Temporal Planning as Refinement-Based Model Checking

Alexander Heinz,[1] **Martin Wehrle,**[2] **Sergiy Bogomolov,**[3]
Daniele Magazzeni,[4] **Marius Greitschus,**[1] **Andreas Podelski**[1]
[1]University of Freiburg, Germany, [2]University of Basel, Switzerland,
[3]Australian National University, Australia, [4]King's College London, UK
h.alexander2@gmail.com, sergiy.bogomolov@anu.edu.au, daniele.magazzeni@kcl.ac.uk,
greitsch@informatik.uni-freiburg.de, podelski@informatik.uni-freiburg.de

Abstract

Planning as model checking based on source-to-source compilations has found increasing attention. Previously proposed approaches for temporal and hybrid planning are based on *static* translations, in the sense that the resulting model checking problems are uniquely defined by the given input planning problems. As a drawback, the translations can become too large to be efficiently solvable. In this paper, we address propositional temporal planning, lifting static translations to a more flexible framework. Our framework is based on a refinement cycle that allows for adaptively computing suitable translations of increasing size. Our experiments on temporal IPC domains show that the resulting translations to timed automata often become succinct, resulting in promising performance when applied with the directed model checker MCTA.

Introduction

In this paper, we address temporal planning as model checking based on source-to-source transformations. Temporal planning is a challenging area, for which many approaches have been proposed (Vidal and Geffner 2004; Eyerich, Mattmüller, and Röger 2009; Coles et al. 2010; 2011; Gerevini, Saetti, and Serina 2010; Vidal 2014; Wang and Williams 2015; Rankooh and Ghassem-Sani 2015). To the best of our knowledge, the only attempt to translate temporal planning to automata-based model checking is a (non-archival) workshop paper by Dierks et al. (2002), which statically translates temporal planning problems to networks of timed automata.

To the best of our knowledge, all existing source-to-source compilation approaches for planning rely on a *static* translation, i. e., on a fixed translation given the input planning problem. A common problem with this approach is the size of the resulting translation, which usually grows quickly for realistic planning problems. In particular, for every automaton in the translation, a separate continuous (i. e., real-valued) *clock* variable is introduced in general, which is supposed to measure the time the automata are running. These additional clock variables can represent a severe bottleneck, because the efficiency of timed automata model checkers like UPPAAL (Behrmann, David, and Larsen 2004; Behrmann et al. 2006) or MCTA (Kupferschmid et al. 2008; Wehrle and Kupferschmid 2012) crucially depends on the number of clocks in the model.

As a central generalization to previous approaches, we move from static to dynamic encodings in order to tackle the problem of (too) large translations. Our dynamic encodings are computed based on refinement cycles, which compute translations adaptively based on the input planning problem. For the evaluation, we apply directed model checking on the translated model checking problem, based on the model checker MCTA (Kupferschmid et al. 2008; Wehrle and Kupferschmid 2012). The experiments show promising performance on common temporal IPC domains.

For a more detailed version of the paper, including the proofs, we refer to a technical report (Heinz et al. 2019).

Preliminaries

We consider propositional temporal planning with PDDL 2.1 at level 3 (Fox and Long 2003). For a set P of propositions and a real-valued *time* variable t, a *state* is a valuation of the propositions in P, together with a value from the real numbers assigned to t. The value of $p \in P$ and time variable t in state s is denoted by $s[p]$ and $s[t]$, respectively.

Definition 1 (Planning Task)**.** *A* planning task *is a tuple* $\Pi = (P, A, s_0, G)$*, where* P *is a finite set of* propositions*,* A *is a finite set of* (durative) actions*,* s_0 *is the initial state with* $s_0[t] = 0$*, and* G *the goal specification.*

Durative actions a have a non-zero duration $dur(a)$. Furthermore, a has three sets of preconditions, representing the propositions that must hold when a starts (denoted by $\text{pre}_{\vdash}$), the propositional invariant $\text{pre}_{\leftrightarrow}$ that must hold throughout a's execution, and the conditions $\text{pre}_{\dashv}$ that must hold at a's end. Similarly, a has four sets of effects: effects that are applied when the action starts ($\text{eff}^+_{\vdash}$ and $\text{eff}^-_{\vdash}$, denoting propositions that are added and deleted, respectively), and effects that are applied at a's end (denoted by $\text{eff}^+_{\dashv}$ and $\text{eff}^-_{\dashv}$).

