Certified Unsolvability for SAT Planning with Property Directed Reachability

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

Abstract

While classical planning systems can usually detect if a task is unsolvable, only recent research introduced a way to verify such a claim. These methods have already been applied to a variety of explicit and symbolic search algorithms, but so far no planning technique based on SAT has been covered with them. We fill this gap by showing how *property directed reachability* can produce proofs while only minimally altering the framework by allowing to utilize certificates for unsolvable SAT queries within the proof. We additionally show that a variant of the algorithm that does not use SAT calls can produce proofs that fit into the existing framework without requiring any changes.

Introduction

Classical planning has recently seen an increased interest in detecting the unsolvability of planning tasks, as shown for example by the emergence of the first Unsolvability IPC (Muise and Lipovetzky 2016), a planning competition aimed specifically at the problem of proving the absence of solutions for a classical planning task. One challenge in the practical application of algorithms that prove unsolvability is that there is no easily available way to verify that they function correctly. When a planning algorithm claims to have found a solution, plan validation tools such as VAL (Howey and Long 2003) and INVAL (Haslum 2017) can independently verify the correctness of this claim. Claims of unsolvability traditionally had to be taken at face value, which severely reduces their trustworthiness.

The propositional satisfiability (SAT) community faced a similar problem: in the satisfying case, it is straightforward to verify that a given assignment satisfies a formula, but verifying unsatisfiability is less direct. As a result, *certificates of unsolvability* in a standard format were defined, which can be independently validated to verify that the SAT solver has indeed correctly established the unsolvability of the given input formula, and the recurring SAT competitions have for some time required participating SAT solvers to produce them in the case of unsolvability.

We recently also proposed such certificates of unsolvability for classical planning (Eriksson, Röger, and Helmert 2017; 2018; Eriksson 2019a). In Eriksson et al. (2017) we argue that a practically useful form of certificates of unsolvability should meet three criteria. Firstly, they should be *efficiently generatable*, i.e., it should be possible to modify a given non-certifying planning algorithm to emit certificates of unsolvability with only moderate extra effort (ideally linear and certainly polynomial in the runtime of the non-certifying algorithm). Secondly, they should be *efficiently verifiable*: an independent certificate verifier should run in polynomial time in the size of its input (planning task description and certificate). Thirdly, they should be *general*: a single form of certificates should cover a large range of planning algorithms so that certificate verifiers can be truly independent of planning algorithms and it is possible to build trust in the implementation of a verifier.

In Eriksson et al. (2018) we describe a form of certificates based on unsolvability proofs that consist of *basic statements* about compactly represented sets of states that can be verified in some tractable fragment of logic or other knowledge compilation framework (Darwiche and Marquis 2002), along with *derivation steps* in a proof calculus that can be mechanically validated by matching them to a set of permissible inference rules. They also provide an implementation of a certificate validator and show that the proof system satisfies the above-mentioned efficiency criteria for a range of explicit and symbolic search techniques, heuristics based on abstraction and delete relaxation, the clause learning state-space search algorithm of Steinmetz and Hoffmann (2016), forward and backward mutexes (Alcázar and Torralba 2015), and the Trapper algorithm (Lipovetzky, Muise, and Geffner 2016). An extended version of this proof system is described in Eriksson (2019a).

A noticeable gap in this list of covered techniques are SAT-based planning algorithms. We see two reasons why this gap exists. Firstly, it appears challenging to integrate SAT-based techniques into the existing framework of polynomial-time certificate verification given that the SAT problem itself is **NP**-complete.

Secondly, traditional SAT-based planning systems (e.g., Kautz and Selman 1992) iteratively build formulas encoding whether plans with a horizon of $n = 0, 1, \dots$ exist and can only terminate with a claim of unsolvability when the horizon reaches some upper bound on the diameter of the search space. Most existing SAT planners use no bounds or only trivial upper bounds (the total number of states minus 1), which can only be reached for very simple planning tasks and only in cases where blind explicit-state search would be vastly more efficient. This has recently changed somewhat with the introduction of nontrivial upper bounds (e.g., Abdulaziz, Gretton, and Norrish 2017), but the number of scenarios in which traditional SAT planning using these techniques is competitive with other approaches for proving unsolvability is still quite limited.

Therefore, our paper focuses on the *Property Directed Reachability* (PDR) algorithm family, which uses SAT queries as a way to infer reachability information while simultaneously performing a form of explicit search (Suda 2014). While clearly owing to the heritage of SAT-based planning algorithms, Property Directed Reachability does not fit in the mold of traditional SAT planners and does not share their weakness regarding detecting unsolvability.

We show that for planning tasks in the propositional STRIPS formalism, we can efficiently generate and verify proofs for Suda's PDR algorithm in the proof system described in Eriksson (2019a) with no modifications to the proof system. However, Suda also describes more general versions of the PDR algorithm for a variant of propositional STRIPS that allows negative preconditions and goals, which is not covered by the existing proof system. Here, we show that the proof system can be extended in a modular way to cover this extended formalism and the more general PDR algorithm. In particular, the extended proof system allows incorporating UNSAT certificates within basic statements in such a way that they can be transparently handled by an UNSAT validator while satisfying the properties of polynomial overhead for generating the certificate of unsolvability and polynomial verifiability of the overall certificate.

Background

We consider classical planning tasks given in propositional STRIPS. A STRIPS planning task $\Pi = \langle \mathbf{V}, \mathbf{A}, \mathbf{I}, \mathbf{G} \rangle$ consists of a set of propositional variables $\mathbf{V}$, a set of actions $\mathbf{A}$, the initial state $\mathbf{I} \subseteq \mathbf{V}$ and a goal description $\mathbf{G} \subseteq \mathbf{V}$. A state of Π is given as a set of variables $s \subseteq \mathbf{V}$ and denotes that in s, propositional variables $v \in s$ are set to true. A state s is a goal state iff $\mathbf{G} \subseteq s$. An action $a \in \mathbf{A}$ consists of three sets $\mathit{pre}(a), \mathit{add}(a), \mathit{del}(a) \subseteq \mathbf{V}$. Action a is applicable in s if $\mathit{pre}(a) \subseteq s$, and for applicable actions the successor state is $s[a] = (s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.

For a sequence of actions $\pi = \langle a_1, \dots, a_n \rangle$, we define $s[\pi] = ((s[a_1]) \dots)[a_n]$ if action a_i is applicable in $s[\langle a_1, \dots, a_{i-1} \rangle]$ for all $1 \leq i \leq n$. A task Π is solvable iff there exists some π such that $\mathbf{I}[\pi]$ is a goal state, in which case π is called a *plan* of Π; otherwise it is unsolvable.

The set of all states of Π is denoted by $\mathbf{S}$, and the set of goal states by $\mathbf{S}_G$. For state set $S \subseteq \mathbf{S}$ and action set $A \subseteq \mathbf{A}$, the set $S[A] = \{s[a] \mid s \in S, a \in A, a \text{ applicable in } s\}$ denotes the set of all successors of S with respect to A, and similarly $[A]S = \{s' \mid a \in A, a \text{ applicable in } s', s'[a] \in S\}$ denotes the set of all predecessors of S with respect to A.

A set of states can be represented by a propositional formula over variables $\mathbf{V}$ by mapping each model of the formula to a state containing the variables that are set to true in the model. More formally we define $\mathit{states}(\varphi) = \{s_\mathcal{I} \mid \mathcal{I} : \mathbf{V} \mapsto \{\top, \bot\}, \mathcal{I} \models \varphi\}$ where $s_\mathcal{I} = \{v \mid \mathcal{I}(v) = \top\}$. Propositional formulas can be represented in a multitude of formalisms $\mathbf{R}$, such as BDDs, Horn formulas or CNF formulas. If we want to specify that a formula is represented with formalism $\mathbf{R}$, we denote the formula as $\varphi_\mathbf{R}$.

