A Comparison of Unsolvability Certificates in Planning and Model Checking

Malte Helmert, Tanja Schindler

University of Basel, Switzerland
{malte.helmert,tanja.schindler}@unibas.ch

Abstract

Classical planning and checking the safety of hardware models are similar problems that can both be expressed as reachability in transition systems. When an algorithm claims that a set of target states is not reachable, this claim cannot be easily independently verified. Certifying algorithms address this limitation by producing computer-verifiable proofs of unreachability. Studying classical planning and hardware model checking, we compare both the underlying representation formalisms and the approaches that have been proposed for certifying unreachability. We show how bounded-cost plan existence can be expressed as a hardware circuit and that the certificates used in planning can be understood as a special case of certificates used in hardware model checking.

Introduction

In many areas of AI, *trustworthiness* has become an important topic. One way to increase the trust in a system is to provide a way to independently verify that its outputs are correct. For this reason, the planning community routinely uses *plan validators* (e.g., Howey, Long, and Fox 2004).

However, plan validators cannot verify that a planning task is unsolvable or that a plan is optimal. In many research communities that deal with reasoning problems similar to planning, this gap has been filled by *certifying algorithms* (McConnell et al. 2011) that allow automatic independent verification of *all* claims made by an algorithm (Bjørner et al. 2025). For example, unsatisfiability certificates have been mandatory in the SAT Competition since 2013 (e.g., Wetzler, Heule, and Hunt 2014; Bogaerts et al. 2023).

Hardware model checking is a reasoning problem with strong similarities to classical planning in which there has been a recent push for *safety certificates*, the equivalent of unsolvability certificates in planning (Froleyks et al. 2025). In this work we investigate the relationship between classical planning and hardware model checking and show how bounded-cost plan existence for STRIPS planning tasks can be compiled into AIG circuits for hardware model checking. We then discuss the relationship between certificates that have been proposed in the two research areas and show that certificates in planning can be seen as a special case of certificates used in hardware model checking. We believe that understanding these connections is important for the further development of certificate mechanisms for planning.

Planning vs. Hardware Model Checking

We use *transition systems* as a unifying mathematical model that can be used for the two problems we study in this paper, STRIPS planning and hardware model checking.

Definition 1 (transition system)**.** *A transition system $\mathcal{T} = \langle S, \Sigma, c, T, S_0, S_\star \rangle$ consists of a finite set of* states S*, a finite set of* transition labels Σ*, a* cost function $c : \Sigma \to \mathbb{N}_0$*, a* transition relation $T \subseteq S \times \Sigma \times S$*, a set of* initial states S_0*, and a set of* accepting states $S_\star$*.*

We write $s \xrightarrow{a} s'$ to denote the existence of a transition from s to s' labeled by a, i.e., $\langle s, a, s' \rangle \in T$.

We only consider deterministic *transition systems, where $s \xrightarrow{a} s' \wedge s \xrightarrow{a} s''$ implies $s' = s''$: a state cannot have two different successors via the same transition label.*

We can think of transition systems as directed graphs with additional annotations and assume familiarity with common concepts for such graphs such as paths and reachability.

Our notations deviate somewhat from the conventions in the planning literature (Σ instead of L for labels; a set of initial states rather than a single one; "accepting" states instead of "goal" states), so that the terminology is a reasonable fit for both planning and hardware model checking. For hardware model checking, it is useful to reserve the symbol L for latches, we can have arbitrarily many initial states, and the word "goal states" is unnatural because the accepting states are generally the ones we would like to *avoid*.

In both planning and model checking, transition systems are specified compactly by formalisms based on propositional logic. We assume familiarity with propositional logic.

Planning

We consider planning tasks expressed in propositional STRIPS with action costs because this is what the certifying approach we consider in this paper uses (Dold et al. 2025). However, we will comment on more expressive planning formalisms, for example with general action preconditions (Pednault 1989) or axioms (Thiébaux, Hoffmann, and Nebel 2005), where appropriate.

Definition 2 (planning task)**.** *A (STRIPS)* planning task *is a tuple $\Pi = \langle V, A, I, G \rangle$ where:*

- *V is a finite set of propositional* state variables*.*

- A *is a finite set of actions, where each action* $a \in A$ *has* preconditions, add effects *and* delete effects $pre(a), add(a), del(a) \subseteq V$ *and a* cost $cost(a) \in \mathbb{N}_0$.
- $I \subseteq V$ *is the* initial state.
- $G \subseteq V$ *is the* goal.

A state s of a planning task can be viewed as the set of atoms $s \subseteq V$ that are true in the state or as a truth assignment (propositional logic interpretation) $s : V \to \{0, 1\}$. Both views are convenient for different purposes, and we use both in this paper.

The planning task $\Pi = \langle V, A, I, G \rangle$ induces the transition system $\mathcal{T}(\Pi) = \langle S, \Sigma, c, T, S_0, S_\star \rangle$ where:

- $S = 2^V$ (the power set of V) (P1)
- $\Sigma = A$ (P2)
- $c = \{a \mapsto cost(a) \mid a \in A\}$ (P3)
- $T = \{\langle s, a, s' \rangle \mid s \in S, a \in A, pre(a) \subseteq s, s' = (s \setminus del(a)) \cup add(a)\}$ (P4)
- $S_0 = \{I\}$ (P5)
- $S_\star = \{s \mid s \in S, G \subseteq s\}$ (P6)

In planning, we want to find *plans*, i.e., paths (sequences of transitions) in the transition system that lead from an initial state to an accepting state. If $\langle s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n \rangle$ is such a sequence, then $\sum_{i=1}^{n} c(a_i)$ is its *cost*.

We are interested in algorithms that certify that plans within a certain cost bound do not exist. For this purpose, we define the following decision problem.

Definition 3 (bounded-cost plan existence)**.** Bounded-cost plan existence *is the following decision problem:*

- Given: *a planning task* Π *and cost bound* $B \in \mathbb{N}_0$
- Question: *does* $\mathcal{T}(\Pi)$ *contain a path from an initial state to an accepting state with cost at most* B*?*

If the answer to the question is "yes", this is easily witnessed by the sequence of actions (transition labels) along the given path. It the answer is "no", a *certifying* planning algorithm produces a *lower-bound certificate*, i.e., a form of proof that can be fed to a validator to confirm that the algorithm indeed proved that no solutions within the bound exist (Mugdan, Christen, and Eriksson 2023; Dold et al. 2025).

