Goal-constrained planning domain model verification of safety properties*

Anas Shrinah and Kerstin Eder
University of Bristol
first.last@bristol.ac.uk

Abstract

The verification of planning domain models is crucial to ensure the safety, integrity and correctness of planning-based automated systems. This task is usually performed using model checking techniques. However, unconstrained application of model checkers to verify planning domain models can result in false positives, i.e. counterexamples that are unreachable by a sound planner when using the domain under verification during a planning task. In this paper, we discuss the downside of unconstrained planning domain model verification. We then introduce the notion of a valid planning counterexample, and demonstrate how model checkers, as well as state trajectory constraints planning techniques, should be used to verify planning domain models so that invalid planning counterexamples are not returned.

1 Introduction

Planning and task scheduling techniques are increasingly applied to real-world problems such as activity sequencing, constraint solving and resource management. Since the failure of such systems could be devastating, verification methods that are robust, trustworthy and systematic are crucial to gain confidence in the safety, integrity and correctness of these systems.

A typical planning system consists of a planning domain model, planning problem, planner, plan, executive, and monitor. Planners take as an input a domain model which describes application-specific states and actions, and a problem that specifies the planning goal and the initial state. From these inputs, a sequence of actions that can achieve the goal starting from the initial state is returned as plan. The plan is then executed by an executive to change the world state so that it matches the desired goal.

Our research focuses on the verification of planning domain models wrt. safety properties. Due to modelling errors, a domain model might be inconsistent, incomplete, or inaccurate [BHO14]. This could cause the planner to fail in finding a plan or to generate unrealistic plans that will fail to execute in the real-world. Moreover, erroneous domain models could lead planners to produce unsafe plans that, when executed, could have catastrophic consequences in the real world.

This paper addresses the fact that the state-of-the-art verification methods for planning domain models are vulnerable to false positive counterexamples. In particular, unconstrained verification tasks might return counterexamples that are unreachable by planners when using the domain under verification (DUV) during a planning task. Such counterexamples can mislead designers to unnecessarily restrict domain models, thereby potentially blocking valid and possibly necessary behaviours.

* Supported by EPSRC grant EP/P510427/1 in collaboration with Schlumberger.

In: Rudolph S., Marreiros G. (eds.): Proceedings of STAIRS, Santiago de Compostela, Spain, august 2020, published at http://ceur-ws.org

To address this shortfall, we propose to use planning goals as constraints during verification, a concept transferred from verification and validation research [D+04]. Thus, we introduce *goal-constrained planning domain model verification*, an approach that eliminates invalid planning counterexamples per se.

We formally prove that goal-constrained planning domain model verification of safety properties is guaranteed to return only valid planning counterexamples, if and only if any exist. We illustrate our method using the Cave Diving planning domain model [IPC14]. Additionally, we perform empirical experiments to demonstrate the feasibility and investigate the behaviour of our approach using the Spin model checker [Hol04] and the MIPS-XXL-IPC5 planner [EJN06].

The rest of this paper is organised as follows. Section 2 informally discusses the problem of invalid planning counterexamples in planning domain model verification. A verification concept of planning domain models that avoids returning invalid planning counterexamples is presented in Section 3. Section 4 reports and discusses the experimental results. Section 5 contrasts the concepts presented here with related work. Finally, Section 6 concludes the paper and suggests future work.

2 Invalid counterexamples in planning domain model verification

Planning domain model verification aims to demonstrate that any produced plan satisfies a set of properties. To achieve this, formal planning domain model verification methods leave the planning goal open. This, we define as *unconstrained* verification of planning domain models, i.e. the verification is expected to hold for any potential goal. Unconstrained verification searches the domain model for a sequence of actions that can falsify the given property, regardless of any other conditions. In particular, whether or not a planner would consider this sequence to be a plan, is not taken into account. This is a critical oversight, because, when the domain model is used to solve a specific planning problem, the sequence of actions that constitutes such a counterexample might, in fact, be "pruned away" by the planner, if it does not satisfy the planning goal. Hence, we consider them to be invalid planning counterexamples.