Timed Automata

Timed automata are introduced by Alur and Dill (1994), representing finite state automata extended with real-valued *clock* variables. Clock variables x are real-valued, and obey the differential equation $\dot{x} = 1$ to represent the increase of time. Later on, the formalism has been extended to also feature integer variables (Behrmann, David, and Larsen 2004).

Let I and C be global sets of integer and clock variables, respectively. For variables $n, m \in I$, comparators $\bowtie \in \{<, \leq, \neq, \geq, >\}$, we denote the set of *integer constraints* of the form $n \bowtie c$, where $c \in \mathbb{N}$, by IC, and the set of *integer assignments* of the form $n := m$ and $n := c$ with IA. Analogously, for clock variables $x \in C$, the set of *clock constraints* of the form $x \bowtie c$ is denoted CC, and the set of clock resets of the form $x := 0$ with CR. For a set A, the powerset of A is denoted by 2^A.

Definition 2 (Timed Automata)**.** *A* timed automaton *is a tuple $\mathcal{A} = (Loc, Inv, E)$, where Loc is a finite set of* locations*, $Inv : Loc \to 2^{CC}$ is a function assigning* clock invariants *to locations, and E a finite set of* labeled edges *between locations in Loc. For edge $e \in E$, e is labeled with a* guard *consisting of integer and clock constraints from $IC \cup CC$, and with an* effect *consisting of integer assignments and clock resets from $IA \cup CR$.*

A system *$\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ of timed automata is defined as a set of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$.*

For a system of timed automata $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ with $\mathcal{A}_i = (Loc_i, Inv_i, E_i)$, the semantics of $\mathcal{S}$ is defined as follows. A *state* s is a mapping from $\mathcal{A}_i$ to locations in Loc_i for all $1 \leq i \leq n$, together with an evaluation of the variables in I and C to their respective domains.

States can be represented symbolically based on *zones*, yielding a symbolic state space $\mathcal{Z}$, called the *zone graph* (Bengtsson and Yi 2003).

For a more detailed description, the reader is referred to the literature (Bengtsson and Yi 2003).

Dynamic Encoding Refinement

We tackle the problem of static and potentially large translations by lifting the approach of Bogomolov et al. (2014a), providing a hierarchy of encodings based on iterative translation refinement. As a first (and minor) contribution, and in particular as the basis for our further approach, we adapt the translation of Bogomolov et al. (2014a) to temporal planning and timed automata (called the *base encoding* in the following). We then introduce our refinement-based translation approach using underapproximations.

Base Encoding

Each durative action $a \in A$ is translated to a corresponding timed automaton $\mathcal{A}^a$. The translation supports the epsilon separation property, which guarantees that actions do neither start nor end at the same time point (Fox and Long 2006). We adapt the translation of Bogomolov et al., taking into account the different features and limitations of timed automata compared to hybrid automata.

Duration normalization. For epsilon separation, ε is usually selected by the user as a small positive real value < 1 to enforce all actions to start or end with a minimal offset of ε. In contrast, to guarantee decidability of reachability, timed automata only support clock comparisons to *integer* values. We normalize a given $\varepsilon \in (0, 1)$ in the form $\varepsilon = 10^{-k}$ for $k \in \mathbb{N}$ to 1, yielding the normalized duration $dur(a)/\varepsilon \in \mathbb{N}$ for all durative actions a.