Unsolvability Proof System

The aim of certifying an algorithm is to provide a line of reasoning that explains how the algorithm arrived at its output and can be verified by an independent verifier. In order to ensure feasible generation and verification, the certifying algorithm should be able to produce a certificate with no more than polynomial overhead in its (non-certifying) runtime, and a certificate should be verifiable in time polynomial in its size.

In this paper we focus on a proof system based certificate first introduced in Eriksson, Röger, and Helmert (2018) and further extended in Eriksson (2019a). This type of certificate is built upon sets of states represented in possibly multiple formalisms. These sets are then used to first establish an initial knowledge base of so-called *basic statements* which take the form of subset relationships. Since the proof simply assumes these statements as facts, an independent verifier must verify whether the subset relations actually hold. In a second step the basic statements are combined with the help of predefined *inference rules*, which are general rules phrased in the style of natural deduction, i.e. if a set of premises is present in the knowledge base, then the conclusion of the rule can be added to the knowledge base as well. The correctness of these rules is already established in the definition of the proof system, thus a verifier must only check that the rule is applied correctly, but can do so on a purely syntactical level without needing to consider the semantics of the premises or conclusion.

On a high level, a proof shows unsolvability by iteratively deducing that certain state sets are *dead*, meaning that no plan can go through any state from the set. A central notion for deducing deadness is the idea of an *inductive set*: a state set S such that all its successors are contained in S (i.e. $S[\mathbf{A}] \subseteq S$). If an inductive set contains no goal state, then S must be dead, because we cannot reach any goal state from it. This concept can also be applied in a regression setting, and if we already have knowledge about certain state sets being dead we can extend the argument for sets with $S[\mathbf{A}] \subseteq S \cup S'$, where S' is a dead set. Unsolvability can then be shown when either the initial state or all goal states have been deduced to be dead.

For example, a proof for an A^* search with some heuristic could first deduce that each dead-end is dead and use this knowledge to state that the progression of the set of expanded states leads only to expanded states or to dead states, thus the set of expanded states must also be dead. Since the initial state is contained in this set, it must therefore be dead and the task must be unsolvable.

Since rule applications are purely syntactical, the only critical part of ensuring that a proof can be verified in time polynomial in its size is verifying the basic statements occurring in the proof, and whether this is possible depends on the $\mathbf{R}$-formalisms representing the involved sets. The proof system restricts the allowed statements to special cases of subset statements over set variables $X_{\mathbf{R}} = \textit{states}(\varphi_{\mathbf{R}})$, set literals $L_{\mathbf{R}} \in \{X_{\mathbf{R}}, \overline{X_{\mathbf{R}}}\}$ and action sets $A \subseteq \mathbf{A}$:

- **B1**: $\bigcap L_{\mathbf{R}} \subseteq \bigcup L'_{\mathbf{R}}$
- **B2**: $(\bigcap X_{\mathbf{R}})[A] \cap \bigcap L_{\mathbf{R}} \subseteq \bigcup L'_{\mathbf{R}}$
- **B3**: $[A](\bigcap X_{\mathbf{R}}) \cap \bigcap L_{\mathbf{R}} \subseteq \bigcup L'_{\mathbf{R}}$
- **B4**: $L_{\mathbf{R}} \subseteq L'_{\mathbf{R}'}$
- **B5**: $A \subseteq A'$

Each basic statement requires certain logical operations of the involved formalisms in order to guarantee efficient verification. The logical operations relevant for this paper are the following:

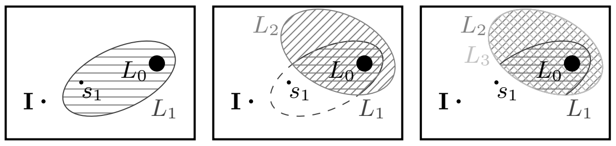
- **CO** (consistency): given $\mathbf{R}$-formula $\varphi_{\mathbf{R}}$, test whether $\varphi_{\mathbf{R}}$ is satisfiable.
- **SE** (sentential entailment): given $\mathbf{R}$-formulas $\varphi_{\mathbf{R}}$ and $\psi_{\mathbf{R}}$, test whether $\varphi_{\mathbf{R}} \models \psi_{\mathbf{R}}$.
- $\wedge$**BC** (bounded conjunction): given $\mathbf{R}$-formulas $\varphi_{\mathbf{R}}$ and $\psi_{\mathbf{R}}$, construct an $\mathbf{R}$-formula representing $\varphi_{\mathbf{R}} \wedge \psi_{\mathbf{R}}$.
- $\wedge$**C** (general conjunction): given $\mathbf{R}$-formulas $\varphi^1_{\mathbf{R}}$ to $\varphi^n_{\mathbf{R}}$, construct an $\mathbf{R}$-formula representing $\varphi^1_{\mathbf{R}} \wedge \cdots \wedge \varphi^n_{\mathbf{R}}$.
- **CL** (conjunction of literals): given a conjunction of literals φ, construct an equivalent $\mathbf{R}$-formula $\varphi_{\mathbf{R}}$.
- **RN**$_\prec$: Given $\mathbf{R}$-formula $\varphi_{\mathbf{R}}$, a variable order $\prec$ and an injective variable renaming r following $\prec$ in the sense that if $v_1 \prec v_2$ then $r(v_1) \prec r(v_2)$, construct an $\mathbf{R}$-formula $\varphi[r]_{\mathbf{R}}$, i.e. φ with each variable v replaced by $r(v)$.

Furthermore, we want to highlight the following inference rules, which we will utilize when building proofs for PDR:

- **ED** (empty set dead): With no premises, $\emptyset$ is dead.
- **SD** (subset dead): If S' is dead and $S \subseteq S'$, then S is dead.
- **RI** (regression inductivity without $\mathbf{I}$): If $[\mathbf{A}]S \subseteq S \cup S'$, S' is dead and $\{\mathbf{I}\} \subseteq \overline{S}$, then S is dead.
- **CG** (conclusion with dead goal): If $\mathbf{S}_G$ is dead, then the task is unsolvable.
- **UR** (union introduction on right-hand side): With no premises, $S \subseteq S \cup S'$.
- **SI** (subset intersection): If $S \subseteq S'$ and $S \subseteq S''$, then $S \subseteq (S' \cap S'')$.
- **ST** (subset transitivity): If $S \subseteq S'$ and $S' \subseteq S''$, then $S \subseteq S''$.

Property Directed Reachability

Property directed reachability (PDR) was originally proposed by Bradley (2011) and refined by Eén, Mishchenko, and Brayton (2011) as a way to analyze reachability in symbolic transition systems, and was adopted to planning by Suda (2014). It iteratively tries to find solutions of length $n = 0, 1, \ldots$ while maintaining a family of sets of states called *layers*, which serve as an overapproximation of the states within a certain distance of any goal state. In iteration n, the algorithm tries to find a path of length n such that $\mathbf{I}$ satisfies layer n, the successor of $\mathbf{I}$ satisfies layer $n-1$, and so forth. Failures in finding such a path are used as a way to *refine* the layers, thus speeding up future iterations. The layers are represented as CNF formulas Λ_i, and we denote the set of states encoded by Λ_i as L_i.



(a) after iteration 1 (b) after iteration 2 (c) after iteration 3

Figure 1: The search space during different phases of PDR on the forklift example.

PDR searches in a space of so-called *obligations* (s, i), which represent that state s satisfies Λ_i. The algorithm first initializes Λ_0 to $\bigwedge_{g \in G} g$ and all other layers to $\top$. In each iteration, the search is initialized with obligation (I, n). Expanding, or *extending* an obligation (s, i) means trying to find a successor $s' = s[a]$ for some $a \in \mathbf{A}$ which satisfies Λ_{i-1}. When PDR cannot extend an obligation (s, i), it extracts a *reason* r: a conjunction of literals satisfied by s such that no state satisfying r can have a successor satisfying Λ_{i-1}. Since this means that only states *not* satisfying r can be within i steps of a goal, the clause $\neg r$ can be added to all Λ_j with $j \leq i$. This *strengthens* the layers, avoiding that the same obligation will be recreated in future iterations, and potentially also avoiding similar unextendable obligations. At the end of each iteration n, PDR tries to additionally strengthen the layers further by *pushing clauses*: for each clause γ occurring in Λ_{i-1} but not in Λ_i, γ can be added to Λ_i if the regression of L_{i-1} logically entails γ. PDR terminates if it either reaches an obligation of the form $(s, 0)$, in which case a plan has been found, or if at the end of iteration n two subsequent layers are identical, which implies that the task is unsolvable.