As an example, we consider the classical cave diving planning domain taken from the Satisficing Track of the IPC-2014 [IPC14]. The problem consists of an underwater cave system with a finite number of partially interconnected locations. Divers can enter the cave from its entrance and swim from one location to an adjacent connected one. They can hold up to four oxygen tanks and they consume one for every swim and take-photo action. Divers have to perform a decompression manoeuvre to go to the surface and this can be done only at the entrance L_0. Additionally, divers can drop tanks in or take tanks from any location if they hold at least one tank or there is one tank available at the location, respectively.

The planning goals of this domain specify the required underwater locations of which photos are to be taken and mandate the divers must return to the surface after completing the mission.

A critical safety property, p, is that divers should not drown, i.e. they should not be in an underwater location, other than the entrance, where neither the divers nor the location have at least one full oxygen tank.

In this example, the chosen planning goal is to have a photo of the first location, L_1, and to get the diver outside the water. In unconstrained verification with only the safety property p, a counterexample could be $\langle$*prepare-a-tank, enter-water, swim*$(L_0, L_1)\rangle$. Indeed, this counterexample leads the diver to a drowning state. At the end of this sequence, the diver will have consumed their oxygen tank and will be in underwater location L_1. This is not the entrance, so they can not surface and they do not have an oxygen tank to swim back to the entrance. However, this is not a plan because it does not achieve any useful goal. Therefore, it will not be produced by any sound planner when it is used in a practical scenario (taking a photo of any location). Though this counterexample is obviously unreachable and should not misguide the designers to overcomplicate the model in an attempt to remove it, in a real-world sized application such invalid planning counterexamples can be much more difficult to recognise as such.

To overcome this, we observe that planning is performed for a specific goal. To exploit this observation for domain model verification, we propose to use the goal given to the planner as a constraint to ensure that the counterexamples returned by a model checker, or other tools used in this context, falsify the given safety property while also achieving the planning goal. The details of this method are further explained in the next section.

3 Goal-constrained verification of planning domain models

Goal-constrained verification can be performed using advanced search algorithms, such as model checkers or classical planners, to find a valid counterexample for the given safety property, if one exists.

We define a valid planning counterexample to be a sequence of actions that, firstly, can falsify the given safety property, secondly, can achieve the planning goal from the given initial state, and, thirdly, none of the sub-sequences of the counterexample can achieve the goal.

Formally, this is defined as follows: Let the planning problem, P, be a tuple, (D, s_0, g), where D is the domain model that describes the set of all available actions, A, s_0 is the initial state and g is the desired goal. The plan π is a solution to the planning problem P, defined as a sequence of actions, $\pi = \langle a_0, a_1, ..., a_n \rangle$. These actions are chosen from A, $a_i \in A$, such that $\pi \models g$. In other words, when π is applied to the initial state s_0 it yields a sequence of states $S(\pi)$, $S(\pi) = \langle s_0, s_1, ..., s_n \rangle$ where the last state s_n satisfies the planning goal g, $s_n \models g$. We say a plan π satisfies a property p, $\pi \models p$, if the sequence of states $S(\pi)$, generated by the plan π, satisfies the property p, $S(\pi) \models p$.

Furthermore, as defined in [GNT04], we call a plan π a *redundant plan*, if π contains a subsequence, π', $\pi' \mid \pi$, that achieves the goal g.

Definition 1: A **valid planning counterexample** for a safety property, p, of a planning problem is a *non-redundant plan*, π, that falsifies the safety property, $\pi \not\models p$.

Plans are required to be non-redundant in the definition of valid planning counterexamples to exclude any plans that are enriched with action sequences which are unnecessary to achieve the planning goal but required to falsify the given safety property. Since sound planners can produce valid plans that have redundant subsequences, the scope of our method is limited to non-redundant planners, i.e. planners that are guaranteed to produce non-redundant plans.

We note that none of the current planning domain model verification methods verify planning domain models in their generality. Firstly, current methods require a grounded model, which represents a finite set of planning problems, in contrast to the infinite set of planning problems that non-ground domain models represent. Secondly, all methods need a specific initial state to be able to perform the verification tasks. In our approach, by using the planning goal as one further constraint, we perform verification of single planning instances. This restriction is the cost associated with delivering a verification method that is robust against invalid planning counterexamples.

In the following subsections, we demonstrate how this concept can be realised using model checkers and state trajectory constraints planning techniques.