Model of propositional invariants. For a durative action a, propositional invariants $pre_{\leftrightarrow}$ of a are modeled by ensuring that $pre_{\leftrightarrow}$ holds when a is started, and $pre_{\leftrightarrow}$ is not violated by any other action during the execution of a. Hence, actions a' with $a' \neq a$ are neither allowed to start nor to end if a' violates $pre_{\leftrightarrow}$ when a is running. To recognize this in the translation, we introduce integer variables $lock_p^{\bot}$ and $lock_p^{\top}$ for all propositions p, with the semantics that $lock_p^{\top} = k$ (or $lock_p^{\bot} = k$, respectively) iff k durative actions are running that require p to have value *true* (or *false*, respectively). The values k of these lock variables are updated when actions start and end, respectively.

Action translation. For a given action a, we adapt the "4 location structure" of the translation $\mathcal{A}^a$ (Bogomolov et al. 2014a). The schematic structure is rehashed in Fig. 1. Following Bogomolov et al. (2014a), $\mathcal{A}^a$ simulates the execution phases "off", "starting", "running", and "finishing".

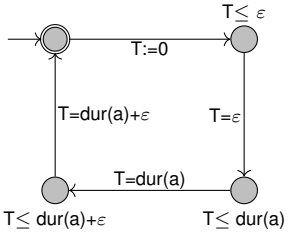

Figure 1: Global structure of timed automaton $\mathcal{A}^a$

In general, each automaton $\mathcal{A}^a$ refers to a separate clock T that keeps track of a's duration. For brevity in Fig. 1, we have only displayed the guards, invariants, and effects that refer to T, leaving out the remaining propositional guards and effects, and integer constraints and effects to provide a *locking* mechanism to ensure the ε-property. These are modeled in a straight forward way with integer variables. For example, propositional preconditions and effects of a are modeled as integer constraints in the guard and as integer assignments in the effect of the corresponding edge in $\mathcal{A}^a$.

Translation of planning tasks. The base encoding of a planning task $\Pi = (P, A, s_0, G)$ to a system of timed automata is rather straight forward: The propositions P are translated to integer variables with domain $\{0, 1\}$, and for $A = \{a_1, \ldots, a_n\}$, we have the timed system $\mathcal{S}^{\Pi} := \{\mathcal{A}^{a_1}, \ldots, \mathcal{A}^{a_n}\}$ of corresponding timed automata.

Theorem 1. *Let Π be a planning task and $\mathcal{S}^{\Pi}$ be its base encoding of timed automata. Then every symbolic plan on the zone graph of $\mathcal{S}^{\Pi}$ corresponds to a concrete plan in Π.*

Dynamic Encoding Framework

In this section, we provide a framework for computing a hierarchy of translations, which represent *underapproximations* of the original planning task with a fewer number of clock variables. This idea has been investigated for classical planning by Heusner et al. (2014). Generally, approximations and their refinements have been thoroughly studied for

planning and model checking. At the same time, such approaches usually rely on *overapproximations* (Clarke et al. 2000; Seipp and Helmert 2018; Bogomolov et al. 2014b), while our framework employs underapproximations.

We propose an encoding hierarchy which yields underapproximations in a slightly different way, by trading the number of clocks in the model versus the number of actions that are allowed to be applied in parallel. In the encoding, actions that are not allowed to be applied in parallel can *share* the same clock variable, because the corresponding automata do not simulate running the corresponding actions in parallel. To conveniently formalize this idea, we introduce the notion of *bucket-based encodings*. For an automaton $\mathcal{A}$ that models action a, we will denote $\mathcal{A}$'s clock variable by $clock(\mathcal{A})$.

Definition 3 (Bucket-Based Encoding)**.** *Consider a planning task $\Pi = (P, A, s_0, G)$ with $A = \{a_1, \dots, a_n\}$ and base encoding $\mathcal{S}^\Pi = \{\mathcal{A}^{a_1}, \dots, \mathcal{A}^{a_n}\}$. Let $\mathcal{B} = \{B_1, \dots, B_m\}$ be a set of* buckets *of actions, such that $B_i \subseteq A$ for $1 \leq i \leq m$, $\bigcup B_i = A$, and $B_i \cap B_j = \emptyset$ for $i \neq j$. The* bucket-based encoding $\mathcal{S}^{\Pi,\mathcal{B}}$ *with respect to Π and $\mathcal{B}$ is defined based on $\mathcal{S}^\Pi$ as follows. For all $1 \leq i \leq m$ and buckets $B_i = \{a_1^i, \dots, a_{n_i}^i\}$:*