As an example, consider a task where we want to lift a crate with a forklift, but the forklift is broken. We describe the position of the crate with three variables `c-on-g` (ground), `c-on-f` (forklift), `c-on-t` (top) and the state of the forklift with two variables `raised` and `lowered`. Initially, the crate is on the ground and the forklift lowered ($\mathbf{I} = \{$`c-on-g`, `lowered`$\}$). Our goal is for the crate to be on top ($\mathbf{G} =$ `c-on-t`). PDR initializes $\Lambda_0 =$ `c-on-t` and terminates iteration 0 immediately since $\mathbf{I}$ does not satisfy Λ_0. Iteration 1 starts with obligation $(\mathbf{I}, 1)$, but fails to extend it as we can only reach Λ_0 if the crate is already on top or if we unload it there, but for the latter the crate would need to be on the forklift. Thus $(\neg$`c-on-t` $\wedge \neg$`c-on-f`$)$ is a possible reason for this failed extension, and its negation is added to Λ_1 and Λ_0. Having explored all obligations, the iteration finishes with a clause pushing phase that results in no changes. Figure 1a depicts the situation at this point. Note that L_0 always represents $\mathbf{S}_G$, since any added clause is already implied by $\mathbf{G}$ with which Λ_0 was initialized.

In the second iteration we start with $(\mathbf{I}, 2)$. Loading the crate on the forklift from $\mathbf{I}$ yields $s_1 = \{\texttt{c-on-f}\}$, which satisfies $\Lambda_1 (= \texttt{c-on-t} \vee \texttt{c-on-f})$ and thus results in a new obligation $(s_1, 1)$. Extending this obligation fails, however, since we cannot unload the crate on top unless the forklift is raised. We can extract the reason $(\neg\texttt{c-on-t} \wedge \neg\texttt{raised})$, whose negation is added to Λ_1 and Λ_0. Extending $(\mathbf{I}, 2)$ with a different successor fails as well since no successor satisfies $(\texttt{c-on-t} \vee \texttt{raised})$ (which is now a clause in Λ_1). Since the forklift can only be raised if it has been all along, we can again use the reason $(\neg\texttt{c-on-t} \wedge \neg\texttt{raised})$ to strengthen Λ_2. With no obligations left we move to clause pushing, again resulting in no changes. Figure 1b shows the changes to the layers: L_1 has become smaller due to the added clause, and we also see that $L_2 \supseteq L_1$ since all clauses in Λ_2 are also present in Λ_1. In iteration 3, obligation $(\mathbf{I}, 3)$ cannot be extended since no successor satisfies $\Lambda_2 = (\texttt{c-on-t} \vee \texttt{raised})$. At this point Λ_2 and Λ_3 are equal, meaning the algorithm terminates claiming the problem to be unsolvable. Figure 1c depicts this final state.

The original PDR algorithm uses SAT solvers in various places, such as finding clauses for failed obligation extensions and to check whether a clause can be pushed in a subsequent layer. Suda (2014) proposed two variants when adopting the algorithm for planning: one which is also based on SAT solvers and one which avoids deciding SAT by exploiting planning-specific properties of the symbolic transition system.

To conclude this section we highlight several important theoretical properties of PDR which will be useful when building certificates:

The way clauses get added to layers ensures that L_j is always an overapproximation of the regression of L_{j-1}. Furthermore, if a clause is present in Λ_i, it must also be present in all Λ_j with $j < i$: when adding a clause due to a failed obligation extension (s, i), the clause is added to all layers up to Λ_i, and when pushing a clause to Λ_i the clause was already present in Λ_{i-1}. This also means that $L_i \supseteq L_j$ holds for all $j < i$.

On a more formal level, PDR guarantees the following when adding a clause γ to Λ_j:

- $[\mathbf{A}]L_{i-1} \subseteq \mathit{states}(\gamma)$ holds for some $i \geq j$
- $\mathbf{S}_G \subseteq \mathit{states}(\gamma)$

Furthermore, the following invariants always hold:

(i) $L_0 = \mathbf{S}_G$

(ii) $L_i \supseteq L_{i-1}$

(iii) $[\mathbf{A}]L_{i-1} \subseteq L_i$

(iv) At the end of iteration n, $\mathbf{I} \notin L_n$.

Certificates for PDR

PDR only claims unsolvability if at the end of an iteration two subsequent layers Λ_x and Λ_{x-1} are equal. Intuitively this shows unsolvability because the algorithm has established that we cannot reach more states with x backwards steps from the goal than with $x - 1$ steps and that the initial state is not among the backwards-reachable states.

In what follows, we will denote the "final" state of layers Λ_x and Λ_{x-1} when unsolvability is detected (at which point they are equal) by Λ_{U}, and the corresponding state set by L_{U}. Since clauses only get added to layers during execution, $L_{\mathrm{U}} \subseteq L_x$ and $L_{\mathrm{U}} \subseteq L_{x-1}$ holds at any point of the algorithm. We will also use the shorthand $\gamma \in \Lambda_i$ to denote that clause γ occurs in Λ_i and will define $|\Lambda_i|$ as the number of clauses in Λ_i.

The proof we aim to build argues in the same fashion as the intuitive explanation above, i.e. it shows that L_{U} contains all goal states, does not contain the initial state, and that its regression does not lead to new states, or more formally $[\mathbf{A}]L_{\mathrm{U}} \subseteq L_{\mathrm{U}}$. However, our proof will contain more fine-grained statements that better fit the actual reasoning done in PDR, and then utilize the composability of the proof system to combine these statements into an argument for unsolvability.

Our proof is based on statements that we derive from the properties of PDR outlined at the end of the previous section. Firstly, we stated that when a clause γ is added to Λ_j, then $[\mathbf{A}]L_{i-1} \subseteq \mathit{states}(\gamma)$ for some $i \geq j$. This specifically also holds when a clause is added to Λ_x. We know that $L_{\mathrm{U}} \subseteq L_{x-1} \subseteq L_{i-1}$ holds: we established the left-hand side in the beginning of this section, and the right-hand side follows from $i \geq x$ and iteratively applying invariant (ii). Furthermore, regression is monotonic in the sense that if $S \subseteq S'$ then $[A]S \subseteq [A]S'$. We can thus state $[\mathbf{A}]L_{\mathrm{U}} \subseteq \mathit{states}(\gamma_i)$ for all $\gamma_i \in L_{\mathrm{U}}$. Next, we need the statement $\mathbf{S}_G \subseteq \mathit{states}(\gamma_i)$ for all $\gamma_i \in \Lambda_{\mathrm{U}}$, which is an explicitly stated property of PDR. Finally, we need $\{\mathbf{I}\} \subseteq \overline{L_{\mathrm{U}}}$, which follows directly from invariant (iv).

