_0} \subseteq \emptyset$ (**B1**) using **D1** and **D3**.

For $k > 0$, we have already proved (1): $\overline{S_{k-1}}$ is dead. In a forward step, we can assume $I \notin \overline{S_{k-1}}$ or else we can already prove unsolvability. Hence we can use the basic statement (2): $\{I\} \subseteq S_k$ (**B1**). From (b) we get the basic statement (3): $S_k[A] \subseteq S_k \cup \overline{S_{k-1}}$ (**B4**). Using (3), (1), (2) in **D7**, we derive that $\overline{S_k}$ is dead as required.

In a backward step, (a') can be rephrased as "all goal states not consistent with D_k are not consistent with D_{k-1}", yielding the basic statement: $\overline{S_k} \cap S_G(\Pi) \subseteq \overline{S_{k-1}}$ (**B3**), which together with (1) proves (2'): $\overline{S_k} \cap S_G(\Pi)$ is dead (**D3**). From (b') we obtain the basic statement (3'): $[A]S_k \subseteq S_k \cup \overline{S_{k-1}}$ (**B5**). Using (3'), (1), (2') in **D8**, we derive that $\overline{S_k}$ is dead as required.

In summary, we can generate a compact proof in our proof system that all states that are prunable according to the iterative dead pairs algorithm are dead. Using **D3**, it is also easy to extract fine-grained results of the form "All states satisfying $\ell_1 \wedge \ell_2$ are dead" for each dead pair $\{\ell_1, \ell_2\}$, which can be converted to representations other than 2CNF (e.g., BDDs or Horn clauses) and used in a larger overall proof. For example, it is not difficult to augment a proof of unsolvability for explicit search to include pruning of states that satisfy a dead pair. We conjecture that it is also possible to extend proofs of unsolvability for certain heuristics to take such dead pairs into account as in *constrained abstraction* (Haslum, Bonet, and Geffner 2005).

Experiments

To test our proof system, we implemented a stand-alone proof verifier and added generation of proofs to the implementations of A* search, linear $h^{\text{M\&S}}$, $h^{\max}$ and h^2 in Fast Downward (Helmert 2006) and to the h^C-based clause-learning algorithm by Steinmetz and Hoffmann (2017) for the setting without early termination, which generates a set C of conjunctions with $h^C(I) = \infty$. Both the verifier and the proof-generating planning algorithms are publicly available.[1]

We also ran all planners in baseline versions without proof generation to measure the overhead of certification. We used time limits of 30 minutes for the planner and 4 hours for the verifier and memory limits of 2 GiB for both. Where applicable, we also tested the inductive certificate approach from our previous work with the same limits. All experiments are run on the same benchmark set of unsolvable planning tasks as was used in Eriksson et al. (2017), which includes the unsolvable tasks of the Unsolvability IPC. Detailed results from these experiments and the used benchmarks are publicly available.[2].

Table 1 shows coverage results for the tested configurations, and Figure 1 shows some details on (1) planner runtime with and without proof generation, (2) verifier runtime as a function of input (proof) size; and (3) a comparison to the inductive certificate approach w.r.t. certifying planner runtime and certificate size.

Although our proof-based approach often required more memory than the inductive certificate approach when generating proofs[3], it succeeded within the given limits in more cases. For the planning algorithms not covered by inductive certificates, generation succeeded in almost all cases. For all configurations almost all created certificates could be verified within the limits.

Aside from one task for DFS-CL verification failed exclusively due to the time limit. The relatively low memory usage is due to the fact that we discard formulas when we do not need them anymore and highlights an advantage of the composability of our approach.

The detailed results in Figure 1 (bottom) emphasize the favorable comparison to our previous approach. For $h^{\max}$ the proof-based certifying planner is an order of magnitude faster on more challenging tasks, while for $h^{\text{M\&S}}$ the two are comparable. In terms of certificate size, there is little difference for $h^{\max}$, but the proof-based $h^{\text{M\&S}}$ certificates are often much smaller than the inductive ones. In one extreme case, the size difference is 60 KiB vs. 19.2 GiB.

One interesting outcome of the experiment was that initially, certificate validation for the h^C-based clause learning approach showed that in two domains the generated proofs

[1] https://doi.org/10.5281/zenodo.1196473

[2] https://doi.org/10.5281/zenodo.1196476

[3] We suspect that building a BDD of all expanded states is the main reason for the higher memory consumption.