The bounded-cost plan existence problem comes in two flavors: the given bound can either be non-strict (the more commonly considered version, as in Definition 3), asking for a plan of cost *at most* B, or strict, asking for a plan of cost *strictly below* B. For lower-bound certificates, we believe that the strict version is more useful because it can be used for the common use case of proving optimality: first provide a plan of (optimal) cost B, then prove that it is not possible to do strictly better than B. For costs that increase in discrete steps, as in our formalization, the distinction is not very important because a strict bound of B is semantically the same as a non-strict bound of $B - 1$, but strict bounds generalize better to settings without such discrete steps, for example when costs can be arbitrary rational numbers. We therefore use strict bounds in the following.

Hardware Model Checking

Classical planning is closely related to (bit-level) *hardware model checking*, and specifically the checking of safety properties. A plan for a classical planning task corresponds to a counterexample that shows the violation of a safety property, while proving unsolvability of a planning task corresponds to verifying a safety property.

The model checking research community regularly hosts the *Hardware Model Checking Competition*[1] (*HWMCC*) as part of the CAV and FMCAD conferences. HWMCC'24, held at FMCAD 2024 and the 13th edition of the competition, was the first competition to require safety certificates, the equivalent of unsolvability certificates in planning.

Froleyks et al. (2025) describe why and how safety certificates were introduced to the bit-level track of HWMCC'24 and discuss the impact on the competition. Our formal definitions of hardware model checking and the associated certificates follow the setup that was used for this competition.

The input to a hardware model checking algorithm is a (sequential) Boolean *circuit*. We first provide a high-level definition following Froleyks et al., which leaves some details of the representation of circuits open.

By a Boolean *predicate* over a set of propositional variables V, we mean a property that is true or false for a truth assignment to these variables. We leave the representation of predicates open for now. A well-known representation is as a propositional formula. For example, the propositional formula $X \wedge Y$ represents a predicate that is true for exactly two truth assignments to the variables $\{X, Y, Z\}$. We will use predicates like formulas in the following. For example, if P and Q are predicates, then $P \wedge Q$ is true for all truth assignments that satisfy P and Q.

Definition 4 (circuit)**.** *A* circuit *is a tuple* $M = \langle I, L, R, F, P, C \rangle$ *where:*

- I *is a finite set of propositional variables called the* inputs.
- L *is a finite set of propositional variables disjoint from* I *called the* latches.
- $R = \{R_l \mid l \in L\}$ *is the set of* reset predicates *for the latches, where each* R_l *is a predicate over* $I \cup L$.
- $F = \{F_l \mid l \in L\}$ *is the set of* transition predicates *for the latches, where each* F_l *is a predicate over* $I \cup L$.
- P *is the* safety property*, a predicate over* $I \cup L$.
- C *is the* constraint*, a predicate over* $I \cup L$.

The reset predicates must be stratified*, i.e., there must not exist a cyclic dependency between reset properties involving multiple latches. The precise notion of dependency is elaborated when fixing a concrete representation for predicates.*

In the wider model checking literature, one would typically separate the definition of the model (here a sequential circuit defined by inputs, latches, reset and transition predicates) from the definition of the property to be checked (here given by a safety property and constraint). We follow Froleyks et al. (2025) by combining both aspects in the definition of circuits, which simplifies the presentation and makes circuits more analogous to planning tasks.

[1] https://hwmcc.github.io/

Like planning tasks, circuits compactly represent transition systems. Roughly speaking, latches are state variables, assignments to the inputs have an analogous role to actions, reset predicates define initial states, transition predicates define the transitions, the negation of the safety property defines the accepting states, and the constraint restricts the set of states.

Formally, the circuit $M = \langle I, L, R, F, P, C\rangle$ induces the transition system $\mathcal{T}(M) = \langle S, \Sigma, c, T, S_0, S_\star\rangle$ where:

- $S = \{s \mid s \in 2^{I\cup L}, s \models C\}$ (M1)
- $\Sigma = 2^I$ (M2)
- $c = \{a \mapsto 0 \mid a \in \Sigma\}$ (M3)
- $T = \{\langle s, a, s'\rangle \mid s \in S, a \in \Sigma, s' \in S$ s.t. for all $i \in I$, $(i \in s')$ iff $(i \in a)$, for all $l \in L$, $(l \in s')$ iff $s \models F_l\}$ (M4)
- $S_0 = \{s \mid s \in S$ s.t. for all $l \in L$, $(l \in s)$ iff $s \models R_l\}$ (M5)
- $S_\star = \{s \mid s \in S, s \models \neg P\}$ (M6)

From a planning perspective, some aspects of this definition warrant further discussion, and we will return to it soon when we compare the two formalisms.

We are now ready to define the model checking problem for such circuits, bearing in mind that we only obtain a fully specified algorithmic problem after we have also fixed the details of the representation. (Put differently, different representations give rise to different variants of the problem.)

Definition 5 (model checking)**.** *We consider the following model checking problem:*

- Given: *a circuit* M
- Question: *does* $\mathcal{T}(M)$ *contain a path from an initial state to an accepting state?*

We remark that the question is typically asked in the complementary way in the model checking literature: a positive answer would be one where the circuit is *safe* because *no* accepting state is reachable from an initial state. (All reachable states satisfy the safety property.) A path from an initial state to an accepting state then is a counterexample to safety, showing that an unsafe state can be reached. We ask for existence (rather than absence) of an unsafe path to make the similarity to plan existence more direct.

Indeed, comparing Definitions 3 and 5 we see that there are only two differences: the type of formalism used to specify the transition system, and the fact that the planning problem is quantitative in nature (considering the costs of transitions) while the model checking problem is qualitative. We will see later how this gap can be bridged.

And-Inverter Graphs

The (bit-level) HWMCC uses *and-inverter graphs* (*AIGs*, Mishchenko et al. 2005) in the AIGER format (Biere, Heljanko, and Wieringa 2011) to specify circuits.

Definition 6 (AIG circuit)**.** *An* AIG circuit *is a tuple* $M_{\text{AIG}} = \langle I, L, G, \mathit{src}, \mathit{reset}, \mathit{trans}, \mathit{prop}, \mathit{cons}\rangle$ *where:*

- *I, L and G are disjoint finite sets of propositional variables called the* inputs*,* latches *and* gates*. Let* $V = I \cup L \cup G$ *be the* variables *of the circuit, and let* $\mathit{Lit} = V \cup \{\neg v \mid v \in V\} \cup \{\top, \bot\}$ *be the* literals *over* V*.*
- $\mathit{src} : G \to \mathit{Lit} \times \mathit{Lit}$ *defines the* inputs *for the gates.*
- $\mathit{reset} : L \to \mathit{Lit}$ *defines the* resets *for the latches.*
- $\mathit{trans} : L \to \mathit{Lit}$ *defines the* transitions *for the latches.*
- $\mathit{prop} \in \mathit{Lit}$ *defines the* property*.*
- $\mathit{cons} \in \mathit{Lit}$ *defines the* constraint*.*

The variables V are totally ordered by a relation $\prec$ such that inputs of a gate precede the gate: if $\mathit{src}(g) = \langle \ell_1, \ell_2\rangle$ where ℓ_i is a literal over the variable $v_i \in V$, then $v_i \prec g$.