Lemma 1. *PDR deduces that* (P_i) $[\mathbf{A}]L_{\mathrm{U}} \subseteq \mathit{states}(\gamma_i)$ *and* (Q_i) $\mathbf{S}_G \subseteq \mathit{states}(\gamma_i)$ *holds for all clauses* $\gamma_i \in \Lambda_{\mathrm{U}}$*, and that* ($\mathrm{R}_1$) $\{\mathbf{I}\} \subseteq \overline{L_{\mathrm{U}}}$ *holds.*

Utilizing the arguments from Lemma 1 as basic statements, a proof in the proof system can now be built by combining the statements of type (P_i) to the form $[\mathbf{A}]L_{\mathrm{U}} \subseteq L_{\mathrm{U}}$, the statements of type (Q_i) to the form $\mathbf{S}_G \subseteq L_{\mathrm{U}}$, and combining it with statement (R_1):

Theorem 1. *If PDR can efficiently represent all statements from Lemma 1 in some* $\mathbf{R}$*-formalism that efficiently supports* ***CL***, ***∧C***, ***SE***, ***RN***$_{\preceq}$ *and* ***CO****, we can build a suitable unsolvability proof.*

Proof. The structure of the proof can be seen in Table 1. It starts by stating with basic statement **B3** for each clause $\gamma_i \in \Lambda_{\mathrm{U}}$ that the regression of L_{U} is contained in the state set represented by γ_i. With the subset intersection rule (**SI**) we can successively combine these statements to conclude that L_{U} is backwards-inductive (i.e., the regression of L_{U} is contained in L_{U}). The next two steps merely reformulate this statement with rules "union introduction right" (**UR**) and "subset transitivity" (**ST**) into a form that is later needed as a premise for deriving that L_{U} is dead.

In the second part, basic statement **B1** says that the goal is included in each state set represented by a clause $\gamma_i \in \Lambda_{\mathrm{U}}$, which we can combine with subset intersection to conclude that the goal is included in L_{U}.

ID	statements	rule	premises
(P_i)	$[\mathbf{A}]L_{\mathrm{U}} \subseteq \mathit{states}(\gamma_i)$	**B3**	
(P'_i)	$[\mathbf{A}]L_{\mathrm{U}} \subseteq \bigcap_{j \leq i} \mathit{states}(\gamma_j)$	**SI**	(P_i) (P'_{i-1})
(P_x)	$[\mathbf{A}]L_{\mathrm{U}} \subseteq L_{\mathrm{U}}$	**SI**	($P_{\lvert\Lambda_{\mathrm{U}}\rvert}$) ($P'_{\lvert\Lambda_{\mathrm{U}}\rvert-1}$)
(P_y)	$L_{\mathrm{U}} \subseteq L_{\mathrm{U}} \cup \emptyset$	**UR**	
(P)	$[\mathbf{A}]L_{\mathrm{U}} \subseteq L_{\mathrm{U}} \cup \emptyset$	**ST**	(P_x) (P_y)
(Q_i)	$\mathbf{S}_G \subseteq \mathit{states}(\gamma_i)$	**B1**	
(Q'_i)	$\mathbf{S}_G \subseteq \bigcap_{j \leq i} \mathit{states}(\gamma_j)$	**SI**	(Q_i) (Q'_{i-1})
(Q)	$\mathbf{S}_G \subseteq L_{\mathrm{U}}$	**SI**	($Q_{\lvert\Lambda_{\mathrm{U}}\rvert}$) ($Q'_{\lvert\Lambda_{\mathrm{U}}\rvert-1}$)
(R_1)	$\{\mathbf{I}\} \subseteq \overline{L_{\mathrm{U}}}$	**B1**	
(R_2)	$\emptyset$ dead	**ED**	
(R_3)	L_{U} dead	**RI**	(P) (R_2) (R_1)
(R_4)	$\mathbf{S}_G$ dead	**SD**	(R_3) (Q)
(R_5)	unsolvable	**CG**	(R_4)

Table 1: General proof structure. Each statement is associated with an ID, the rule or basic statement that derived the statements, and in case of rules the IDs of the required premises.

Lastly, we state with **B1** that the initial state is not included in L_{U}. Deriving with rule **ED** that $\emptyset$ is dead, we can now say that L_{U} is dead, since it can only reach itself or a dead set (in this case the empty set) with one regression step and does not contain the initial state. Furthermore, since the set of all goal states is contained in L_{U} we can deduce that all goal states must be dead as well, and thus conclude that the task is unsolvable.

The proof utilizes only the arguments from Lemma 1 as basic statements and assumes that each clause $\gamma_i \in \Lambda_{\mathrm{U}}$ is encoded with the same **R**-formalism, with L_{U} represented as an intersection of all state sets encoded by the clauses. In order to be able to represent these sets, **R** only needs to efficiently support **CL**. According to Theorems 5.5 and 5.6 from Eriksson (2019a), basic statements (P_i) can be verified if **R** efficiently supports **CL**, $\wedge$**C**, **SE** and **RN**$_{\prec}$. For (Q_i) we require **SE**, and for (R_1) **CO** and $\wedge$**C** is sufficient. $\square$

When looking at the variant of PDR that does not utilize SAT solvers, Suda (2014) remarks that for STRIPS tasks, the layers consist of clauses with only positive literals, a special case of *dual-Horn* formulas. Since dual-Horn is one of the maximal tractable classes in Schaefer's (1978) dichotomy, it supports all required operations efficiently and is thus a suitable formalism for the proof in Theorem 1, meaning we can build proofs for this variant of PDR within the existing framework. Moreover, we can actually formalize a much more succinct proof directly stating (P_x) and (Q) as basic statements with a single dual-Horn formula representing L_{U}.

Furthermore, the variant of PDR that utilizes SAT solvers will also result in purely positive (and therefore dual-Horn) clauses due to the "reason minimization" technique employed by PDR and the monotonicity of STRIPS tasks. Intuitively speaking, the monotonicity property for STRIPS tasks states that it is always beneficial to have task variables set to true: the goal is a set of positive literals, so when s is a goal state, dominating states (with a superset of true state variables) are also goal states, and this dominance property also holds for action preconditions and is preserved by action application. The key step in PDR that is responsible for deriving new clauses is to find a *reason* (a conjunction of literals over state variables) that explains why a given subgoal (as defined by the layers of the algorithm) cannot be reached. The monotonicity of STRIPS implies that a minimal reason is always a conjunction of negative literals, and hence the clause added to the algorithm (which is the negation of the generated reason) is always purely positive.

Corollary 1. *For regular STRIPS tasks, a proof for PDR both utilizing and not utilizing SAT can be generated and verified efficiently by representing L_{U} as a dual-Horn formula.*

STRIPS with Negation

In order to fully cover the PDR algorithm for classical planning, we must extend the planning formalism we consider to *propositional STRIPS with negation* (PSN) (e.g., Brafman and Domshlak 2003), which is the most general planning formalism considered by Suda (2014). Unlike regular STRIPS, the goal description and action preconditions in PSN tasks may contain negative literals. In this formalism, the layers in PDR are no longer guaranteed to consist of dual-Horn clauses.

Definition 1. *A PSN task $\Pi = \langle \mathbf{V}, \mathbf{A}, \mathbf{I}, \mathbf{G} \rangle$ consists of a set of propositional variables $\mathbf{V}$, a set of actions $\mathbf{A}$, where an action a is defined by four sets of variables $\mathit{pre}^+(a), \mathit{pre}^-(a), \mathit{add}(a), \mathit{del}(a) \subseteq \mathbf{V}$, initial state $\mathbf{I} \subseteq \mathbf{V}$ denoting the variables that are true initially, and a goal description $\mathbf{G} = \langle \mathbf{G}^+, \mathbf{G}^- \rangle$, where $\mathbf{G}^+$ denotes the variables that must be true in a goal state and $\mathbf{G}^-$ the variables that must be false, i.e. $\mathbf{G}^+ \subseteq s_G$ and $\mathbf{G}^- \cap s_G = \emptyset$ for all goal states $s_G \in \mathbf{S}_G$. Action a is applicable in a state s iff $\mathit{pre}^+(a) \subseteq s$ and $\mathit{pre}^-(a) \cap s = \emptyset$, and applying a in s results in $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.*

Before we investigate how we can build proofs for PDR in this setting, we need to make sure that the proof system framework, which was only defined for regular STRIPS tasks, can also be used for PSN tasks. Fortunately, this is the case without needing to alter the framework in any way:

Theorem 2. *The proof system framework from Eriksson (2019a) can verify proofs for tasks given in PSN.*

Proof. Since the only difference between STRIPS and PSN is the description of **G** and **A**, we check in which places these parts of the task occur in the proof framework:

The only requirement for **G** is that the set of all goal states must be representable with **CL**. Changing **G** to include negative literals does not affect this requirement and thus requires no change in the framework. Actions need to be represented when verifying basic statements **B2** and **B3**. In these statements, the formalism needs to be able to represent effects and preconditions of an action with **CL**, which is still possible with negative preconditions.

