Verified Certification of Unsolvability of Temporal Planning Problems

David Wang and Mohammad Abdulaziz

King's College London, London, UK
{mohammad.abdulaziz,david.wang}@kcl.ac.uk

Abstract. We present an approach to unsolvability certification of temporal planning. Our approach is based on encoding the planning problem into a model checking problem of a network of timed automata. We then use an efficient model checker to check the network followed by a certificate checker to certify the output of the model checker. We formally verify our implementation of the encoding to timed automata using the theorem prover Isabelle/HOL, use an existing certificate checker (also formally verified in Isabelle/HOL), and use an unverified, optimised model checker. This presents a compromise between trustworthiness of the certification, performance of the certification, and ease of implementation.

Keywords: Temporal Planning · Certification · Formal Verification.

1 Introduction

Automated planning is a mature area of AI research, where great strides have been achieved in the performance of planning systems as well as their ability to process rich descriptions of planning problems. This can be most evidently seen in the improvements in performance and expressiveness through the different planning competitions, which started in 1998 [6, 9, 15–17], and by the fact that state-of-the-art planning systems scale to real-world sequential decision problems.

Despite the great performance of current planning systems and the expressiveness of their input formalisms, more could be achieved to improve the trustworthiness of their outputs. This is of great importance given the high complexity of state-of-the-art planning algorithms and software, both at an abstract algorithmic level and at the implementation optimisation level. Indeed, a lot of work went already into improving the trustworthiness of planning systems and software. Much of that work has been put into developing *certificate checkers* for planning, which are (relatively) small, independent pieces of software that can check the correctness of the output of a planner. This includes the development of plan validators [11, 14], which are programs that, given a planning problem and a candidate solution, confirm that the candidate solution indeed solves the planning problem. Furthermore, to improve the trustworthiness of these validators, other authors have formally verified them, i.e. mathematically proved that those validators and their implementations are correct w.r.t. a semantics of the respective planning formalism. This was done for a validator for classical planning [3], which is the simplest planning formalism, and another for temporal planning [1], which is a richer planning formalism wielding a notion of action duration w.r.t. a dense timeline and that allows for concurrent plan action execution.

Improving trustworthiness of the correctness of a planner's output in case it reports that a planning problem is unsolvable or that a computed plan is optimal is much harder, nonetheless. In the case of solvability, the computed plan, which is in most practical cases succinct, is a certificate that can be executed by the planner. In the cases of unsolvability or optimality claims, obtaining a reasonably compact certificate substantiating the planner's output is much harder: in the worst-case, unless NP = PSPACE [4, Chapter 4], such a certificate can be exponentially long in terms of the planning task's size for most relevant planning formalisms. In practice, such certificates are usually exponentially long, unless very carefully designed, like those by Eriksson et al. [7], who devised unsolvability certification schemes for classical planning which can be succinct for large classes of problems.

In this work, we consider the problem of certifying unsolvability of temporal planning, i.e. planning with durative actions [8]. In general, devising a practical unsolvability certification scheme is not always possible as it may require changes to the planning algorithm to make it more likely to produce succinct certificates. This is particularly difficult in the case of state-of-the-art algorithms for temporal planning due to their technical complexity. The other approach (see Figure 1), which we follow here, is to encode the respective planning problem into another type of computational problem for which there already exists a practical certifying algorithm. A challenge with this approach's trustworthiness, however, is that the encoding procedure could be of high complexity, meaning that, although the output of the target problem's solver is certified, the certification's implication for the source problem depends on the correctness of the encoding and its implementation. To resolve that, the encoding, its implementation, and the certificate checker are formally verified using a theorem prover. This approach was devised by Abdulaziz and Kurz [2], who used it to develop a certifying SAT-based planning system.