3.1 Goal-constrained planning domain verification using model checkers

Model checkers verify safety properties by searching for counterexamples that falsify those properties. In the case of planning applications, any sequence of actions that does not achieve the given goal, will be pruned by any sound planner. Therefore, in planning domain verification, any counterexample that does not achieve the given planning goal should be eliminated on the basis that this counterexample is unreachable by the planner.

To force model checkers to only return valid planning counterexamples, the safety property is first negated and then joined with the planning goal in a conjunction. This conjunction is then negated and supplied to the model checker as an input property. The final property requires the model checker to find a counterexample that both, falsifies the safety property and satisfies the planning goal. Note that, unlike Def. 1, this permits sequences that falsify the property after satisfying the goal. To eliminate these sequences, model transitions should be augmented with an additional guard, representing the negation of the goal, to restrict all transitions once the goal is achieved. With this modification, the model checker is forced to return counterexamples that falsify the safety property before achieving the goal.

For a verification problem, $V = (D, (s_0, g), p)$, we first check whether the planning goal is achievable, then we translate the domain model D into the model checker's input language, obtaining the model M that incorporates the initial state s_0. Then, the model M is modified to M' by augmenting the guards of all transitions with the negation of the goal condition.

From the definition of M', we can derive two properties: First, ***P1****: all plans generated from M are also plans that can be generated from M'.* Proof: any sequence of transitions from M that ends with a transition that achieves the goal is also a sequence of transitions from M'. These sequences represent valid plans as they terminate with a transition that achieves the goal. Therefore, all plans generated from M are also plans in M'. Second, ***P2****: any valid counterexample for M' is also a valid counterexample for M.* Proof: as M' is a more constrained version of M, the set of all legal transition of M', $\Pi(M')$, is contained in the set of all legal transitions of M, $\Pi(M)$, i.e. $\Pi(M) \supseteq \Pi(M')$. It follows that any valid counterexample in $\Pi(M')$ is also in $\Pi(M)$.

The model checker is then applied to the verification problem $V'' = (M', p')$, where p' is defined using F, the

eventually operator from Linear Temporal Logic (LTL) [CGP99], as follows:

$$p' = \neg\big(F(\neg p) \wedge F(g)\big) \tag{1}$$

There are two possible outcomes. First, if the model checker returns a counterexample π:

$$\exists \pi \in \Pi(M').\pi \not\models p' \equiv \ \exists \pi \in \Pi(M').\pi \models (F(\neg p) \wedge F(g)) \tag{2}$$

From the definition of the LTL *eventually* operator F:

$$\exists i \geq 0,\ s_i \in S(\pi), s_i \models \neg p \tag{3}$$

$$\exists j \geq 0,\ s_j \in S(\pi), s_j \models g \tag{4}$$

It follows that there is at least one sequence $S(\pi)$ that falsifies the property p, and there is a state s_j in that sequence which satisfies the goal g, according to (3) and (4). In addition to that, in the sequence $S(\pi)$, p is guaranteed to be falsified before g is satisfied. This is because $\pi \in \Pi(M')$ and M' is constrained to not produce any transitions after achieving the goal. Thus, the counterexample π is a valid planning counterexample in M' for the original safety property p as per Def. 1. Furthermore, from (***P2***), π is also a counterexample in M. This proves that the DUV does not satisfy the safety property p *with respect to the goal* g.

The other potential outcome is that the model checker fails to find a counterexample, then $\forall \pi \in \Pi(M')$:

$$\pi \models p' \ \equiv \ \pi \models \neg(F(g) \wedge F(\neg p)) \ \equiv \ \pi \models (\neg F(g) \vee \neg F(\neg p)) \tag{5}$$

$$\equiv \ \pi \models \neg F(g) \vee \pi \models \neg F(\neg p) \ \equiv \ \pi \not\models F(g) \vee \pi \models G(p) \tag{6}$$

$$\equiv \ \pi \models F(g) \Rightarrow \pi \models G(p) \tag{7}$$

where G is the LTL *always* operator. This means p is always true for any sequence of actions in M' that achieves the goal, i.e. for all possible plans. Since from (***P1***) all plans generated from M are also plans in M', and from (7) all plans in M' are safe, we can conclude that all plans in M are safe. This proves that the DUV satisfies the original property *with respect to the goal*.