Moreover, the resets are stratified *under the same relation $\prec$: if $\mathit{reset}(l) = v$ with $v \neq l$ or $\mathit{reset}(l) = \neg v$, then $v \prec l$.*

Intuitively, the gates of an AIG circuit represent AND logic gates with two inputs, either or both of which can be negated. Such gates can compactly encode all other common types of logic gates.

An AIG circuit $M_{\text{AIG}} = \langle I, L, G, \mathit{src}, \mathit{reset}, \mathit{trans}, \mathit{prop}, \mathit{cons}\rangle$ with variables V and literals *Lit* represents a circuit $M = \langle I, L, R, F, P, C\rangle$ (with the same inputs and latches) in the sense of Def. 4 as follows. For every literal $\ell \in \mathit{Lit}$, we define a predicate over $I \cup L$, denoted by $[\![\ell]\!]$. If $\ell \in \{\top, \bot, v, \neg v\}$ for $v \in I \cup L$, it already is a predicate over $I \cup L$, and we set $[\![\ell]\!] = \ell$. For gate variables $g \in G$ with $\mathit{src}(g) = \langle \ell_1, \ell_2\rangle$, we set $[\![g]\!] = [\![\ell_1]\!] \wedge [\![\ell_2]\!]$ and $[\![\neg g]\!] = \neg[\![g]\!]$. The total order $\prec$ ensures that this is not a cyclic definition.

The reset predicates, transition predicates, safety property and constraint follow directly from the AIG circuit: $R_l = [\![\mathit{reset}(l)]\!]$ and $F_l = [\![\mathit{trans}(l)]\!]$ for all latches l, $P = [\![\mathit{prop}]\!]$, and $C = [\![\mathit{cons}]\!]$. Restricting to single literals is not limiting because gates can represent arbitrary predicates.

We observe that the definition of stratified resets explicitly allows directly self-referential resets of the form $\mathit{reset}(l) = l$ as a special case. (Without this special case, such resets would violate stratification because $l \not\prec l$.) With condition (M5) in the definition of $\mathcal{T}(M)$ we provided, such state variables can be freely chosen to be true or false initially, and this is the idiomatic way to express state variables with unconstrained initial value.

Indeed, in the AIGER format (Biere, Heljanko, and Wieringa 2011) that is commonly used for hardware model checking benchmarks, such "unconstrained" resets are the *only* reset functions allowed besides the constant literals $\top$ and $\bot$. However, the general stratified resets in Def. 6 are allowed in safety certificates for the Certifaiger certificate checker (Froleyks 2024) used in the HWMCC competitions.

Comparison of the Formalisms

Our major objective in this paper is to formally relate the certificate mechanisms for STRIPS planning and hardware model checking. In order to achieve this, we must first be able to translate between the formalisms themselves and in particular represent a STRIPS task as a circuit. Towards this end, we first discuss the main similarities and differences in the equations (P1–P6) vs. (M1–M6).

The set of states (P1, M1) is defined very similarly in the two formalisms. States are assignments to propositional variables (state variables in planning, inputs and latches in

circuits). From a planning perspective, it may appear somewhat surprising that the inputs – which receive new values in every transition – are part of the state definition, but this is an important aspect of circuit semantics. For example, the definitions of initial states (via resets) and accepting states (via the property) involve the inputs.

Another difference is that for circuits the set of states is further restricted by the constraint. STRIPS planning does not have such a feature, but it is not difficult to simulate in richer planning formalisms. For example, we can use a derived variable C to express a constraint with axioms and then add C to all action preconditions and the goal. This does not remove constraint-violating states from the transition system, but rules out that they are part of a solution, which suffices. In PDDL3 (Gerevini and Long 2005), this can be expressed directly as the plan constraint "(always C)".

The definition of transition labels (P2, M2) highlights a major difference between the two formalisms. In planning, the actions serve as transition labels, which means that the number of labels is at most linear in the task representation size. In circuits, each truth assignment to the inputs is a different transition label, so the number of labels and hence the maximal number of successors of a state in the transition system can be exponential in the circuit representation size.

This has immediate consequences for the complexity of certain subproblems. For example, checking if a plan that consists of a constant number k of transitions exists is a polynomial problem (exponential in k). In circuits, checking if a circuit admits an unsafe path of length 1 is already NP-complete, even if we restrict resets to constants (to obtain a single initial state).

This implies that a polynomial compilation from circuits to planning tasks that preserves the length of transition sequences exactly in the sense of Nebel (2000) is not possible. A polynomial compilation from circuits to planning thus requires encoding a single circuit transition as a sequence of multiple planning transitions.

Transition costs (P3, M3) only exist in planning. Therefore, in order to express bounded-cost plan existence via circuits, we have to keep track of the cost of a plan in a different way, by encoding it inside the state of the circuit.

Regarding transitions (P4, M4), planning follows an action-centric update semantics where the successor state is computed as the current state plus changes (delete effects and add effects). Latches in circuits use a variable-centric semantics and have no notion like the "frame" in planning (variables keep their values unless explicitly modified). Instead, the semantics is in the spirit of successor-state axioms in situation calculus (Reiter 2001) where the truth value of every state variable in the successor state is explicitly specified as a predicate of the current state and transition label.

Because the current values of the latches and inputs completely determine the values of the latches in the successor state, all possible state transitions from a given state lead to the same next values for the latches. The choice in the state transition is therefore not in how to update the latches but how to set the next inputs. These can be chosen arbitrarily, as long as the constraint is satisfied.

To convert between the two formalisms, planning action semantics can be encoded as successor-state axiom circuits. The converse direction of going from circuits to planning without an exponential increase in representation size requires breaking up a single circuit transition into multiple planning transitions as described above, due to the exponential difference in the number of transition labels.

The definition of initial states (P5, M5) is another major difference between the formalisms. Planning tasks have a single initial state, which is given directly. In hardware model checking, there can be exponentially many initial states with a complex structure.

However, the stratification requirement imposes some restrictions, which make reasoning about initial states somewhat more manageable. Without stratification, testing if there exists an initial state would already be an NP-complete problem, but because of stratification initial states always exist and can be produced systematically in a bottom-up computation that follows the variable order of the circuit, starting from any assignment to the inputs and unrestricted ($\mathit{reset}(l) = l$) latches. A conversion from circuits to planning tasks could represent these semantics with a multi-step initialization phase that records an assignment to the inputs and free latches in stages and then uses derived variables to evaluate the reset predicates.