	FD-$h^{\max}$			FD-$h^{\text{M\&S}}$			FD-h^2			FD-max($h^{\text{M\&S}}$, h^m)			DFS-CL		
	base	cert.	ver.	base	cert.	ver.	base	cert.	ver.	base	cert.	ver.	base	cert.	ver.
3unsat (30)	15	10 (10)	10 (10)	15	10 (10)	10 (10)	5	5	5	5	5	5	5	5	5
bag-barman (20)	8	8 (8)	8 (4)	8	8 (8)	8 (4)	0	0	0	0	0	0	0	0	0
bag-gripper (25)	2	2 (2)	1 (1)	2	2 (2)	1 (1)	0	0	0	0	0	0	0	0	0
bag-transport (29)	6	6 (6)	6 (4)	7	6 (6)	6 (4)	15	15	15	8	8	8	2	2	2
bottleneck (25)	20	17 (15)	17 (15)	10	9 (8)	9 (7)	11	11	11	11	11	11	9	9	9
cave-diving (25)	7	6 (5)	6 (5)	7	7 (7)	7 (6)	2	2	2	2	2	2	6	6	6
chessboard-pebbling (23)	5	4 (3)	4 (3)	5	5 (5)	5 (4)	2	2	2	2	2	2	2	2	2
diagnosis (13)	5	5 (4)	5 (4)	4	1 (1)	1 (1)	2	2	2	2	2	2	8	8	7
document-transfer (20)	7	6 (5)	6 (5)	12	10 (10)	5 (5)	2	1	1	8	7	3	5	5	4
mystery (9)	2	1 (0)	1 (0)	2	1 (2)	1 (1)	8	8	8	8	8	8	7	7	7
nomystery (150+24)	54	33 (20)	33 (17)	54	41 (47)	40 (33)	44	40	40	52	48	48	137	137	137
over-rovers (150+20)	14	11 (8)	11 (8)	25	24 (24)	24 (20)	66	66	66	70	70	70	154	155	155
over-tpp (25+30)	22	15 (10)	15 (10)	35	23 (26)	23 (24)	9	8	8	14	14	14	32	32	31
pegsol (24)	24	24 (20)	24 (20)	24	24 (24)	24 (24)	14	14	14	14	14	14	14	14	14
pegsol-row5 (15)	5	5 (4)	5 (4)	5	5 (5)	5 (4)	3	3	3	3	3	3	4	4	4
sliding-tiles (20)	10	10 (10)	10 (10)	10	10 (10)	10 (10)	0	0	0	0	0	0	0	0	0
tetris (20)	5	5 (5)	5 (5)	5	5 (5)	5 (5)	0	0	0	5	5	5	0	0	0
total (697)	211	168 (135)	167 (125)	230	191 (200)	184 (163)	183	177	177	204	199	195	385	386	383

Table 1: Completed tasks by domain: *base* is the base planner, *cert.* the certifying planner with proof generation and *ver.* the verifier. For $h^{\max}$ and $h^{\text{M\&S}}$, the numbers in parentheses are for the inductive certificates of Eriksson et al. (2017).