3.2 Goal-constrained planning domain verification using planning techniques with state trajectory constraints

Domain models can be verified to only produce safe plans, in terms of satisfying a given safety property, for a specific goal using planners that use breadth first search. This is achieved by consulting a planner over the DUV to produce a plan that can satisfy the goal and the negation of the property. If the domain model permits producing plans that, along with achieving the goal, contradict the safety property, then an unsafe plan can be found. Thus, the returned plan is a counterexample.

For a verification problem, $V = (D, (s_0, g), p)$, the safety property, p, is negated and expressed in terms of PDDL3.0 modal operators as shown in [GHL$^+$09]. The result is added as a state trajectory constraint to the original domain model.

Using the algorithm proposed in [EJN06], the new model is transformed into a PDDL2 compatible version. This is performed by first translating the state trajectory constraint into a non-deterministic finite state automaton (NFA). The NFA which can capture property violations is then embodied in the model in terms of additional predicates and conditional effects. These additions observe the behaviour of the automaton that represents the constraint and stop goal satisfaction unless the constraint is satisfied too.

This yields a new planning problem, $P' = (D', s'_0, g')$, where $D',\ s'_0,\ g'$ are modified instances of $D,\ s_0,\ g$ that are supplemented with the additional predicates and conditional effects of the automaton that represents the introduced constraint. Let the set of legal sequences of actions that can be generated from D be $\Pi(D)$ and from D' be $\Pi(D')$. Note that D' is an augmented version of D and the additions to D' do not affect the number of original operators, their preconditions, or their effects. Furthermore, the additional conditional effects do not affect the original predicates. Hence, $\Pi(D) = \Pi(D')$.

Then, a planner is applied to P' with two possible outcomes. First, if the planner finds a plan π then: $\exists \pi \in \Pi(D').\ \pi \models g'$. Since the satisfaction of g' implies both, the satisfaction of the original goal g at the last state of the sequence $S(\pi)$, and the satisfaction of the state trajectory constraint (the negation of the safety property) by the sequence $S(\pi)$: $\exists \pi \in \Pi(D').\ (\pi \models g \wedge \pi \models \neg p)$. Furthermore, since $\Pi(D) = \Pi(D')$:

$$\exists \pi \in \Pi(D).\ (\pi \models g \wedge \pi \models \neg p) \tag{8}$$

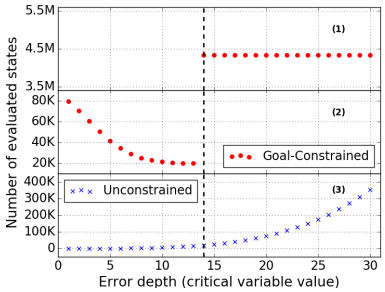
Furthermore, from (8) it follows that $\pi \not\models p$, confirming that there is at least one sequence of actions from D

that achieves the goal while not respecting the safety property. Therefore, this sequence is a valid planning counterexample for that property as per Def. 1. Hence, the DUV does not satisfy the property *wrt. the planning goal.* Alternatively, if the planner fails to find a plan, then, as opposed to (8), we have:

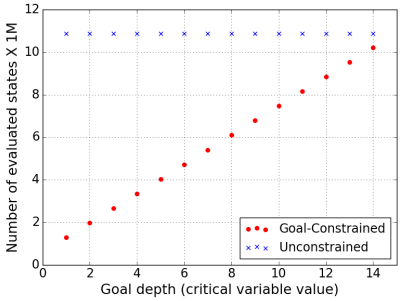
$$\nexists \pi \in \Pi(D).\ (\pi \models g \wedge \pi \models \neg p) \equiv \forall \pi \in \Pi(D).\ (\pi \not\models g \vee \pi \not\models \neg p) \tag{9}$$

$$\equiv \forall \pi \in \Pi(D).\ (\pi \models g \Rightarrow \pi \models p) \tag{10}$$

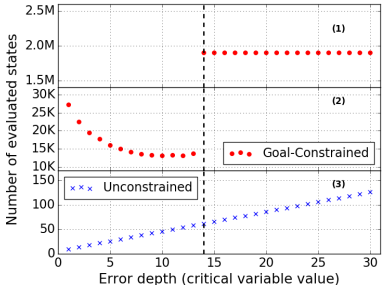
Hence, any sequence of actions from D that achieves the goal also satisfies the safety property. Therefore, the property holds for the planning domain model *wrt. the given goal.*