Finally, the definition of accepting states (P6, M6) is similar in the two formalisms. STRIPS goals are restricted to conjunctions of atoms when interpreted as predicates, but richer planning formalisms permit arbitrary predicates. A superficial difference it that the target states are expressed positively in planning tasks (states that satisfy the goal condition) and negatively in circuits (state that do not satisfy the safety property).

Planning as Hardware Model Checking

We now describe how to convert planning tasks into circuits for hardware model checking, or more precisely: how to polynomially reduce the bounded-cost plan existence problem (the variant of Definition 3 with a strict cost bound) to the hardware model checking problem (Definition 5). We remark that the general idea of casting planning as model checking is not new (e.g., Cimatti et al. 2003; Balyo and Suda 2016), but this particular conversion to AIG circuits is.

It is important for the later discussion of certificates that the produced circuit does not just provide an arbitrary reduction but a direct simulation where every transition of the planning task that stays below the cost bound corresponds to a single transition of the circuit. As argued in the preceding section, a polynomial translation with this property does not exist for the converse direction. However, because both problems are PSPACE-complete (e.g., Sistla and Clarke 1985; Bylander 1994), more indirect conversions exist.

Given a STRIPS planning task $\Pi = \langle V, A, \mathit{Init}, G\rangle$ and cost bound $B \in \mathbb{N}_0$, we construct a circuit $M(\Pi, B) = \langle I, L, R, F, P, C\rangle$ as follows:

- $I = A$: we have one input for each action. Setting this input to 1 corresponds to selecting this action.
- $L = V \cup V_{\mathrm{c}}$ where $V_{\mathrm{c}} = \{c_0, \ldots, c_m\}$ with $m = \lceil \log_2 B \rceil - 1$ and the c_i are fresh variables not present in

V: the latches encode the planning task state and the cost accrued by the plan so far, using binary digits $c_0, \ldots, c_m$.

- $R = \{R_l \mid l \in L\}$ where $R_v = \top$ for all $v \in \mathit{Init}$, $R_v = \bot$ for all $v \in V \setminus \mathit{Init}$, and $R_{c_i} = \bot$ for all $0 \leq i \leq m$: initialize to the initial state of the planning task and a "plan cost so far" of 0.
- $F = \{F_l \mid l \in L\}$ is described below.
- $P = \neg \bigwedge_{v \in G} v$: the unsafe states correspond to the goal states of the planning task.
- C is the conjunction of the condition $\sum_{i=0}^{m} 2^i c_i < B$, an at-most-one constraint on the inputs (actions) I, and the condition $\bigwedge_{a \in A}(a \rightarrow \bigwedge_{v \in \mathit{pre}(a)} v)$: it is not allowed to exceed the cost bound, at most one action must be selected in each step, and we may only select actions whose preconditions are satisfied.

  Note that it is permitted to select no action. Otherwise it would not be possible for the circuit to simulate entering a planning task state in which no action is applicable. (The circuit cannot enter a state without simultaneously choosing the next inputs.) Choosing no action makes it possible to satisfy the constraint for such states.

It remains to describe the transition predicates. For every planning task state variable v, we must describe the condition under which v is true after a state transition from a state s encoded by the latches via an action encoded by the inputs. Under STRIPS semantics, v is true after an action application if it is added by a selected action or currently true and not deleted by a selected action. (This includes the case where v is currently true and no action is selected.) This leads to the following transition predicates F_v for all $v \in V$:

$$F_v = \bigvee_{a \in A, v \in \mathit{add}(a)} a \vee (v \wedge \neg \bigvee_{a \in A, v \in \mathit{del}(a)} a).$$

For the latches V_c that encode the plan cost so far, the transition predicates F_{c_i} must encode that selecting action a increases the encoded value by $\mathit{cost}(a)$. Because at most one action can be selected in each transition and not selecting an action should leave the encoded value as is, we can represent this relationship as the arithmetic expression

$$g' = g + \sum_{a \in A} a \cdot \mathit{cost}(a),$$

where g and g' represent the number encoded before and after the state transition in the latches V_c and we use the Boolean value 0 (not selected) or 1 (selected) for the input a as an integer in this arithmetic expression.[2] This shows that we can compute the new values of V_c with a basic arithmetic circuit using only operations such as addition and multiplication with 1 or 0, which are easy to represent in logic gates. The individual transition predicates F_{c_i} then correspond to the output gates of this circuit.

It is clear that R, F, P and C can be encoded as AIG circuits in such a way that M can be constructed in polynomial time from Π and B. (In fact, with a bit of care we can define a linear-time encoding.)

To illustrate the relationship between the planning task Π and the circuit $M(\Pi, B)$, consider a path $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$ in the transition system of Π with cost at most B. For all $0 \leq i \leq n$, set $g_i = \sum_{j=1}^{i} \mathit{cost}(a_j)$. Every state of $M(\Pi, B)$ encodes a planning task state s in variables V, the accumulated plan cost g in variables V_c, and the next action a in the inputs I. If we denote such a state by the triple $\langle s, g, a \rangle$, then the path in the circuit transition system that corresponds to the path π is $\langle s_0, g_0, a_1 \rangle \xrightarrow{a_2} \langle s_1, g_1, a_2 \rangle \xrightarrow{a_3} \ldots \xrightarrow{a_n} \langle s_{n-1}, g_{n-1}, a_n \rangle \xrightarrow{a_\bot} \langle s_n, g_n, a_\bot \rangle$, where $a_\bot$ represents that no action is selected.

This completes the reduction from bounded-cost plan existence to hardware model checking: the circuit exactly simulates the paths from the initial state with cost at most B, and this simulation ends in a circuit state violating the safety property iff the path in the planning task ends in a goal state.

Certificates

We now turn to certifying algorithms for planning and hardware model checking. In this setting, we do not trust the planning systems or model checkers and do not accept claims like "There does not exist a plan of cost at most B" at face value. Instead, we require *certifying* planning systems or model checkers, which produce a *certificate* of their claim as part of their output.

This certificate is then checked by a separate, independent component, the certificate validator. If the certificate passes validation and the validator itself can be trusted, then we can be certain that the claim made by the planning system is true.

Certificates for Planning

In the planning literature, certifying algorithms were first introduced for proving the unsolvability of planning tasks (Eriksson, Röger, and Helmert 2017, 2018; Eriksson 2019; Eriksson and Helmert 2020). More recent work introduced lower-bound certificates, showing that no plan within a given cost bound exists, which generalizes unsolvability (Mugdan, Christen, and Eriksson 2023; Dold et al. 2025). We follow the more recent work by Dold et al. (2025), which operates at a finer level of granularity than the earlier work and is closer in spirit to certificate approaches used in other research communities such as SAT (Wetzler, Heule, and Hunt 2014; Bogaerts et al. 2023).