1. *For all actions $a_k^i, a_t^i \in B_i$, $clock(\mathcal{A}^{a_k^i}) = clock(\mathcal{A}^{a_t^i})$, i. e., all action automata for actions in the same bucket have the same clock variable.*
2. *The automata $\mathcal{A}^1, \dots, \mathcal{A}^{n_i}$ corresponding to the actions in B_i embody an additional integer variable p_i with domain $\{0, 1\}$, initially equal to 0, such that p_i is required to be zero for $a \in B_i$ in order to start a, p_i is set to 1 once a is started, and reset to 0 again once a is finished.*

The latter condition in Def. 3 ensures that at most one automaton in each bucket is running at every time point.

Bucket-based encoding refinement. A way to *refine* is to successively allow more behavior within a refinement cycle. Generalized to temporal planning as model checking with bucket-based encodings, the refinement algorithm starts with the most strict bucket-based encoding $\mathcal{S}_0^{\Pi,\mathcal{B}}$, allowing for no parallelism at all. Inductively, if no plan can be found in $\mathcal{S}_n^{\Pi,\mathcal{B}}$ (i. e., in the bucket-based encoding applied in iteration n), then the encoding is refined to $\mathcal{S}_{n+1}^{\Pi,\mathcal{B}}$ such that strictly more behavior is possible in the refined encoding. The skeleton of the algorithm is provided in Algorithm 1.

To ensure completeness, there are two conceptual questions to be addressed, namely 1) *how* and 2) *when* to refine the encodings. We discuss these points in the following.

1) We establish a *progress property* guaranteeing that the refinement process eventually converges to a planning task with the same semantics as the original one by splitting at least one bucket in $\mathcal{B}$ into at least two buckets.

2) The decision of refining $\mathcal{S}_n^{\Pi,\mathcal{B}}$ can take place at any point in time when no solution has been found so far, if $\mathcal{S}_n^{\Pi,\mathcal{B}} \neq \mathcal{S}_{n+1}^{\Pi,\mathcal{B}}$. In contrast, if $\mathcal{S}_n^{\Pi,\mathcal{B}} = \mathcal{S}_{n+1}^{\Pi,\mathcal{B}}$, then "no solution found in $\mathcal{S}_n^{\Pi,\mathcal{B}}$" triggers iff the whole zone graph is explored without finding a solution.

Algorithm 1 Skeleton of refinement

```
1: function PLAN-WITH-REFINEMENT(P, A, s_0, G)
2:     n := 0
3:     B := {A}  // no parallelism initially
4:     while true do
5:         explore zone graph of S_n^{Π,B}
6:         if no solution found in S_n^{Π,B} then
7:             if S_n^{Π,B} ≠ S_{n+1}^{Π,B} then
8:                 n := n + 1
9:             else
10:                return unsolvable
11:            end if
12:        else
13:            return solution
14:        end if
15:    end while
16: end function
```

We emphasize that the discussions of questions 1) and 2) are of conceptual nature, with the primary objective of guaranteeing completeness of the resulting planning algorithm (we provide a concrete instantiation in the next section).

Proposition 1. *Consider a planning task Π, and let $\mathcal{S} = \{\mathcal{S}_0^{\Pi,\mathcal{B}}, \mathcal{S}_1^{\Pi,\mathcal{B}}, \dots\}$ be bucket-based encodings of Π computed based on 1) and 2). Then there exists a bucket-based encoding $\mathcal{S}_i^{\Pi,\mathcal{B}} \in \mathcal{S}$ such that there exists a trace in $\mathcal{S}_i^{\Pi,\mathcal{B}}$ that corresponds to a plan in Π iff Π is solvable.*