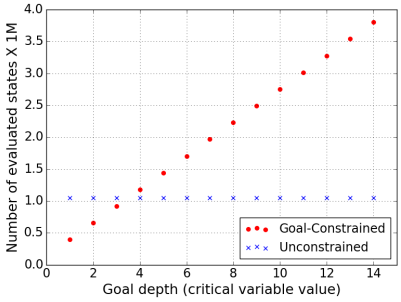
(a) Evaluated states vs error depth using Spin



(b) Evaluated states vs goal depth using Spin



(c) Evaluated states vs error depth using MIPS



(d) Evaluated states vs goal depth using MIPS

Figure 1: The behaviour of our verification method with different verification tasks using Spin and MIPS.

4 Experiments

To evaluate the feasibility and the behaviour of our approach, we designed two experiments to investigate how constraining the verification with the planning goal impacts the verification cost. This cost is measured by the number of states evaluated by the verification tools to confirm whether or not a counterexample exists.The scripts to repeat the experiments along with the data are available online.[1]

The first experiment focuses on comparing the cost of both unconstrained and goal-constrained verification tasks while varying the safety property violation depth in order to explore situations with and without a valid planning counterexample. The safety property violation depth is hereafter termed "error depth". We synthesised a fully reachable model that consists of one critical and three independent variables, each with a range from 0 to 31. Each variable has two actions, one to increase and one to decrease its value by one. The goal is achieved when the critical variable value reaches 14. The error condition is changed from the value of the critical variable being 1 to 31. The range of the variables is chosen as 31 to expose any possible trends. Consequently, the number of variables is set to four to allow the model to be explored within a memory limit of 10 GB.

The second experiment investigates the effect of the early termination of the verification process, after achieving the goal, on the cost of verification tasks while increasing the depth of the planning goal. The model used in this experiment has one critical and four independent variables, each with a range from 0 to 15. Variables have two actions as in the previous model. This time, there is no error in the model and the goal condition is varied from critical value 1 to 14. The variables' range is reduced to 15 to permit increasing the number of variables to five while keeping the required memory within the 10 GB constraint. Both experiments are performed using the Spin model checker and the MIPS-XXL-IPC5 planner with breadth first search option.

[1] https://github.com/Anas-Shrinah/Goal-constrained-planning-domain-model-verification-repository

4.1 Results and discussion

In the first experiment, our approach showed broadly similar behaviour when it was applied using Spin and MIPS in Figure 1a and 1c. Note that the aim of these experiments is to showcase the feasibility of using our approach and to explore its behaviour, rather than comparing the performance of the verification tools. We believe such comparison depends heavily on the model under verification, for more insights the reader is referred to [Ede03, ABM09, LSD+12]. Ergo, we focus our discussion on the results obtained from model checking.

The vertical line in Figure 1a marks the goal level (critical value of 14) and splits the graph into two areas. On the right-hand side, the errors are deeper than the goal, i.e. the errors can only be reached after the goal is achieved. Thus, these errors are regarded as invalid planning counterexamples by our method as per Def. 1. Therefore, unlike unconstrained verification approaches, our method continues its exhaustive search to confirm the non-existence of any valid planning counterexamples. Thus, our method evaluates the maximum number of states for these verification tasks as shown in Figure 1a-(1).

On the left-hand side, the errors are shallower than the goal, i.e. the errors are reachable before achieving the goal. Hence, these errors are considered as valid planning counterexamples according to Def. 1. For the same verification task, Figure 1a-(2) shows that our method assesses more states than the unconstrained approaches as depicted in Figure 1a-(3). This is due to the fact that after finding an error, a safety property violation, our method keeps exploring and searching for a path to the planning goal while traditional methods terminate as soon as an error is found. However, the short counterexamples returned by these methods may or may not be valid planning counterexamples, whereas our method is guaranteed to return valid planning counterexamples only. The extra states visited by our approach are the cost associated with this guarantee.

In Figure 1a-(2) (and in Figure 1c-(2), respectively), we notice a drop in the number of evaluated states by our method as the error depth gets closer to the goal depth. This is attributed to the fact that the safety property (state trajectory constraint) in the model checker (planner) is translated into an automaton. This automaton influences the state space exploration during the verification process. The automaton has a transition that is activated when an error is reached. Therefore, if an error is reached in an early stage in the verification, the error transition is triggered and the verification tool is forced to explore more states than if the error transition was triggered closer to the goal. Once both the error and the goal transitions are triggered, then the automaton reaches an acceptance state. Thus, the search terminates with a valid planning counterexample.