Actions also occur in basic statement **B5**, and both actions and the set of all goal states additionally occur in rule applications, but in both of these places they occur only as named variables and do not need to be interpreted semantically. $\square$

PDR with SAT employs a SAT solver to determine the clauses that are added to layers, meaning the SAT solver must guarantee that the clauses have the properties we utilize in our proof from Theorem 1. In order to verify the proof we thus in turn need to be able to verify UNSAT claims, which are in general **coNP**-complete.

SAT solvers themselves have a long history of certifying their output (e.g., van Gelder 2002; 2008; Heule, Hunt, and Wetzler 2013a; 2013b) and almost all SAT solvers are capable of producing a certificate for their unsolvability claims. With this in mind, we tackle the above issue by letting the SAT solver used in PDR build certificates for its UNSAT outputs and integrating those certificates into the proof as a help to verify the claims. Since UNSAT certificates can be generated in time polynomial in the runtime of the SAT solver and can be verified in time polynomial in the size of the certificate, we are able to stay within the desired polynomial bounds as well.

Concretely, we propose to introduce new types of basic statements into the proof system which are built on state sets represented as CNF formulas and which require one or several SAT certificates as input.

Definition 2. *Let φ_{CNF} and ψ_{CNF} be CNF formulas and $A \subseteq \mathbf{A}$.*

We extend the proof system in Eriksson (2019a) with the following basic statements, which accept additional input in the form of one or several UNSAT certificates:

C1a $\mathit{states}(\varphi_{CNF}) \subseteq \mathit{states}(\psi_{CNF})$

C1b $\mathit{states}(\varphi_{CNF}) \subseteq \overline{\mathit{states}(\psi_{CNF})}$

C2a $\mathit{states}(\varphi_{CNF})[A] \subseteq \mathit{states}(\psi_{CNF})$

C2b $\mathit{states}(\varphi_{CNF})[A] \subseteq \overline{\mathit{states}(\psi_{CNF})}$

C3a $[A]\mathit{states}(\varphi_{CNF}) \subseteq \mathit{states}(\psi_{CNF})$

C3b $[A]\mathit{states}(\varphi_{CNF}) \subseteq \overline{\mathit{states}(\psi_{CNF})}$

Since we want to verify the statements with the help of the provided UNSAT certificates, we need ensure that they actually imply the statement. For statements **C2** and **C3**, we assume that the set of successors or predecessors with respect to A is encoded with help of a *transition formula* T_A. Several encodings for such transition formulas exist (e.g., Rintanen 2012; Huang, Chen, and Zhang 2012) and they share the general idea that the formula encodes *pairs of states* where the successor state is encoded with fresh variables $\mathbf{V}' = \{v' \mid v \in \mathbf{V}\}$. More formally, we assume a CNF formula T_A over $\mathbf{V} \cup \mathbf{V}' \cup V_{\text{aux}}$ for some (possibly empty) set of auxiliary variables V_{aux}, such that the set of models $\mathcal{I}$ of T_A encode all pairs of states $\langle s, s' \rangle$ with $s = \{v \mid v \in \mathbf{V}, \mathcal{I}(v) = \top\}$, $s' = \{v \mid v \in \mathbf{V}, \mathcal{I}(v') = \top\}$ and $s[a] = s'$ for some $a \in A$. In what follows, we denote with φ' the formula that is obtained by replacing each occurrence of variable $v \in \mathbf{V}$ in φ with the corresponding $v' \in \mathbf{V}'$.

Theorem 3. *The basic statements in Definition 2 can be verified if the statement is provided with UNSAT certificates for the following formulas and can be validated in time polynomial in the size of the UNSAT certificates:*

- ***C1a***: *$\varphi_{CNF} \wedge \neg\gamma$ for each clause $\gamma \in \psi_{CNF}$*
- ***C1b***: *$\varphi_{CNF} \wedge \psi_{CNF}$*
- ***C2a***: *$\varphi_{CNF} \wedge T_A \wedge \neg\gamma'$ for each clause $\gamma \in \psi_{CNF}$*
- ***C2b***: *$\varphi_{CNF} \wedge T_A \wedge \psi'_{CNF}$*
- ***C3a***: *$\varphi'_{CNF} \wedge T_A \wedge \neg\gamma$ for each clause $\gamma \in \psi_{CNF}$*
- ***C3b***: *$\varphi'_{CNF} \wedge T_A \wedge \psi_{CNF}$*

Proof. For **C1a**, we have that $\varphi_{\mathbf{CNF}} \wedge \neg\gamma$ being unsatisfiable for all $\gamma \in \psi_{\mathbf{CNF}}$ implies that $\bigvee_{\gamma \in \psi_{\mathbf{CNF}}} (\varphi_{\mathbf{CNF}} \wedge \neg\gamma) \equiv \varphi_{\mathbf{CNF}} \wedge \neg\psi_{\mathbf{CNF}}$ is unsatisfiable, meaning $\varphi_{\mathbf{CNF}}$ must imply $\psi_{\mathbf{CNF}}$. From this we can conclude $\mathit{states}(\varphi_{\mathbf{CNF}}) \subseteq \mathit{states}(\psi_{\mathbf{CNF}})$.

For **C1b**, $\varphi_{\mathbf{CNF}} \wedge \psi_{\mathbf{CNF}}$ being unsatisfiable means that $\varphi_{\mathbf{CNF}}$ implies $\neg\psi_{\mathbf{CNF}}$. Since $\mathit{states}(\neg\psi_{\mathbf{CNF}}) = \overline{\mathit{states}(\psi_{\mathbf{CNF}})}$, we thus have that $\mathit{states}(\varphi) \subseteq \overline{\mathit{states}(\psi_{\mathbf{CNF}})}$ holds.

The argumentation of **C2** and **C3** is very similar except that we need to encode the progression (for **C2**) and regression (for **C3**). A formula $\varphi_1 \wedge T_A \wedge \varphi'_2$ represents all pairs of states $\langle s, s' \rangle$ where s satisfies φ_1, s' satisfies φ_2 and $s[a] = s'$ for some $a \in A$. This formula being unsatisfiable therefore means that all successors of any $s \in \mathit{states}(\varphi_1)$ are contained in $\overline{\mathit{states}(\varphi_2)}$, or equivalently that all predecessors of any $s' \in \mathit{states}(\varphi_2)$ are contained in $\overline{\mathit{states}(\varphi_1)}$.

In summary, the formulas for which UNSAT certificates are requested directly imply the statement, and we can verify each UNSAT certificate in time polynomial in its size with the help of a SAT verifier. □

With the above extension to the proof system, we can build a proof akin to Table 1 by replacing statements **B1** and **B3** with **C1** and **C3**, respectively. But in order to verify the proof we must provide it with the appropriate UNSAT certificates.

Basic statements **C3a** in (P$_i$) require an UNSAT certificate for $\Lambda'_{\mathrm{U}} \wedge T_{\mathbf{A}} \wedge \neg\gamma_i$ for each $\gamma_i \in \Lambda_{\mathrm{U}}$, which we can get from SAT calls that are performed when PDR adds a clause to a layer either during a failed obligation extension or during clause pushing. When an obligation (s, i) could not be extended, PDR called SAT on the formula $\varphi \wedge T_{\mathbf{A}} \wedge \Lambda'_{i-1}$ with $\varphi = \bigwedge_{v \in s} v \wedge \bigwedge_{v \in \mathbf{V} \setminus s} \neg v$ encoding state s, and this call returned UNSAT. PDR then tries to iteratively remove conjuncts from φ (while keeping the negation of at least one goal literal) such that the resulting formula is still unsatisfiable. The last such φ' is negated and added to layers Λ_j with $j \leq i$. Thus for each γ added as a result of a failed obligation we have an UNSAT certificate for $\Lambda'_{i-1} \wedge T_{\mathbf{A}} \wedge \neg\gamma$.[1] This certificate is also valid for $\Lambda'_{j-1} \wedge T_{\mathbf{A}} \wedge \neg\gamma$. (with $j \leq i$), since Λ'_{j-1} contains all clauses in Λ'_{i-1}. Similarly, during clause pushing a clause γ can only be pushed to Λ_j if a SAT call of the form $\Lambda'_{j-1} \wedge T_{\mathbf{A}} \wedge \neg\gamma$ returned UNSAT, meaning we can produce a certificate for this formula as well.