Framework Instantiation

We provide a simple instantiation of the refinement framework with a focus on the conceptual question on how to refine the encoding. A particular (and intuitive) situation where actions a and a' potentially need to be applied in parallel is that a's start effect supports a condition that is needed by a'. In particular, this is the case if a supports a condition that is needed as an *invariant* throughout the whole execution of a'. In the following, we propose a refinement scheme by successively splitting buckets according to actions that support invariants and preconditions of other actions. We say that an action a *supports an invariant* of action a', denoted by $a \rightsquigarrow_i a'$, if the start effect of a sets a variable to a value needed by the propositional invariant of a', i. e., there exists a proposition $p \in P$ such that $\text{eff}_\vdash^a \models p$ and $\text{pre}_\leftrightarrow^{a'} \models p$. More generally, we say that a supports an invariant of a' after n steps, denoted by $a \rightsquigarrow_i^n a'$, if there exist actions $a_1, \dots, a_n$ such that $a \rightsquigarrow_i a_1, \dots, a_n \rightsquigarrow_i a'$. Analogously, we say that a *supports a precondition* of a', denoted by $a \rightsquigarrow_p a'$, if there exists a proposition $p \in P$ such that $\text{eff}_\vdash^a \models p$, and additionally, $\text{pre}_\vdash^{a'} \models p$ or $\text{pre}_\dashv^{a'} \models p$. We define $a \rightsquigarrow_p^n a'$ on propositions analogously to $a \rightsquigarrow_i^n a'$. Furthermore, for a set of buckets $\mathcal{B}$, we say that $\mathcal{B}$ *respects* $\rightsquigarrow_i^n$ if for all actions $a_1, \dots, a_n$, $a_i \neq a_j$ for $i \neq j$, with $a_1 \rightsquigarrow_i a_2, \dots, a_{n-1} \rightsquigarrow_i a_n$, these actions are located in different buckets in $\mathcal{B}$, i. e., there are buckets $B_1, \dots, B_n \in \mathcal{B}$, $B_i \cap B_j = \emptyset$ for $i \neq j$, and $a_1 \in B_1, \dots, a_n \in B_n$. The corresponding definition for $\rightsquigarrow_p$ is analogous.

Definition 4 (Encoding Refinement)**.** *Let Π be a planning task, $\mathcal{B}$ be a set of buckets, and $\mathcal{S}^{\Pi,\mathcal{B}}$ be an encoding for Π*

	coverage							makespan						
Dom.	MCTA^{r}	MCTA^{b}	TFD	OPTIC	POPF	COLIN	ITSAT	MCTA^{r}	MCTA^{b}	TFD	OPTIC	POPF	COLIN	ITSAT
Crewp.	**30**	**30**	**30**	**30**	28	**30**	**30**	7769.1	3316.6	6239.7	**2622.9**	2747.1	**2622.9**	2836.7
Elev.	12	2	**30**	19	14	16	13	587.0	250.0	309.4	**172.0**	180.5	**172.0**	243.7
Opens.	**30**	19	**30**	**30**	**30**	**30**	24	1085.3	414.2	613.8	**123.7**	177.6	**123.7**	211.8
Parcp.	**30**	15	22	12	17	12	25	565782.6	181110.3	201830.6	75255.3	82077.5	75255.3	**74555.4**
Pegsol.	**30**	27	29	29	28	28	**30**	14.4	10.2	9.2	7.6	7.5	7.5	**7.1**
Sokob.	12	11	12	14	12	12	**16**	31.7	28.5	**19.7**	23.0	22.5	22.5	20.6
Match.	**20** (2)	**20**	**20**	0	**20**	**20**	**20**	74.2	**57.0**	72.6	-	**57.0**	**57.0**	57.2
TMS	0 (3)	0	0	0	0	0	**14**	-	-	-	-	-	-	**20**
T&O	2 (2)	0	**18**	9	8	8	5	175.6	-	101.5	40.0	38.0	42.5	**33.3**
Drv.	11 (7)	0	7	0	10	10	**15**	382.6	-	268.3	-	**122.3**	**122.3**	142.6