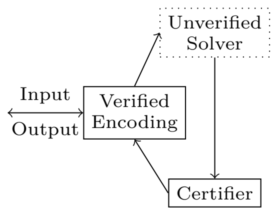
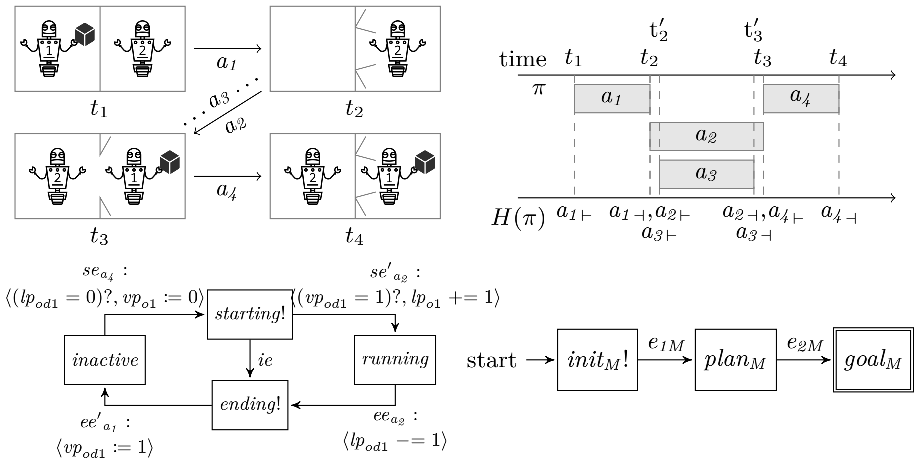


Fig. 1: The verified planner architecture proposed by Abdulaziz and Kurz, combining verified (solid) and unverified (dotted) modules.

Our main contribution is that we use this approach, where we formally verify an encoding of temporal planning into timed automata by Heinz et al. [12] and use a verified certificate checker for timed automata model checking by Wimmer and von Mutius [18]. Doing that successfully requires substantial engineering of the implementation and its correctness proof.



a_1
a_3
a_2
a_4
t_1
t_2
t_3
t_4
time
t'_2
t'_3
π
$H(\pi)$
$a_{1\vdash}$
$a_{1\dashv}, a_{2\vdash}$
$a_{3\vdash}$
$a_{2\dashv}, a_{4\vdash}$
$a_{3\dashv}$
$a_{4\dashv}$
se_{a_4} : $\langle (lp_{od1} = 0)?, vp_{o1} := 0\rangle$
se'_{a_2} : $\langle (vp_{od1} = 1)?, lp_{o1} \mathrel{+}= 1\rangle$
inactive
starting!
running
ending!
ie
ee'_{a_1} : $\langle vp_{od1} := 1\rangle$
ee_{a_2} : $\langle lp_{od1} \mathrel{-}= 1\rangle$
start
$init_M$!
e_{1M}
$plan_M$
e_{2M}
$goal_M$

Fig. 2: A figure demonstrating a planning problem and its encoding as a network of timed automata. The planning problem models two robots (rb_1 and rb_2), two rooms (rm_1 and rm_2), and four actions ($a_1 \dots a_4$): a_1/a_4 open/close the door (d_1), and a_2/a_3 move rb_1/rb_2 from rm_1/rm_2 to rm_2/rm_1. Top left: States induced by a plan. rb_2 opens the door in the interval (t_1, t_2). rb_1 and rb_2 move from rm_1/rm_2 to rm_2/rm_1 in the interval (t_2, t_3). The start of a_2 deletes $in(rb_1, rm_1)$. The start of a_1/a_4 deletes $closed(d_1)/open(d_1)$. Top right: Timeline of actions. The end of a_1 ($a_{1\dashv}$) sets $open(d_1)$ to True at t_2. The start of a_2 ($a_{2\vdash}$) is scheduled for t_2. $a_{1\dashv}$ and $a_{2\vdash}$'s effects are applied simultaneously. $a_{4\vdash}$ is scheduled concurrently with $a_{2\dashv}$. $a_{3\vdash}$ and $a_{3\dashv}$ must be separated from $a_{1\dashv}$ and $a_{4\dashv}$ by time, because they all modify the $idle(rb_2)$ proposition. Bottom left: An abstract state-space of an automaton $\mathcal{A}_{a_i}$ to encode an action a_i. Transitions of some $\mathcal{A}_{a_i}$ are labelled. Binary variables vp and locking counters lp for propositions p encode propositional states and invariants respectively. vp_{od1} and lp_{od1} belong to $open(d_1)$. Bottom right: A main automaton ($\mathcal{A}_M$). e_{1M} initialises variables. e_{2M} checks that propositional variable assignments satisfy a goal.

Availability A more complete description[1] of the project as well as the formalisation itself[2] can be found online.

2 A Verified Encoding of Temporal Planning

We formally verify the correctness of our certification approach using the theorem prover Isabelle/HOL. This is done by verifying one direction (*if a plan exists for the given problem then a trace in the network of timed automata satisfying the formula exists*) of the correctness of our encoding, which is based on an encoding presented by Heinz et al. [12].