[1]When utilizing a technique called inductive reason minimization, the SAT call is altered to $\Lambda'_{i-1} \wedge \gamma' \wedge T_{\mathbf{A}} \wedge \neg\gamma$, i.e., we assume γ is already added to Λ_{i-1}. Since this will happen immediately afterwards and $\mathit{states}(\Lambda_{x-1} \wedge \gamma)$ is thus still a subset of L_{U}, the certificate is still valid for the formula in the basic statement.

Altogether we can say that whenever a clause γ_i is added to a Λ_j, an UNSAT certificate for $\Lambda'_{j-1} \wedge T_{\mathbf{A}} \wedge \neg\gamma_i$ can be efficiently produced. Since Λ_{U} consists of a superset of the clauses in Λ'_{x-1} (where Λ_x is the state of Λ_{U} at the time when γ_i is added to it), the UNSAT certificate for these SAT calls can also be used as certificates for $\Lambda'_{\mathrm{U}} \wedge T_{\mathbf{A}} \wedge \neg\gamma_i$ for all $\gamma_i \in \Lambda_{\mathrm{U}}$, which are exactly the certificates we need.

For basic statements **C1a** in (Q_i) we need an UNSAT certificate for $\varphi_{\mathbf{G}} \wedge \neg\gamma_i$ for each $\gamma_i \in \Lambda_{\mathrm{U}}$. Since $\varphi_{\mathbf{G}}$ is a conjunction of literals and γ_i is a clause, these types of formulas consist of only unit clauses, meaning UNSAT can be shown with a single resolution step. Finally, basic statement **C1b** in (R_1) requires an UNSAT certificate for $\varphi_{\mathbf{I}} \wedge \Lambda_{\mathrm{U}}$. Since $\varphi_{\mathbf{I}}$ represents an assignment to all variables $v \in \mathbf{V}$, UNSAT can be shown with unit propagation, which again is easily translated into a certificate.

In summary, all basic statements from Table 1 can be replaced with statements for CNF formulas from Definition 2, and appropriate UNSAT certificates are either provided through SAT calls performed by PDR or can be constructed in time polynomial in Λ_{U}:

Theorem 4. *For PDR with SAT on PSN tasks, a proof for the proof system extended by Definition 2 can be generated in time polynomial in the runtime of PDR and can be verified in time polynomial in the size of the proof.*

PDR without SAT

When PDR without SAT is applied on a PSN task, it cannot efficiently perform the checks required for pushing clauses and thus omits this phase. While this reduces PDR's ability to detect unsolvability, PDR without SAT can still detect unsolvable PSN tasks and as we will see we can produce a proof for this case as well.

The only point where clauses get added to layers in this setting is when an obligation extension (s, i) fails. The clause to add is then determined by first finding several sets of literals for each action $a \in \mathbf{A}$ such that each set either consists of a precondition literal l which is not satisfied in s or is a clause $\gamma \in \Lambda_{i-1}$ which would be violated in $s[a]$. The final clause is then produced by finding a preferably small set of literals that includes at least one literal set from each action, as well as at least one goal literal.

More formally, we can say that (P_i^*) for each $\gamma_i \in \Lambda_{\mathrm{U}}$ and each action $a \in \mathbf{A}$ there is some clause γ_j in Λ_{U} such that *states*$(\neg\gamma_i)[a] \subseteq$ *states*$(\neg\gamma_j)$ holds.[2] Furthermore, since each clause $\gamma_i \in \Lambda_{\mathrm{U}}$ contains at least one goal literal, we have that (Q_i^*) *states*$(\neg\gamma_i) \cap \mathbf{S}_G \subseteq \emptyset$ holds. Finally, since $\mathbf{I}$ is not in L_{U}, there must be some clause γ in Λ_{U} that is not satisfied by $\mathbf{I}$, which we can express with (R_1^*) $\{\mathbf{I}\} \subseteq$ *states*$(\neg\gamma)$. These statements enable us to build a proof:

Theorem 5. *For tasks in PSN, a proof for PDR without SAT can be efficiently generated based on state sets states*$(\neg\gamma_i)$ *for each* $\gamma_i \in \Lambda_{\mathrm{U}}$*, which can be represented by any* $\mathbf{R}$*-formalism efficiently supporting* ***CL****. In order to efficiently verify the proof,* $\mathbf{R}$ *must efficiently support* ***CL****,* $\wedge$***BC****,* ***SE****,* ***RN***$_\prec$ *and* ***CO****.*

[2]For actions a not applicable in *states*$(\neg\gamma_i)$ we can choose any γ_j since *states*$(\neg\gamma_i)[a]$ is empty.

Proof. The statements from (P_i^*), (Q_i^*) and (R_1^*) actually exactly match the definition of a *1-disjunctive certificate* (Eriksson, Röger, and Helmert 2017), which is a family of sets whose union forms a set that is inductive ($S[\mathbf{A}] \subseteq S$), contains no goal state and contains the initial state. The interesting twist on 1-disjunctive certificates is that they actually have stronger properties that imply the aforementioned ones but can be verified without needing to consider the explicit union of the sets. For $\mathcal{F} = \{$*states*$(\neg\gamma_i) \mid \gamma_i \in \Lambda_{\mathrm{U}}\}$, these stronger properties are exactly the ones we mention in (P_i^*), (Q_i^*) and (R_1^*), meaning $\mathcal{F}$ is a 1-disjunctive certificate. While 1-disjunctive certificates are a different form of certificates from the proof-based certificates considered here, we can integrate them into the proof system with a translation provided by Theorem 5.10 from Eriksson (2019a).

Regarding representation and required operations, we can state (P_i^*) with basic statement **B2** and (Q_i^*) and (R_1^*) with **B1**, where each $\neg\gamma_i$ is represented by $\mathbf{R}$ with the help of **CL**. For verifying statements of type (P_i^*) we need **CL**, $\wedge$**BC**, **RN**$_\prec$ and **SE**; for statements of type (Q_i^*) we need **CL**, $\wedge$**BC** and **CO**, and for statements of type (Q_i^*) we need **CL**, $\wedge$**BC** and **CO**, and for (R_1^*) we need **SE**. □

Reversing the Search Direction

The original PDR algorithm, which was designed for more general transition systems with potentially several initial states, performs its search in the reverse direction, where the obligation space is explored backwards from the goal description and the layers Λ_i denote properties that states at most i steps away from the initial states must satisfy. Suda's (2014) adaption mainly focuses on the forward direction, which has the advantage that the obligations consider single states rather than sets of states, but also discusses the alternate search direction.

Since the transition function used in the SAT version can be used for both directions, PDR with SAT can reverse the search direction by only switching the initial state and goal description and switching the primed and unprimed variables in the transition function. This in turn requires us to change our proof from Table 1 in an analogous fashion. We need to replace $\mathbf{S}_G$ with $\{\mathbf{I}\}$ in all statements of type Q, replace $\{\mathbf{I}\}$ with $\mathbf{S}_G$ in statement R_1, and change all occurrences of $[\mathbf{A}]L_{\mathrm{U}}$ with $L_{\mathrm{U}}[\mathbf{A}]$, i.e., consider the set of successors of L_{U} rather than the set of all predecessors. Intuitively, the proof now states that all successors of L_{U}, the initial state and no goal state is contained in L_{U}.