In the work by Dold et al., a lower-bound certificate for bound B consists of a *pseudo-Boolean Circuit*, which encodes a set of state/path-cost pairs that satisfy an inductive invariant property φ, along with three proofs in the *cutting planes with reification* (CPR) proof system (Buss and Nordström 2021) which use φ to prove that no plan of cost strictly

[2] Because g and g' share the same finite range, the new path cost g' can exceed that range if g is close to B, in which case it is not possible to represent its value with the latches. This can only happen in circuit states where the successor state would violate the circuit constraint, so it is sufficient to handle such cases with any mechanism that ensures that the successor state violates the constraint. We omit further details for brevity.

less than B exists. We follow the certificate definition by Dold et al. with some minor simplifications.

For the purposes of this paper, we do not need to introduce the CPR proof system, but we do need to introduce pseudo-Boolean constraints.

Definition 7 (pseudo-Boolean constraint)**.** *A* pseudo-Boolean constraint *(*PB constraint*) over a set of propositional variables V is an inequality of the form*

$$\sum_{i=1}^{n} a_i \ell_i \geq A,$$

where the ℓ_i are literals over V and $a_1, \ldots, a_n, A \in \mathbb{N}_0$.

PB constraints are interpreted arithmetically by treating the truth values 0 and 1 as numbers and interpreting negative literals $\neg v$ as $1 - v$. In PB constraints, negative literals are written as $\overline{v}$ rather than $\neg v$. PB constraints generalize clauses: if all coefficients a_i and A are 1, the PB constraint in Def. 7 is equivalent to $\ell_1 \vee \cdots \vee \ell_n$.

The certificate approach heavily uses *reification* (representing constraints as variables): if C is a pseudo-Boolean constraint and r is a propositional variable, then $r \leftrightarrow C$ denotes the constraint that $r = 1$ iff C holds. Reifications can be represented as PB constraints in the sense of Definition 7, but the details of this are not important here. Reifications form the basis of *pseudo-Boolean circuits*.

Definition 8 (pseudo-Boolean circuit)**.** *A* pseudo-Boolean circuit *(*PB circuit*) over a set of propositional variables V is a pair $\langle \mathcal{C}, r \rangle$ where*

- *$\mathcal{C}$ is a sequence $\langle r_1 \leftrightarrow \varphi_1, \ldots, r_n \leftrightarrow \varphi_n \rangle$ of reifications, where each constraint φ_i only uses variables from the set $V \cup \{r_1, \ldots, r_{i-1}\}$, and*
- *$r \in \{r_1, \ldots, r_n\}$ is called the* output variable*.*

We can think of a PB circuit as computing a value for r from the values for V in the same way that a logic circuit computes an output from its inputs. PB circuits generalize regular logic circuits in the sense that standard logic gates can easily be expressed with pseudo-Boolean reifications.

A PB circuit for a planning task $\Pi = \langle V, A, I, G \rangle$ and cost bound B is a PB circuit whose input variables are the state variables V and variables $V_{\mathrm{c}} = \{c_0, \ldots, c_{\lceil \log_2 B \rceil - 1}\}$ that encode a cost of reaching a state represented by an assignment to V, as in our encoding of planning tasks as circuits.

A lower-bound certificate consists of a PB circuit that represents a set of state/path-cost pairs in a planning task, along with three proofs that the circuit has certain properties. The proof obligations are based on the following pseudo-Boolean encoding of Π, where $m = \lceil \log_2 B \rceil - 1$:

$$r_{\mathrm{I}} \leftrightarrow \sum_{v \in I} v + \sum_{v \in V \setminus I} \overline{v} \geq |V| \tag{1}$$

$$r_{\mathrm{G}} \leftrightarrow \sum_{v \in G} v \geq |G| \tag{2}$$

$$\Delta c^{=k} \leftrightarrow \sum\nolimits_{i=0}^{m} 2^i c'_i - \sum\nolimits_{i=0}^{m} 2^i c_i = k \tag{3}$$

$$\mathit{cost}_{\geq k} \leftrightarrow \sum\nolimits_{i=0}^{m} 2^i c_i \geq k \tag{4}$$

$$\mathit{cost}'_{\geq k} \leftrightarrow \sum\nolimits_{i=0}^{m} 2^i c'_i \geq k \tag{5}$$

$$\mathit{geq}_{v,v'} \leftrightarrow v + \overline{v'} \geq 1 \tag{6}$$

$$\mathit{leq}_{v,v'} \leftrightarrow \overline{v} + v' \geq 1 \tag{7}$$

$$\mathit{eq}_{v,v'} \leftrightarrow \mathit{leq}_{v,v'} + \mathit{geq}_{v,v'} \geq 2 \tag{8}$$

$$r_a \rightarrow \Delta c^{=\mathit{cost}(a)} + \sum_{v \in \mathit{pre}(a)} v + \sum_{v \in \mathit{add}(a)} v' + \sum_{v \in \mathit{del}(a)} \overline{v'} + \sum_{v \in V \setminus (\mathit{add}(a) \cup \mathit{del}(a))} \mathit{eq}_{v,v'} + \overline{\mathit{cost}'_{\geq B}} \geq 2 + |\mathit{pre}(a)| + |V| \tag{9}$$

$$r_{\mathrm{T}} \leftrightarrow \sum_{a \in A} r_a \geq 1 \tag{10}$$

The value k in (3) is a parameter: we instantiate this reification for every action cost that occurs in the planning task. Similarly, (4) and (5) are instantiated with $k = 1$ and $k = B$, (6)–(8) are instantiated with every state variable $v \in V$ and (9) with every action $a \in A$.

The encoding uses the variables $V \cup V_{\mathrm{c}}$ to encode a state and associated path cost and primed copies $V' \cup V'_{\mathrm{c}}$ of the same variables to encode a successor state and associated path cost. Each equation introduces reified variables that correspond to certain semantic concepts: r_{I} is true if we are in the initial state, r_{G} in a goal state, $\mathit{cost}_{\geq k}$ if we incurred a cost of at least k so far, r_a if the state/cost pairs described by variables $V \cup V_{\mathrm{c}}$ and $V' \cup V'_{\mathrm{c}}$ are connected by a transition using action a, and r_{T} if we take a transition by some action. We refer to Dold et al. (2025) for a more detailed discussion.

Given a planning task Π, we write $\mathcal{C}_\Pi$ for the set of constraints (1)–(10). If $\langle \mathcal{C}, r \rangle$ is a PB circuit, we write $\langle \mathcal{C}', r' \rangle$ for the same PB circuit where every variable v is replaced by its primed copy v'. We are now ready to define lower-bound certificates.