[1] https://arxiv.org/abs/2510.10189
[2] https://doi.org/10.5281/zenodo.17354028

A *durative action*, a, has two operators, $a_\vdash$ and $a_\dashv$, which are applied when the action is scheduled to start and end respectively, and an *invariant* condition which must hold throughout the action's execution. The operators have *preconditions* and *effects*. The goal of planning is to find a schedule of actions leading to a *goal* state. A *network of timed automata* augments multiple simultaneously running finite automata with clock variables, which record the passage of time. Time cannot pass in *urgent* locations marked with (!).

The characterisation of all abstract states of the network encountered by a trace simulating the updates in an instant and the proof that the conditions of transitions are satisfied by the abstract states to which they are applied are essential components of the larger proof. Unlike in temporal planning, the state in one instant cannot be updated in a single operation in the network. Instead, the changes to the states of each action and the effects of the action on the world are applied in sequence. For instance, when simulating the transition to the state at t_2 in the upper left of Figure 2, the network must encounter an abstract state in which rb_1 is still present in rm_1, whereas this state is not encountered in the state sequence of the plan.

We modify Heinz et al.'s [12] reduction by making $starting_a!$ and $ending_a!$, for any action a, urgent and permitting any action's automaton to enter and leave said locations in the same instant, if the simulated operators (i.e. the start $a_\vdash$ and end $a_\dashv$) do not *interfere* with other simulated operators in the same instant. *Interference* (i.e. contradictory preconditions or effects) of mutually exclusive operators is avoided by preventing their applications to be simulated in the same instant using clock constraints in transitions like Gigante et al. [10] do.

We use the idea of binary variables vp_p and locking variables lp_p for propositions p from Heinz et al.'s [12] encoding. As seen in the bottom left of Figure 2, transitions that decrement and increment lp_{od1} must occur before and after those that can modify vp_{od1} respectively. Thus, the order in which abstract states are visited to simulate an instant is important; a detail that is present in the proof.

We formalise a notion of temporal planning problems and use an existing formalisation of a notion of timed automata model checking [18]. All definitions are formalised as data types and functions in Isabelle/HOL's logic. The encoding is formalised as a functional program operating on these data types.

We formally prove that the encoding is complete. We define a formula Φ, which asserts that a run exists, in which $\mathcal{A}_M$ eventually reaches $goal_M$. We prove that Φ is satisfied by the network $\mathcal{A}$ if a plan π exists (Listing 1.1 shows the formal theorem statement of the contrapositive in Isabelle/HOL). Wimmer and von Mutius [18] have formally verified a certificate checker which can certify that $\mathcal{A}$ does not satisfy Φ using certificates computed by an efficient unverified model checker. This thus completes our own certificate checker.

Listing 1.1: Formal theorem statement in Isabelle/HOL

```
corollary form_not_sat_imp_no_valid_plan:
  assumes "¬(ref_model_checking.net_impl.sem,
      ref_model_checking.a₀ ⊨ reduction_ref_impl.
      formula_spec)"
```

```
shows "¬(∃π::(nat, 'action, int) temp_plan.
    temp_plan_for_problem_list_impl_int' at_start at_end
     over_all lower upper pre adds dels init goal ϵ
    props actions π)"
using assms valid_plan_imp_form_holds by auto
```

In addition to providing formal correctness guarantees, our formalisation relaxes some conditions to support a more general definition of plan validity. For example, by using clock constraints like Gigante et al. [10], as opposed to a locking automaton like Bogomolov et al. [5] and Heinz et al. [12], we admit plans with multiple actions starting and ending in an instant. As a drawback, some transitions must query the value of a number of clock variables, which scales linearly with the number of actions. We also add the *ie* transition, which we conjecture to precisely simulate the execution of an action schedule with a duration of 0.

Discussion and Future Work The closest piece of work to what we present here is the verified encoding of *classical planning (STRIPS)* to propositional satisfiability problems (*SAT*) by Abdulaziz and Kurz [2]. In comparison, the verification of our encoding is substantially more complex as temporal planning is more expressive than STRIPS planning, due to the inclusion of a timeline and concurrency. Because of that, the size of our formal proof of correctness of the abstract reduction (in terms of number lines of code) is 19K, which is the same as encoding of STRIPS to SAT by Abdulaziz and Kurz. Note, however, that our encoding is not executable, while that of Abdulaziz and Kurz is. We plan to add executability to our development and we expect it to be a substantial engineering effort (e.g. in Abdulaziz and Kurz's and other verified algorithms projects, it represents almost 40% of the effort).