Table 1: Overview of coverage and makespan results (best results in bold). Abbreviations: Crewp.: Crewplanning, Elev.: Elevators, Opens.: Openstacks, Parcp.: Parcprinter, Pegsol.: Peg Solitaire, Sokob.: Sokoban, Match.: Matchcellar, TMS: Temporal Machine Shop, T&O: TurnAndOpen, Drv: DriverLog Shift

and $\mathcal{B}$. The refinement $\mathcal{S}_r^{\Pi,\mathcal{B}_r}$ *of* $\mathcal{S}^{\Pi,\mathcal{B}}$ *is defined as follows:*

1. *If there exists $n \in \mathbb{N}$ such that $\mathcal{B}$ respects $\rightsquigarrow_i^{n-1}$, but does not respect $\rightsquigarrow_i^n$, then compute $\mathcal{B}_r$ by splitting the buckets in $\mathcal{B}$ such that $\mathcal{B}_r$ respects $\rightsquigarrow_i^n$.*
2. *If $\mathcal{B}$ respects $\rightsquigarrow_i^N$ for a maximal $N \in \mathbb{N}$, then apply bullet point 1. using the relation $\rightsquigarrow_p^n$ instead of $\rightsquigarrow_i^n$.*
3. *If $\mathcal{B}$ respects $\rightsquigarrow_i^N$ and $\rightsquigarrow_p^M$ for maximal $N, M \in \mathbb{N}$, split $\mathcal{B}$ so that only actions that cannot be applied in parallel according to Π's semantics occur in equal buckets.*

Definition 4 guarantees that an exact encoding can eventually be computed. The third point can be implemented, e. g., by having each action in a separate bucket, or by sharing the same bucket only if actions have mutex invariants.

Proposition 2. *Plan-with-refinement (Alg. 1) when computing $\mathcal{S}_{n+1}^{\Pi,\mathcal{B}}$ from $\mathcal{S}_n^{\Pi,\mathcal{B}}$ according to the encoding refinement ($\mathcal{S}_r^{\Pi,\mathcal{B}_r}$ from $\mathcal{S}^{\Pi,\mathcal{B}}$ as in Def. 4) is completeness preserving.*

The most canonical (though not efficient) strategy *when* to refine is when the zone graph is explored completely.

Experiments

We conducted a feasibility study on common IPC domains, using an implementation that translates PDDL to timed automata and refines if no plan is found. As a basis, we used the model checker MCTA (Kupferschmid et al. 2008; Wehrle and Kupferschmid 2012) applied with greedy best-first search and the h^U heuristic (Kupferschmid et al. 2006). So far, we have not optimized the h^U heuristic to our specific setting. We refine when the current zone graph is explored completely. In this case, we use a simplified variant of the encoding strategy of Def. 4 to decide how to refine. The implementation of our refinement approach is called MCTA^{r}.

We compare MCTA^{r} to Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), OPTIC (Benton, Coles, and Coles 2012), POPF (Coles et al. 2010), COLIN (Coles et al. 2012), and ITSAT (Rankooh and Ghassem-Sani 2015). We also compare to MCTA applied with the base encoding, called MCTA^{b}, that allows for full parallelism. In our experiments, some action automata in the base encoding already share clocks if the corresponding actions are not applicable in parallel (i. e., still allowing full parallelism). We used the propositional temporal domains Crewplanning, Elevator, Openstacks, Parcprinter, Peg Solitaire, and Sokoban from IPC'08, Matchcellar, Temporal Machine Shop, and TurnAndOpen from IPC'14[1], and the DriverLog Shift domain (Coles et al. 2009). We used a timeout of 30 minutes and a memory limit of 4 GB per run.