Definition 9 (lower-bound certificate)**.** *Let Π be a planning task, and let $B \in \mathbb{N}_0$ be a cost bound.*

A lower-bound certificate for Π with bound B *is a tuple $\langle \langle \mathcal{C}_\varphi, r_\varphi \rangle, \mathcal{P}_{\text{init}}, \mathcal{P}_{\text{goal}}, \mathcal{P}_{\text{ind}} \rangle$ where:*

- *$\langle \mathcal{C}_\varphi, r_\varphi \rangle$ is a PB circuit for Π and B.*
- initial state lemma: *$\mathcal{P}_{\text{init}}$ is a CPR proof for $\mathcal{C}_\Pi \cup \mathcal{C}_\varphi \models (r_{\mathrm{I}} \wedge \overline{\mathit{cost}_{\geq 1}}) \rightarrow r_\varphi$.*
- goal lemma: *$\mathcal{P}_{\text{goal}}$ is a CPR proof for $\mathcal{C}_\Pi \cup \mathcal{C}_\varphi \models (r_{\mathrm{G}} \wedge r_\varphi) \rightarrow \mathit{cost}_{\geq B}$.*
- inductivity lemma: *$\mathcal{P}_{\text{ind}}$ is a CPR proof for $\mathcal{C}_\Pi \cup \mathcal{C}_\varphi \cup \mathcal{C}'_\varphi \models (r_\varphi \wedge r_{\mathrm{T}}) \rightarrow r'_\varphi$*

Dold et al. (2025) show that a lower-bound certificate for Π and B exists iff there does not exist a plan for Π with cost strictly less than B.

Given a planning task Π and a tuple *Cert* claimed to be a lower-bound certificate, one can test in polynomial time in $\|\Pi\|$ and $\|\mathit{Cert}\|$ if *Cert* is a valid certificate and therefore B is indeed a lower bound on plan cost.

Moreover, Dold et al. (2025) show how an A* search using a pattern database heuristic (Edelkamp 2001) or $h^{\max}$ (Bonet and Geffner 2001) can be modified (with polynomial overhead) to produce a lower-bound certificate that proves optimality of the generated solution. They conjecture that

lower-bound certificates can also be produced with low overhead for a much wider class of planning approaches.

Certificates for Hardware Model Checking

The safety certificates for hardware model checking by Froleyks et al. (2025) follow a different philosophy from the lower-bound certificates for planning. Syntactically, a certificate takes the same form as the input to the model checking problem: it is a circuit. A circuit W that certifies the safety of another circuit M is called a *witness* for M.

Checking that a circuit is a witness requires proving several properties, similarly to the lemmas for lower-bound certificates (Def. 9). In order to formulate these properties, we must first encode circuit semantics in a similar way to constraints (1)–(10) for planning. Our presentation is based on Froleyks et al. (2025), which describes the encoding in somewhat abstract terms. We refine their description to a concrete encoding for AIG circuits.

Similarly to the planning case, the encoding uses propositional variables v to describe a circuit state and primed variables v' to describe a successor state after a state transition. Differently from the planning case, the encoding uses propositional formulas rather than pseudo-Boolean constraints.

Let $M = \langle I, L, G, \mathit{src}, \mathit{reset}, \mathit{trans}, \mathit{prop}, \mathit{cons}\rangle$ be an AIG circuit. We define the components of its encoding as follows:

- $G_M = \bigwedge_{g \in G, \mathit{src}(g) = \langle \ell_1, \ell_2 \rangle} (g \leftrightarrow (\ell_1 \wedge \ell_2))$
- for $K \subseteq L$: $R_M^K = \bigwedge_{l \in K} (l \leftrightarrow \mathit{reset}(l))$
- for $K \subseteq L$: $F_M^K = \bigwedge_{l \in K} (l' \leftrightarrow \mathit{trans}(l))$
- $P_M = \mathit{prop}$
- $C_M = \mathit{cons}$

G_M encodes that the gates of M are correctly evaluated. We must always enforce that this formula holds for the other parts of the encoding to make sense. R_M^K and F_M^K describe resets and transitions. They are parameterized with a subset of latches because this is needed to define witnesses. Finally, P_M and C_M encode that the property and constraint hold.

With this, we can define witnesses, which are the certificates used for hardware model checking. (A primed formula like C'_M is obtained from an unprimed formula like C_M by replacing each occurrence of a variable v by v'.)

Definition 10 (witness)**.** *Let M be an AIG circuit with latches L_M, let W be an AIG circuit with latches L_W, and let $K = L_M \cap L_W$. Let $\gamma = G_M \wedge G_W$. Then W is a* witness *of M if the following logical implications hold:*

- reset: $\gamma \models (R_M^K \wedge C_M) \rightarrow (R_W^K \wedge C_W)$
- transition: $\gamma \models (F_M^K \wedge C_M \wedge C'_M \wedge C_W) \rightarrow (F_W^K \wedge C'_W)$
- property: $\gamma \models (C_M \wedge C_W) \rightarrow (P_W \rightarrow P_M)$
- base: $\gamma \models (R_W^{L_W} \wedge C_W) \rightarrow P_W$
- step: $\gamma \models (P_W \wedge F_W^{L_W} \wedge C_W \wedge C'_W) \rightarrow P'_W$

This form of witnesses was used as certificates for safety in HWMCC'24 (Froleyks et al. 2025), building on earlier work by the same researchers (Yu, Biere, and Heljanko 2021; Yu et al. 2022, 2023; Froleyks et al. 2024). Like lower-bound certificates in planning, they are sound and complete: a witness W for a circuit M exists iff M is safe.

Unlike the situation in planning, testing if a circuit is a witness for another circuit is *not* a polynomial-time problem (unless P = NP): it is coNP-complete in $\|M\| + \|W\|$, the combined size of the two circuit representations.

In practice, the unsatisfiability checks required for testing the five implications in Def. 10 have turned out to be manageable. Froleyks et al. (2025) report that certificate validation completed within reasonable time bounds for all certificates generated during the competition and usually only took a fraction of the time used by the model checkers. Certificate validation showed that 21 of the safety certificates generated in the competition were invalid, demonstrating the wisdom of not trusting the participating model checkers blindly.

Feedback from the participants showed that producing certificates was not considered an overly arduous requirement. Moreover, even though mandatory certification was a new feature of the competition, there were significantly more participants than in previous editions.

From Lower-Bound Certificates to Witnesses

We finish our discussion of certificates for planning and hardware model checking by showing that lower-bound certificates for planning tasks can be polynomially converted into witnesses for the corresponding circuits.