Our reduction currently takes grounded temporal planning problems as input. The semantics of those ground problems closely correspond to a subset of those supported by the verified plan validator by Abdulaziz and Koller. We will prove this equivalence formally. Another step of future work is making our certification mechanism executable. This will involve two things. First, at an engineering level, we will need to prove the current certification mechanism equivalent to an executable certificate checker. Second, for full guarantees, we will need to formally verify a grounder, which will not be straightforward, if we are to use a scalable grounding algorithm, e.g. the one by Helmert [13], as it will involve verifying (or certifying the output of) a datalog solver.

Bibliography

[1] Abdulaziz, M., Koller, L.: Formal semantics and formally verified validation for temporal planning. In: The 36th AAAI Conference on Artificial Intelligence (AAAI) (2022), https://doi.org/10.1609/aaai.v36i9.21197

[2] Abdulaziz, M., Kurz, F.: Formally verified SAT-Based AI planning. In: The 37th AAAI Conference on Artificial Intelligence (AAAI) (2023), https://doi.org/10.1609/aaai.v37i12.26714

[3] Abdulaziz, M., Lammich, P.: A Formally Verified Validator for Classical Planning Problems and Solutions. In: The 30th International Conference on Tools with Artificial Intelligence (ICTAI) (2018), https://doi.org/10.1109/ICTAI.2018.00079

[4] Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, 1 edn. (2009), https://doi.org/10.1017/CBO9780511804090

[5] Bogomolov, S., Magazzeni, D., Podelski, A., Wehrle, M.: Planning as model checking in hybrid domains. In: Proceedings of the National Conference on Artificial Intelligence, vol. 3 (2014), ISSN 2159-5399, https://doi.org/10.1609/aaai.v28i1.9037

[6] Coles, A.J., Coles, A., Olaya, A.G., Celorrio, S.J., López, C.L., Sanner, S., Yoon, S.: A Survey of the Seventh International Planning Competition. AI Mag. (2012), https://doi.org/10.1609/aimag.v33i1.2392

[7] Eriksson, S., Röger, G., Helmert, M.: Unsolvability Certificates for Classical Planning. In: The 27th International Conference on Automated Planning and Scheduling (ICAPS) (2017), URL https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15734

[8] Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research (2003), URL http://arxiv.org/abs/1106.4561

[9] Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. AIJ (2009)

[10] Gigante, N., Micheli, A., Montanari, A., Scala, E.: Decidability and complexity of action-based temporal planning over dense time. Artificial Intelligence **307** (2022), ISSN 00043702, https://doi.org/10.1016/j.artint.2022.103686

[11] Haslum, P.: Patrikhaslum/INVAL (2024), URL https://github.com/patrikhaslum/INVAL

[12] Heinz, A., Wehrle, M., Bogomolov, S., Magazzeni, D., Greitschus, M., Podelski, A.: Temporal Planning as Refinement-Based Model Checking. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019 (2019), URL https://ojs.aaai.org/index.php/ICAPS/article/view/3476

[13] Helmert, M.: The Fast Downward Planning System. J. Artif. Intell. Res. (2006), https://doi.org/10.1613/jair.1705

[14] Howey, R., Long, D., Fox, M.: VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI) (2004), https://doi.org/10.1109/ICTAI.2004.120

[15] McDermott, D.V.: The 1998 AI Planning Systems Competition. AI Mag. (2000), https://doi.org/10.1609/aimag.v21i2.1506

[16] Taitler, A., Alford, R., Espasa, J., Behnke, G., Fi$v$ser, D., Gimelfarb, M., Pommerening, F., Sanner, S., Scala, E., Schreiber, D., Segovia-Aguas, J., Seipp, J.: The 2023 International Planning Competition. AI Magazine (2024), https://doi.org/10.1002/aaai.12169

[17] Vallati, M., Chrpa, L., Grzes, M., McCluskey, T.L., Roberts, M., Sanner, S.: The 2014 International Planning Competition: Progress and Trends. AI Mag. (2015), https://doi.org/10.1609/aimag.v36i3.2571

[18] Wimmer, S., von Mutius, J.: Verified Certification of Reachability Checking for Timed Automata. In: 26th International Conference on the Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2020), https://doi.org/10.1007/978-3-030-45190-5_24