For PDR without SAT, reversing the search direction is not as trivial since we normally would now need to deal with sets of states in the obligations. For STRIPS tasks however, Suda (2014) argues that we can transform a task Π to a different STRIPS task Π^d that is dual in the sense that if π is a plan in Π, then reversing π is a plan for Π^d (Suda 2013). The general idea is to describe states by what *can be false*, rather than what is true. The initial state in Π^d is given by the variables that are not true in $\mathbf{G}$, and the goal description by the variables false in $\mathbf{I}$. Intuitively, solutions to Π^d describe backward paths from goal states to a state that is a *subset* of the initial state. When running PDR on Π^d, the models $\mathcal{I}$ of

Λ_i can similarly be understood to denote states where certain variables can be false, i.e., states where they are false as well as supersets of those states. Due to the monotonicity of STRIPS, we can ignore the supersets and interpret variables that can be false as simply being false. This means that interpreting Λ_{U} for Π yields a *Horn formula* (since we have information about variables being *false*) denoting a set of states whose successors are also in Λ_{U}, which contains $\mathbf{I}$ but no $s_{\mathbf{G}} \supseteq \mathbf{G}$. From this we can build a proof analogously to the SAT case.

When reversing the search direction, PDR also adds invariants to its layers in the form of clauses which are true for any state reachable by $\mathbf{I}$. The purpose of these invariants is to guide the backwards search towards the initial state. When building a proof we need to ensure that this information can also be represented by the formalism representing the layers. For the invariants considered by Suda (2014) this is the case since they are of the form $\neg a \vee \neg b$, which can be represented by both CNF and Horn formulas.

Experiments

In order to evaluate whether generating and verifying certificates for PDR is practically feasible, we augmented the implementation of PDR from Suda (2014)[3] (which considers only STRIPS tasks, utilizes no SAT solver and searches forward) to be able to generate certificates when run with standard settings. We denote this certifying version as $\mathrm{PDR_C}$. In order to verify the generated certificates we augmented the proof verifier from Eriksson (2019a), denoted as verify, by additionally supporting dual-Horn formulas as representation formalism.

The experiments were performed on single cores in a cluster consisting of Core Intel Xeon Silver 4114 processors with a clock speed of 2.2 GHz. As a benchmark set we chose the same collection used in previous work in certifying unsolvability (Eriksson 2019b), minus the domains bag-barman and tetris, which PDR does not support due to negative preconditions. We ran both original PDR and $\mathrm{PDR_C}$ with limits of 30 minutes and 2 GiB of memory, and verify was given 2 hours and 2 GiB of memory to verify generated certificates. All used code (Eriksson and Helmert 2020a; 2020c) and the produced data (Eriksson and Helmert 2020b) are publicly available.

Table 2 summarizes for how many problems PDR could detect unsolvability, $\mathrm{PDR_C}$ could additionally also construct a certificate within the same limits, and verify could verify generated certificates. First comparing PDR and $\mathrm{PDR_C}$, we see that only four tasks were lost due to the overhead incurred by creating a certificate. This is not surprising since the proof mainly consists of writing a description for the clauses of L_{U}. Figure 2a, which compares the runtime of PDR and $\mathrm{PDR_C}$, reinforces this observation: $\mathrm{PDR_C}$ never needs significantly more time than PDR and sometimes even needs less time, suggesting that the overhead for generating the certificate is not much higher than random noise.

[3]`https://github.com/quickbeam123/PDRplan` (accessed 19.11.2019)

	PDR	$\mathrm{PDR_C}$	verify
3unsat (30)	10	10	10
bag-gripper (25)	0	0	0
bag-transport (29)	1	1	1
bottleneck (25)	20	20	18
cave-diving (25)	5	5	5
chessboard-pebbling (23)	3	3	3
diagnosis (20)	15	15	15
document-transfer (20)	4	4	4
mystery (9)	2	2	2
nomystery (150+24)	132	130	130
pegsol (24)	16	14	14
pegsol-row5 (15)	3	3	3
rovers (150+20)	164	164	164
sliding-tiles (20)	0	0	0
tpp (25+30)	13	13	13
total (704)	388	384	382

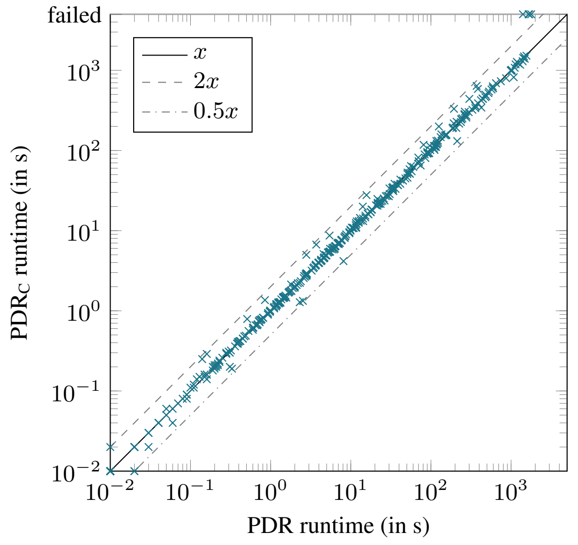
Table 2: Coverage results for PDR (original version), $\mathrm{PDR_C}$ (certifying version) and verify (certificate verifier).

	base	certifying	verifier
PDR	388	384	382
FD-$h^{\mathrm{M\&S}}$	224	197	178
FD-$h^{\max}$	203	156	140
DFS-CL	394	386	385

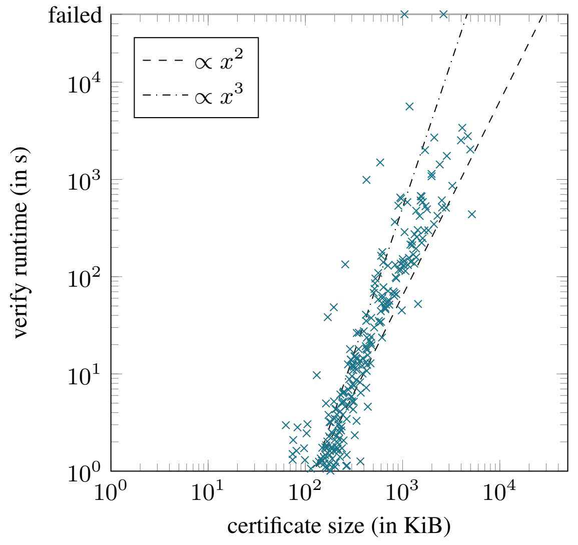
Table 3: Coverage comparison between different certifying planners. The uncertified version is denoted by "base".

When looking at whether the generated certificates could be verified within limits, only two tasks in the bottleneck domain could not be verified due to a timeout. This suggests that PDR finds rather concise reasons for unsolvability that are not difficult to verify; and indeed, the generated certificates are on average less than 0.5 MiB in size. Figure 2b takes a closer look at how certificate size and verification time are related, supporting the claim that certificates can be verified in time polynomial in their size.