Table 1 shows the results of our evaluation. The *coverage* results show the number of tasks where a goal trace has been found. For the domains that require concurrency, we not only report the number of tasks for which a goal trace has been found, but also the number of refined encodings, in parentheses, that were used for all runs. MCTA^{r} often finds goal traces for a similar number of tasks compared to the other tools, and offers its strengths in Pegsol and Parcprinter. In particular, in Parcprinter, MCTA^{r} is the only implementation that solves all tasks. In addition, we observe that, for most domains, the coverage of the refinement approach is considerably higher compared to the base encoding (MCTA^{b}). The *makespan* results in Table 1 show the average makespans per domain on the commonly solved tasks, i. e., on the tasks solved by all planners. To evaluate the "pure" makespan of the plans found by the search, the results for TFD are (like the results for MCTA^{r} and MCTA^{b}) given without improving the makespan in a post-processing step. Generally, as our approach trades efficiency versus parallelism, the makespan computed by MCTA^{r} is expected to be higher compared to the other temporal planners, which can be observed for all domains. MCTA^{b} mostly finds traces with shorter makespan than MCTA^{r} since MCTA^{b} allows for full parallelism and MCTA^{r} uses an underapproximation.

Conclusions

We proposed a generic framework for temporal planning as model checking which is based on dynamic encoding refinement. Empirically, we provided an instantiation which shows the feasibility of our approach, revealing complementary strengths to well-established planners. To further exploit its potential, it will be interesting to investigate more fine-grained instantiations, including more sophisticated strategies when to refine the encodings, as well as specific adaptations of the applied heuristic in MCTA.

[1]The IPC domains are available at https://github.com/potassco/pddl-instances.

Acknowledgments

We thank the anonymous reviewers for their comments, which helped improve the paper.

Sergiy Bogomolov was partially supported by the ARC project DP140104219 (Robust AI Planning for Hybrid Systems) and by the Air Force Office of Scientific Research under award number FA2386-17-1-4065. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force. Daniele Magazzeni was partially supported by InnovateUK under the grant TS/R018790/1.

References

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science*.

Behrmann, G.; David, A.; Larsen, K.; Håkansson, J.; Pettersson, P.; Yi, W.; and Hendriks, M. 2006. UPPAAL 4.0. In *QEST*.

Behrmann, G.; David, A.; and Larsen, K. 2004. A tutorial on Uppaal. In *SFM-RT*.

Bengtsson, J., and Yi, W. 2003. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014a. Planning as model checking in hybrid domains. In *AAAI*.

Bogomolov, S.; Frehse, G.; Greitschus, M.; Grosu, R.; Pasareanu, C. S.; Podelski, A.; and Strump, T. 2014b. Assume-guarantee abstraction refinement meets hybrid systems. In *HVC*.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *CAV*.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *ICAPS*.

Coles, A.; Coles, A.; Clark, A.; and Gilmore, S. 2011. Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. In *ICAPS*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *JAIR*.

Dierks, H.; Behrmann, G.; and Larsen, K. 2002. Solving planning problems using real-time model checking. In *AIPS-Workshop Planning via Model-Checking*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *JAIR*.

Gerevini, A.; Saetti, A.; and Serina, I. 2010. Temporal planning with problems requiring concurrency through action graphs and local search. In *ICAPS*.

Heinz, A.; Wehrle, M.; Bogomolov, S.; Magazzeni, D.; Greitschus, M.; and Podelski, A. 2019. Temporal planning as refinement-based model checking: Proofs and additional descriptions. Technical Report 288, University of Freiburg.

Heusner, M.; Wehrle, M.; Pommerening, F.; and Helmert, M. 2014. Under-approximation refinement for classical planning. In *ICAPS*.

Kupferschmid, S.; Hoffmann, J.; Dierks, H.; and Behrmann, G. 2006. Adapting an AI planning heuristic for directed model checking. In *SPIN*.

Kupferschmid, S.; Wehrle, M.; Nebel, B.; and Podelski, A. 2008. Faster than Uppaal? In *CAV*.

Rankooh, M. F., and Ghassem-Sani, G. 2015. ITSAT: An efficient SAT-based temporal planner. *JAIR*.

Seipp, J., and Helmert, M. 2018. Counterexample-guided Cartesian abstraction refinement for classical planning. *JAIR*.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *AAAI*.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In *IPC*.

Wang, D., and Williams, B. 2015. tBurton: A divide and conquer temporal planner. In *AAAI*.

Wehrle, M., and Kupferschmid, S. 2012. Mcta: Heuristics and search for timed systems. In *FORMATS*.