This result does not follow from our earlier comparison of the *formalisms*: just because bounded-cost plan existence can be modeled as a circuit, it does not follow that any *certificate* mechanism for such circuits can efficiently capture any certificate mechanism for planning. The fact that this is the case here suggests that the concepts underlying witnesses are at least as general as the concepts underlying lower-bound proofs, and suggests that there may be scope for a more general certificate mechanism for planning. We will return to this point in the conclusion.

Because lower-bound certificates include proofs of the properties that need to hold (initial state lemma, goal lemma, inductivity lemma) and witnesses do not, we first introduce a variant of lower-bound certificates without the proofs.

Definition 11 (stripped lower-bound certificate)**.** *Let $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ be a PB circuit for planning task Π and bound B. We say that $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ is a* stripped lower-bound certificate *for Π with bound B if there exists a lower-bound certificate for Π with bound B whose PB circuit is $\langle \mathcal{C}_\varphi, r_\varphi \rangle$.*

We can think of stripped lower-bound certificates as lower-bound certificates from which the proofs of the three lemmas from Def. 9 have been removed. The lemmas still need to *hold*, but the proofs are not part of the certificate, paralleling the definition of witnesses. It is not hard to see that checking if a given PB circuit is a stripped lower-bound certificate is coNP-complete, matching the result for witnesses.

If we are given a PB circuit for a planning task Π and bound B (which may or may not be a stripped lower-bound certificate), we can use it to construct a circuit (which may or may not be a witness) for $M(\Pi, B)$.

Definition 12 (candidate witness)**.** *Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and $B \in \mathbb{N}_0$, and let $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ be a PB circuit for Π and B.*

Let M be an AIG circuit representation of $M(\Pi, B)$, the circuit corresponding to bounded-cost plan existence for task Π and bound B.

Then the candidate witness *corresponding to Π, B and $\langle \mathcal{C}_\varphi, r_\varphi \rangle$, denoted by $W(\Pi, B, \langle \mathcal{C}_\varphi, r_\varphi \rangle)$ is the AIG circuit which is identical to M except that its property encodes r_φ.*

By "identical to M except that its property encodes r_φ", we mean that $W(\Pi, B, \langle \mathcal{C}_\varphi, r_\varphi \rangle)$ has the same inputs, latches, resets, transitions and constraints as M, and it also has all the gates of M with the same gate inputs. Further, it has additional gates that compute r_φ from the variables of the latches. Recall that the variables $V \cup V_c$ over which $\mathcal{C}_\varphi$ is defined are all present as latches in the circuit M. For every reification $r_i \leftrightarrow \varphi_i$ in $\mathcal{C}_\varphi$, we can therefore build an arithmetic circuit that evaluates the pseudo-Boolean constraint φ_i and assigns the result to the new gate r_i. The gates of $W(\Pi, B, \langle \mathcal{C}_\varphi, r_\varphi \rangle)$ are then the original gates of M, the new gates for every variable r_i defined by $\mathcal{C}_\varphi$, plus any auxiliary gates necessary to evaluate the constraints φ_i. This is a polynomial-time construction and results in a circuit which behaves like $M(\Pi, B)$ except that its safety condition is r_φ.

We are now ready to prove our final result: the candidate witness is a witness iff the PB circuit is a stripped lower-bound certificate.

Theorem 1 (compilation of certificates)**.** *Let Π be a planning task, and let $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ be a PB circuit for Π and $B \in \mathbb{N}_0$.*

Then $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ is a stripped lower-bound certificate for Π with bound B iff $W(\Pi, B, \langle \mathcal{C}_\varphi, r_\varphi \rangle)$ is a witness for the AIG circuit representation of $M(\Pi, B)$.

Proof. Let M be the AIG circuit for $M(\Pi, B)$ and $W = W(\Pi, B, \langle \mathcal{C}_\varphi, r_\varphi \rangle)$. We must show that the initial state lemma, goal lemma and inductivity lemma (Def. 9) hold for $\langle \mathcal{C}_\varphi, r_\varphi \rangle$ iff the reset, transition, property, base and step implications (Def. 10) hold for M and W.

M and W have the same latches, which we denote by L. In the notation of Def. 10, we get $L_M = L_W = K = L$. M and W also have the same resets, transitions and constraint, so we can replace R^L_W by R^L_M, F^L_W by F^L_M and C_W by C_M. The implications that must hold for witnesses then become:

- *reset:* $\gamma \models (R^L_M \wedge C_M) \rightarrow (R^L_M \wedge C_M)$
- *transition:* $\gamma \models (F^L_M \wedge C_M \wedge C'_M \wedge C_M) \rightarrow (F^L_M \wedge C'_M)$
- *property:* $\gamma \models (C_M \wedge C_M) \rightarrow (P_W \rightarrow P_M)$
- *base:* $\gamma \models (R^L_M \wedge C_M) \rightarrow P_W$
- *step:* $\gamma \models (P_W \wedge F^L_M \wedge C_M \wedge C'_M) \rightarrow P'_W$

We see that *reset* and *transition* trivialize because the implication goals are tautologies. Every formula of the form $\varphi \rightarrow \varphi$ is a tautology, as in *reset*, and remains so if the left-hand side of "$\rightarrow$" is strengthened, as in *transition*.

We can rewrite the implication goal for *property* as $(C_M \wedge P_W) \rightarrow P_M$: if the constraint and the witness property hold, then the property of M holds. From the definitions of the constraint, witness property and the property of M, this is: "If we have not yet exceeded the cost bound and we select 0 or 1 applicable actions to be applied next (constraint) and r_φ holds (witness property), then we are not in a goal state (property of M)." This is an equivalent reformulation of the goal lemma in Def. 9: "if we are in a goal state and r_φ holds, then we have exceeded the cost bound".

The implication goal for *base* states: if the reset formula for all latches holds (we are in an initial state of M) and the constraint holds, the witness property holds. In an initial state of M, we are in the initial state of the planning task and the accumulated cost so far is 0. The constraint adds the requirement that we have selected 0 or 1 applicable actions, which does not affect the implication. The witness property is r_φ. Therefore, we can write this as: "if we are in an initial state and have not incurred any cost yet, then r_φ holds". This is a direct restatement of the initial state lemma in Def. 9.