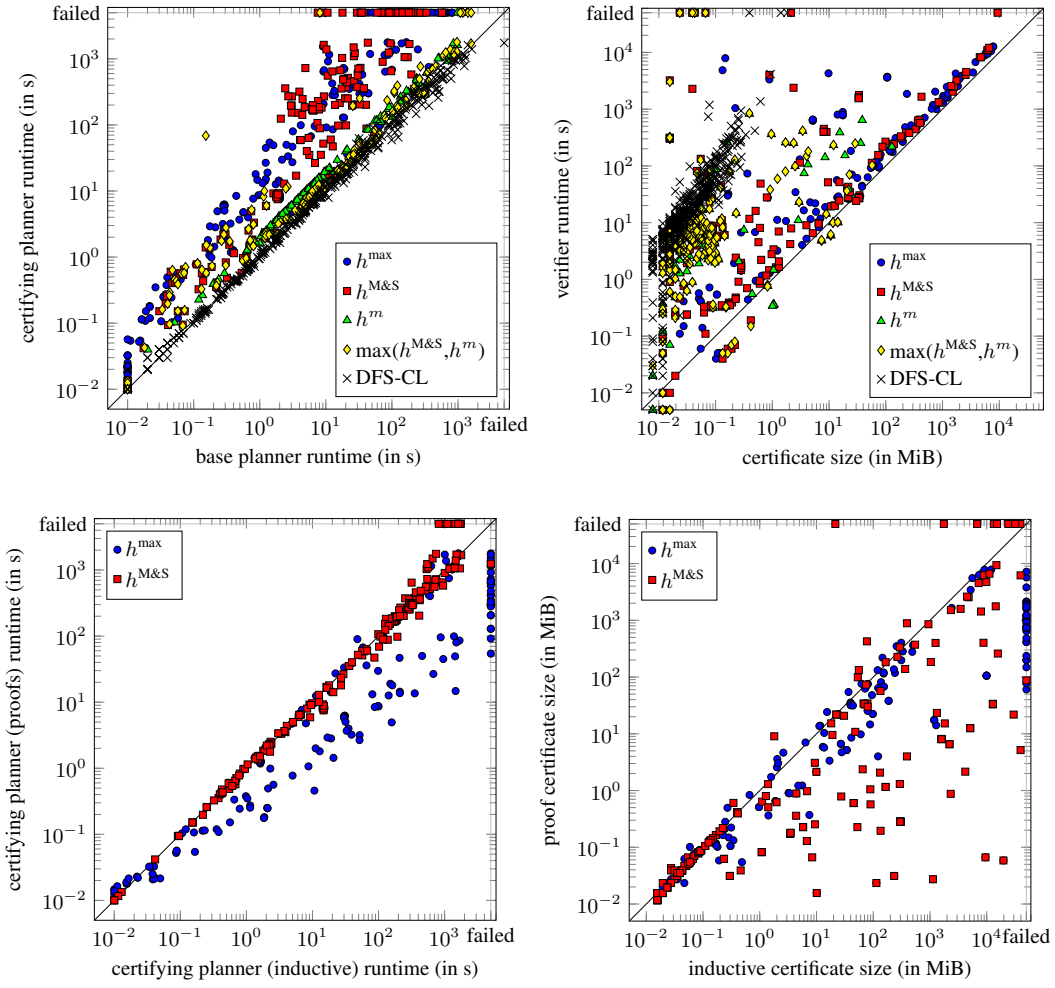

Figure 1: Detailed results. Top: runtime overhead of proof generation (left) and runtime of verifier (right). Bottom: comparison of proof certificates vs. inductive certificate regarding certifying planner runtime (left) and certificate size (right).

were invalid. On closer inspection, it turned out that the implementation of the algorithm does not actually compute h^C as defined in the paper by Steinmetz and Hoffmann (2017) but rather a stronger variant of the heuristic augmented by mutexes. Adding the mutex pairs to the set of conjunctions C addressed this issue, and the results shown here are for these proofs. We report this anecdote here because we believe it highlights the perils of accepting a claim of unsolvability at face value and showcases the usefulness of independent validation.

Conclusion

We introduced a proof system for unsolvable planning tasks that can efficiently capture the arguments for unsolvability implicit in a variety of planning techniques. We also implemented a verifier for proofs expressed in our proof systems and developed certifying variants of several planning algorithm implementations based on Fast Downward and the clause-learning approach of Steinmetz and Hoffmann.

Compared to the inductive certificates suggested in our earlier work, the main advantage of the proof system is that it is compositional and can hence easily integrate information from different sources.

We do not believe that the proof system is the finished product: in future work, we would like to extend its applicability to further planning techniques, and this will likely require extending it by further representation formalisms, basic statements, or derivation rules. This notwithstanding, we believe that the basic approach underlying the proof system is robust enough to be applied much more widely.

Acknowledgments

This work was supported by the European Research Council as part of the project "State Space Exploration: Principles, Algorithms and Applications".

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Bryant, R. E. 1985. Symbolic manipulation of Boolean functions using a graphical representation. In Ofek, H., and O'Neill, L. A., eds., *Proceedings of the 22nd ACM/IEEE Conference on Design Automation (DAC 1985)*, 688–694.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Dechter, R., and Itai, A. 1992. Finding all solutions if you can find one. In *AAAI 1992 Workshop on Tractable Reasoning*, 35–39.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability certificates for classical planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 140–149. AAAI Press.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.

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.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

Muise, C., and Lipovetzky, N., eds. 2016. *Unsolvability International Planning Competition: Planner Abstracts*.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, 568–572.

Seipp, J.; Pommerening, F.; Sievers, S.; Wehrle, M.; Fawcett, C.; and Alkhazraji, Y. 2016. Fast Downward Aidos. In *Unsolvability International Planning Competition: planner abstracts*, 28–38.

Steinmetz, M., and Hoffmann, J. 2016. Towards clause-learning state space search: Learning to recognize dead-ends. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 760–768. AAAI Press.

Steinmetz, M., and Hoffmann, J. 2017. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *Artificial Intelligence* 245:1–37.