In the second experiment, Figure 1b shows that when using Spin with a planing domain model with no counterexample, our approach explores fewer states than unconstrained verification methods. This reduction in the verification cost is realised by the early termination of the verification search once the goal is achieved and no error could be found at shallower depths. This advantage of the goal-constrained verification approach comes at the cost of limiting the verification results to a single planning goal. Additionally, it is observed that the number of evaluated states by the goal-constrained verification method rises as an effect of the increasing goal depth. This is caused by the expansion of the part of the model that needs to be checked as the goal depth increases. On the other hand, the unconstrained methods visit a constant number of states as they are independent from the goal depth by definition.

Another interesting observation when using the planner in Figure 1d is that our method explores more states than the unconstrained verification approaches when the planning goal depth is more than three. This behaviour is caused by the interaction of two factors. In our approach, MIPS translates the state trajectory constraint to an automaton which is then incorporated in the planning domain model. Thus, the model used in our method is more complicated than the model used by the unconstrained approaches were state trajectory constraints are not used. After a certain depth of the planning goal, the extra states evaluated as a result of the additional state trajectory constraint in our method outweigh the saving from the early termination of the verification process.

5 Related Work

Closely related, but different, is the work by [ABM09]. Their main objective is to treat verification as a planning task, whereas our aim is to demonstrate how model checkers and planners can be used for domain model verification. They proposed to perform system model verification using classical planners. To do this, they first translated the model of the system under verification into a planning domain model. Then, the negation of the safety property to be established is used as the goal for the planner, which is then consulted to find a plan that acts as counterexample for the given property. In our study, because our aim is to verify domain models against a given safety property with respect to a specific goal, we use state trajectory constraints to restrict counterexamples to identify plans that can achieve the planning goal while falsifying the safety property. In their

work the negation of the safety property is used as the goal. Whereas, in our method, the negation of the safety property is represented as a state trajectory constraint and the goal is the given planning goal.

[RPB09] also applied verification as planning to verify planning domain models, starting from LTL specifications. This work fundamentally differs from our work. They tested the impact of individual atomic propositions on the validity of the overall verified property by translating the specification properties into trap formulas. However, their method does not consider the interaction between property testing and the original planning goal. Note that finding a planning constraint to exercise a specific atomic proposition is not enough to ensure the constraint itself would be exercised during the planning process. For example, a planning goal might be achieved through a state trajectory that does not exercise the hard constraint used to represent the tested property. Our work is mainly focused on investigating this interaction. Therefore, we use state trajectory constraints to guarantee the property is tested while achieving the planning goal.

[GKS12] also used classical planners for planning systems verification, but they examined verifying plans rather than domain models. They proposed an approach that uses classical planners to find counterexamples for a given planning problem and plan instance. Their work and ours are related in that both suggest performing planning verification for a specific planning problem rather than attempting unconstrained verification of a planning system. However, their work is limited to the verification of single plan instances, whereas our method verifies all potential plans that can be spun from a domain model for a specific goal.

Among others, the researchers in [PPH98, KMH00, SHCS05, HGH$^+$08, CFF$^+$10] used model checkers to verify planning domain models. They translated the respective domain models into the input language of the selected model checker. The model checker is then applied to verify the domain model wrt. a given specification property. Similarly, we also propose a method to verify domain models using model checkers. However, our method differs from the others in two aspects. First, in the way we define the planning domain model verification problem, and, second, in the way we use model checkers to perform verification. As explained in Section 3, we constrain the verification of planning domain models with a specific goal. In contrast, previous studies perform unconstrained verification of domain models, i.e. they leave the goal open. As discussed in Section 2, the unconstrained goal may cause the model checker to return counterexamples that are unreachable when a planner uses the DUV. On the other hand, when the goal is constrained for verification, then we show that the returned counterexamples, if any, are guaranteed to be reachable by any sound planner. The second difference is that, after the planning domain model is translated to the model checker's input language, we augment the model transitions, introducing the negation of the goal as a new constraint, thereby forcing the model checker to terminate once the goal is reached. This modification prevents the model checker from returning counterexamples that falsify the given property after satisfying the goal; these are unreachable by planners.