Finally, the implication goal for *step* states: if the witness property holds, we take a transition and the constraint holds before and after the transition, then the witness property holds afterwards. In other words, if r_φ holds, we take a transition, we have selected 0 or 1 applicable actions before and after the transition and the incurred cost is within the bound before and after the transition, then r_φ holds afterwards. In the case where we selected 0 actions, this holds trivially because in this case none of the variables referenced by r_φ are changed by the transition. In the case where we selected an applicable action, this is a reformulation of the inductivity lemma in Def. 9. (In particular, r_T in the inductivity lemma ensures that we take an applicable action and that the cost bound is not exceeded before or after the transition.) This concludes the proof. □

The theorem shows that any argument that can be expressed as a lower-bound certificate can also be expressed as a witness circuit with a polynomial-time compilation. In this sense, witness circuits are at least as powerful as lower-bound certificates. Moreover, the compilation does not require the full power of witness circuits (such as using a different set of latches or different transition function), as it is sufficient to consider witnesses that only differ from the original circuit in their property. This suggests that, in their full generality, witness circuits might be strictly more expressive than lower-bound certificates.

Conclusion

Starting from the observation that there are close conceptual similarities between classical planning and hardware model checking, we studied the question of how the *absence* of paths with certain properties in transition systems can be supported by certificates in the two formalisms.

Lower-bound certificates in planning use pseudo-Boolean circuits to encode an invariant property. Witness circuits in hardware model checking also include an invariant property, and together with a reduction from planning tasks to circuits, this allows us to express any lower-bound certificate as a witness circuit with a polynomial compilation.

However, witness circuits have additional bells and whistles that we did not need in our compilation. For example, they can introduce additional latches, for which the planning equivalent would be new state variables with their own transition semantics, "derived variables" evolving over time that can capture information about paths rather than just states.

This leads to the question: would it be useful to extend lower-bound certificates with features that are currently missing compared to the hardware model checking world? And could this potentially let us certify planning techniques that have so far proved resistant to efficient certification, such as partial-order reduction (e.g., Wehrle and Helmert 2012; Röger et al. 2020)? Or, conversely, can we show that some of the additional features of witness circuits do *not* increase expressiveness and are essentially syntactic sugar?

On the practical side, an advantage of lower-bound certificates over witnesses is that they are verifiable in polynomial time because they include the proofs for the initial state, goal and inductivity lemmas, which in witness circuits have to be rediscovered by the certificate validator. Can we take our compilation to hardware model checking one step further and also compile these *proofs*, so that the witness validator no longer needs a SAT solver but can directly use an UNSAT proof checker, thus also making it polynomial? Or, conversely, are state-of-the-art SAT solvers efficient enough that stripped lower-bound certificates are all we need?

In conclusion, we see that the connections we have made in this paper open up several interesting new research questions, which we look forward to investigating in the future.

Acknowledgments

This work was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Balyo, T.; and Suda, M. 2016. Reachlunch Entering The Unsolvability IPC 2016. In *Unsolvability IPC: Planner Abstracts*, 3–5.

Biere, A.; Heljanko, K.; and Wieringa, S. 2011. AIGER 1.9 And Beyond. Technical Report 11/2, Johannes Kepler University, Institute for Formal Models and Verification.

Bjørner, N. S.; Heule, M. J. H.; Kaufmann, D.; Nordström, J.; and Koops, W. 2025. Certifying Algorithms for Automated Reasoning (Dagstuhl Seminar 25231). *Dagstuhl Reports*, 15(6): 1–31.

Bogaerts, B.; Gocht, S.; McCreesh, C.; and Nordström, J. 2023. Certified Dominance and Symmetry Breaking for Combinatorial Optimisation. *JAIR*, 77: 1539–1589.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *AIJ*, 129(1): 5–33.

Buss, S.; and Nordström, J. 2021. Proof Complexity and SAT Solving. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 233–350. IOS Press.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *AIJ*, 69(1–2): 165–204.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AIJ*, 147: 35–84.

Dold, S.; Helmert, M.; Nordström, J.; Röger, G.; and Schindler, T. 2025. Pseudo-Boolean Proof Logging for Optimal Classical Planning. In *Proc. ICAPS 2025*, 54–63.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proc. ECP 2001*, 84–90.

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

Eriksson, S.; and Helmert, M. 2020. Certified Unsolvability for SAT Planning with Property Directed Reachability. In *Proc. ICAPS 2020*, 90–100.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability Certificates for Classical Planning. In *Proc. ICAPS 2017*, 88–97.

Eriksson, S.; Röger, G.; and Helmert, M. 2018. A Proof System for Unsolvable Planning Tasks. In *Proc. ICAPS 2018*, 65–73.

Froleyks, N. 2024. Certifaiger. https://github.com/Froleyks/certifaiger. Accessed December 8, 2025.

Froleyks, N.; Yu, E.; Biere, A.; and Heljanko, K. 2024. Certifying Phase Abstraction. In *Proc. IJCAR 2024, Part I*, 284–303.

Froleyks, N.; Yu, E.; Preiner, M.; Biere, A.; and Heljanko, K. 2025. Introducing Certificates to the Hardware Model Checking Competition. In *Proc. CAV 2025, Part I*, 281–295.

Gerevini, A. E.; and Long, D. 2005. Plan Constraints and Preferences in PDDL3. Technical Report R. T. 2005-08-47, University of Brescia, Department of Electronics for Automation.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *Proc. ICTAI 2004*, 294–301.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying algorithms. *Computer Science Review*, 5(2): 119–162.

Mishchenko, A.; Chatterjee, S.; Jiang, R.; and Brayton, R. K. 2005. FRAIGs: A Unifying Representation for Logic Synthesis and Verification. ERL technical report, UC Berkeley.

Mugdan, E.; Christen, R.; and Eriksson, S. 2023. Optimality Certificates for Classical Planning. In *Proc. ICAPS 2023*, 286–294.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *JAIR*, 12: 271–315.

Pednault, E. P. D. 1989. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In *Proc. KR 1989*, 324–332.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Röger, G.; Helmert, M.; Seipp, J.; and Sievers, S. 2020. An Atom-Centric Perspective on Stubborn Sets. In *Proc. SoCS 2020*, 57–65.

Sistla, A. P.; and Clarke, E. M. 1985. The Complexity of Propositional Linear Temporal Logics. *JACM*, 32(3): 733–749.

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

Wehrle, M.; and Helmert, M. 2012. About Partial Order Reduction in Planning and Computer Aided Verification. In *Proc. ICAPS 2012*, 297–305.

Wetzler, N.; Heule, M.; and Hunt, W. A., Jr. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *Proc. SAT 2014*, 422–429.

Yu, E.; Biere, A.; and Heljanko, K. 2021. Progress in Certifying Hardware Model Checking Results. In *Proc. CAV 2021, Part II*, 363–386.

Yu, E.; Froleyks, N.; Biere, A.; and Heljanko, K. 2022. Stratified Certification for K-Induction. In *Proc. FMCAD 2022*, 59–64.

Yu, E.; Froleyks, N.; Biere, A.; and Heljanko, K. 2023. Towards Compositional Hardware Model Checking Certification. In *Proc. FMCAD 2023*, 1–11.