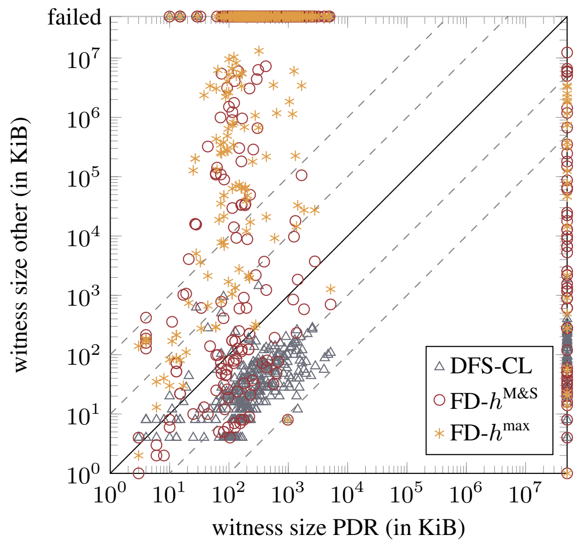
Finally, we also compared certified PDR to other certified planners, namely DFS-CL and updated versions of FD-$h^{\max}$ and FD-$h^{\mathrm{M\&S}}$ from Eriksson, Röger, and Helmert (2018). Looking at the coverage results in Table 3, we see that PDR performs similarly to DFS-CL, both in terms of absolute coverage and in how many tasks are lost in the certifying version or when trying to verify certificates. While the former has no straightforward explanation (in many tasks one algorithm succeeds while the other fails), the latter is likely due to the fact that DFS-CL also finds very concise final reasons for unsolvability and does not need to document the search process on a state-by-state level. Figure 2c reinforces this explanation, showing that DFS-CL in general even produces smaller certificates than PDR, while FD-$h^{\max}$ and FD-$h^{\mathrm{M\&S}}$ require certificates up to five magnitudes larger than PDR. Figure 2d on the other hand shows that these differences cannot be directly transferred to verification time: while DFS-CL in general seems to require less time for verification, PDR and FD-$h^{\mathrm{M\&S}}$ are overall more balanced (albeit with significant differences in both directions depending on the task), and $h^{\max}$ in general requires more time than PDR.



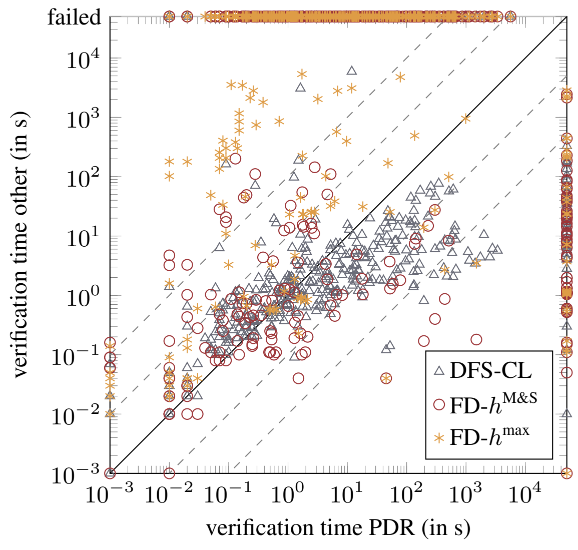
(a) Runtime comparison between PDR and PDR$_C$, showing the overhead incurred by certificate generation.



(b) Time needed for verifying a certificate as a function of the certificate size.



(c) Certificate size comparison PDR vs. other planners.



(d) Verification time comparison PDR vs. other planners.

Figure 2: Detailed results.

Conclusion

In this paper we presented a way to extend an unsolvability proof system for classical planning tasks in such a way that it can utilize UNSAT certificates as part of a proof. By studying the Property Directed Reachability algorithm (in its various variants), we showed that this enables us to also generate certificates of unsolvability for planning systems which rely on SAT queries. Furthermore we have also shown that if PDR is used without a SAT solver, the existing proof system is already expressive enough to represent concise certificates of unsolvability in its unmodified form.

Much of the recent work in planning as satisfiability (e.g., Rintanen, Heljanko, and Niemelä 2006; Rintanen 2012) focuses on the question of how to best represent the planning transition semantics in a SAT formula. One question that naturally arises is which kinds of transition formulas can be natively covered by the existing proof system and which techniques might require extensions. More ambitiously, one could also consider including the argument why this encoding is correct directly into the proof certificate itself.

In order to be able to produce certificates for more traditional SAT-based planning systems with modern upper-bounding techniques (Abdulaziz, Gretton, and Norrish 2017), three ingredients would be needed. Firstly, the computation of meaningful horizon upper bounds would need to be certified. Secondly, the reasoning of the actual SAT planner would need to be certified in order to establish a lower bound. Finally, one would need to certify that the combination of the lower and upper bounds proves the absence of a solution.

Related to this but more generally, it would be interesting to investigate how to certify proofs of unsolvability that rely on problem reformulations, for example based on symmetry (e.g., Pochter, Zohar, and Rosenschein 2011) or forward-backward duality (Suda 2013), or to incorporate techniques like partial-order reduction (Wehrle and Helmert 2012) and dominance pruning (Torralba and Hoffmann 2015). All of these techniques require a form of "what-if" reasoning, essentially arguing that candidate solutions of a certain kind need not be considered because if they exist, then there also exist other solutions (which are considered). Formalizing arguments of this kind in a uniform, automatically verifiable way would go a long way towards fully capturing the richness of current planning approaches for the purposes of certified correct computations.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Certified Correctness and Guaranteed Performance for Domain-Independent Planning" (CCGP-Plan).

References

Abdulaziz, M.; Gretton, C.; and Norrish, M. 2017. A state-space acyclicity property for exponentially tighter plan length bounds. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 2–10. AAAI Press.

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.

Bradley, A. R. 2011. SAT-based model checking without unrolling. In Jhala, R., and Schmidt, D., eds., *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2011)*, 70–87.

Brafman, R. I., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research* 18:315–349.

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

Eén, N.; Mishchenko, A.; and Brayton, R. 2011. Efficient implementation of property directed reachability. In Bjesse, P., and Slobodova, A., eds., *Proceedings of the 11th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2011)*, 125–134.

Eriksson, S., and Helmert, M. 2020a. Code from Eriksson-Helmert, ICAPS 2020. `https://doi.org/10.5281/zenodo.3691796`.

Eriksson, S., and Helmert, M. 2020b. Data set from Eriksson-Helmert, ICAPS 2020. `https://doi.org/10.5281/zenodo.3691812`.

Eriksson, S., and Helmert, M. 2020c. Modified PDRplan from Eriksson-Helmert, ICAPS 2020. `https://doi.org/10.5281/zenodo.3694110`.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability certificates for classical planning. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 88–97. AAAI Press.

Eriksson, S.; Röger, G.; and Helmert, M. 2018. A proof system for unsolvable planning tasks. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 65–73. AAAI Press.

Eriksson, S. 2019a. *Certifying Planning Systems: Witnesses for Unsolvability*. Ph.D. Dissertation, University of Basel.

Eriksson, S. 2019b. Unsolvable PDDL benchmarks. `https://doi.org/10.5281/zenodo.3355446`.

Haslum, P. 2017. INVAL: the Independent PDDL plan Validator. `https://github.com/patrikhaslum/INVAL`. Accessed September 29, 2017.

Heule, M.; Hunt, W. A.; and Wetzler, N. 2013a. Trimming while checking clausal proofs. In Jobstmann, B., and Ray, S., eds., *Proceedings of Formal Methods in Computer Aided Design (FMCAD 2013)*, 181–188. IEEE.

Heule, M. J. H.; Hunt, W. A.; and Wetzler, N. 2013b. Verifying refutations with extended resolution. In *Proceedings of the Twenty-Fourth International Conference on Automated Deduction (CADE-24)*, 345–359. Springer, Berlin, Heidelberg.

Howey, R., and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2.1 used in the International Planning Competition. In Edelkamp, S., and Hoffmann, J., eds., *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*.

Huang, R.; Chen, Y.; and Zhang, W. 2012. SAS$^+$ planning as satisfiability. *Journal of Artificial Intelligence Research* 43:293–328.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, 359–363. John Wiley and Sons.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 211–215. AAAI Press.

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

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12–13):1031–1080.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Schaefer, T. J. 1978. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC '78)*, 216–226. New York: ACM Press.

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

Suda, M. 2013. Duality in STRIPS planning. arXiv:1304.0897v1 [cs.AI].

Suda, M. 2014. Property directed reachability for automated planning. *Journal of Artificial Intelligence Research* 50:265–319.

Torralba, Á., and Hoffmann, J. 2015. Simulation-based admissible dominance pruning. In Yang, Q., and Wooldridge, M., eds., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1689–1695. AAAI Press.

van Gelder, A. 2002. Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM 2002)*.

van Gelder, A. 2008. Verifying RUP proofs of propositional unsatisfiability. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008)*.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 297–305. AAAI Press.