6 Conclusions and future work

The verification of planning domain models is essential to guarantee the safety of planning-based automated systems. Invalid planning counterexamples returned by unconstrained planning domain model verification techniques undermine the verification results. They can mislead system designers to perform unnecessary remediations that can be prone to errors. In this paper, we introduced goal-constrained verification, a new concept to address this problem, which restricts the verification task to a specific goal. This limits counterexamples to those practically reachable by a planner that is tasked with achieving the goal. Since we have excluded redundant plans from our definition of valid planning counterexamples, the verification results of our method are limited to the application of non-redundant planners. A weaker form of non-redundancy will be considered in future work.

We have demonstrated how model checkers and planning techniques can be used to perform goal-constrained planning domain model verification. Our experimental evaluation confirms the feasibility of our method and presents its benefits and limitations compared to unconstrained verification methods.

We note that a grounded planning domain model defines a finite set of planning problems. Only a limited number of these problems are typically useful in practice. As such, our method might completely verify such subsets. As future work we intend to employ verification techniques such as verification reusability and compositionality [BLA$^+$99], abstraction [CGL94] and symmetry reduction [CEJS98] to perform complete verification for practical subsets of grounded planning domain models.

6.0.1 Acknowledgements

The authors are grateful to Derek Long for his useful comments.

References

[ABM09] A. Albarghouthi, J. A. Baier, and S. A. McIlraith. On the use of planning technology for verification. In *In VVPS09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.

[BHO14] S. Bensalem, K. Havelund, and A. Orlandini. Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer*, 16(1):1–12, Feb 2014.

[BLA+99] G. Behrmann, K. G. Larsen, H. R. Andersen, H. Hulgaard, and J. Lind-Nielsen. Verification of hierarchical state/event systems using reusability and compositionality. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 163–177, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[CEJS98] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In *International Conference on Computer Aided Verification*, pages 147–158. Springer, 1998.

[CFF+10] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Validation and verification issues in a timeline-based planning system. *The Knowledge Engineering Review*, 25(3):299–318, 2010.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[D+04] R. Drechsler et al. *Advanced formal verification*, volume 122. Springer, 2004.

[Ede03] S. Edelkamp. Limits and possibilities of PDDL for model checking software. *Edelkamp, & Hoffmann (Edelkamp & Hoffmann, 2003)*, 2003.

[EJN06] S. Edelkamp, S. Jabbar, and M. Nazih. Costoptimal planning with constraints and preferences in large state spaces. In *International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Preferences and Soft Constraints in Planning*, pages 38–45, 2006.

[GHL+09] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[GKS12] R. P. Goldman, U. Kuter, and A. Schneider. Using classical planners for plan verification and counterexample generation. In *Proceedings of AAAI Workshop on Problem Solving Using Classical Planning. To appear*, 2012.

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[HGH+08] K. Havelund, A. Groce, G. Holzmann, R. Joshi, and M. Smith. Automated testing of planning models. In *International Workshop on Model Checking and Artificial Intelligence*, pages 90–105. Springer, 2008.

[Hol04] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[IPC14] IPC. International planning competition 2014 web site. https://helios.hud.ac.uk/scommv/IPC-14/domains.html, 2014. [Online; accessed 15-July-2019].

[KMH00] L. Khatib, N. Muscettola, and K. Havelund. Verification of plan models using UPPAAL. In *International Workshop on Formal Approaches to Agent-Based Systems*, pages 114–122. Springer, 2000.

[LSD+12] Y. Li, J. Sun, J. S. Dong, Y. Liu, and J. Sun. Planning as model checking tasks. In *2012 35th Annual IEEE Software Engineering Workshop*, pages 177–186. IEEE, 2012.

[PPH98] J. Penix, C. Pecheur, and K. Havelund. Using model checking to validate AI planner domain models. In *Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*, 1998.

[RPB09] F. Raimondi, C. Pecheur, and G. Brat. PDVer, a tool to verify PDDL planning domains. 2009.

[SHCS05] M. H. Smith, G. J. Holzmann, G. C. Cucullu, and B. Smith. Model checking autonomous planners: Even the best laid plans must be verified. In *Aerospace Conference, 2005 IEEE*, pages 1–11. IEEE, 2005.