Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR in Planning and Scheduling

Generalizing Platform-Aware Mission Planning for Infinite-State Timed Transition Systems

Stefan Panjkovic[1,2] , **Alessandro Cimatti**[1] , **Andrea Micheli**[1] , **Stefano Tonetta**[1]
[1]Fondazione Bruno Kessler, Trento, Italy
[2]University of Trento, Italy
{spanjkovic, cimatti, amicheli, tonettas}@fbk.eu

Abstract

The Platform-Aware Mission Planning (PAMP) problem formalizes the relationship between an automated temporal planning problem and an execution platform modeled as a Timed Automaton. The PAMP problem consists in finding a valid plan that guarantees the plan executability and the satisfaction of a safety property on the platform, regardless of non-determinism. In this paper, we significantly generalize the PAMP problem along three directions. First, we consider platforms represented as infinite-state timed transition systems (TTSs), allowing a more natural and expressive modeling of realistic systems. Second, we introduce a new feature to model relations between the fluents of the planning problem and the platform variables. Finally, we generalize the semantics to cope with unbounded traces. We define a solution method for the resulting generalized PAMP, combining an automated temporal planner and an infinite-state model-checker. Our method is largely more efficient than the existing approaches for bounded PAMP problems, despite being strictly more expressive.

1 Introduction

Automated planning is a central problem in AI and it has been studied since the beginning of the field (Ghallab, Nau, and Traverso 2004). Temporal planning considers the case in which temporal constraints are present, and plans are schedules of durative actions, which is particularly relevant in applications like robotics (Ingrand and Ghallab 2017), that require reasoning on the specific timings of events and where parallelism between activities is possible.

As in any model-based technique, the plans produced by automated planning are as good as the models used to generate them. For complexity and efficiency reasons, these planning models often omit or represent very coarsely the low-level safety constraints that solution plans need to satisfy, when they are executed on a particular platform in an uncertain environment. For example, in robotic applications, a robot may be required to avoid collisions with obstacles or satisfy physical constraints such as maximum speed or acceleration. To address this issue, several authors and techniques focus on adapting the plan at runtime, using replanning (Ingrand and Ghallab 2017) or rescheduling (Morris 2016) to adjust the plan in case of unforeseen events.

In applications where there is a formal model of the execution platform and the environment, it is possible to reason explicitly about the low-level behavior and produce plans with robustness guarantees on their execution. Recently, the Platform-Aware Mission Planning (PAMP) problem has been introduced, which characterizes situations where automated planning is used to generate plans that are executed on a *platform*, where safety constraints must be satisfied and can be formally modeled (Panjkovic et al. 2025). A solution plan for the planning problem is considered valid only if all the possible executions of the platform controlled by the plan satisfy the safety constraints. This is different from runtime adaptation of the plan, as all the reasoning is done at planning time and the generated plan is *guaranteed* to be safe for all the non-deterministic behaviors of the platform.

However, in the original formulation of the PAMP problem, several simplifying assumptions are made that severly limit the applicability of the method. The platform can only be represented using timed automata, which highly restrict the expressiveness by allowing only finite domain variables and comparison of clocks to constants in guards. The proposed approaches only work in a bounded setting, where traces are assumed to have a maximum length $k|\pi|$, with $|\pi|$ being the length of the plan. Moreover, it is not possible to express *mixed* safety properties which take into consideration both the state of the planner and of the controlled platform, and this can result in plans that reach low-level states that are not aligned with the state of the planner.

In this paper, we address these issues by generalizing the PAMP problem to infinite state systems, to enable a more precise modeling of the platform constraints such as boundaries on the physical movements or on resource management. First, we reformulate the PAMP problem over Symbolic Infinite-State Timed Transitions Systems (Cimatti et al. 2019). In this way, we allow the modeling of more complex platforms using arithmetic constraints and infinite-state variables. Second, we introduce a mechanism to model and check relations between the variables of the planning problem and the platform, allowing for controlling the alignment of the traces at planning and platform levels. Third, we consider the PAMP problem in an unbounded setting, where we look for plans that ensure the safety constraints on the platform for traces of any length. All these features are uniformly modeled in our PAMP formalism and we propose a new method for tackling the problem. The basic idea is to generate candidate plans in the form of Simple Temporal

Networks (STN) (Dechter, Meiri, and Pearl 1991), allowing for temporal flexibility in the scheduling of plan commands, and use infinite-state model-checking to refine these timings until either we reject the plan as unsafe and learn a prefix of the plan that must be avoided at planning time, or we find a subset of timings allowed by the STN that guarantee the safety of the platform, hence finding a PAMP solution.

We provide an extensive experimental evaluation of the approaches against the techniques proposed in (Panjkovic et al. 2025), and show that our approach is largely superior in terms of scalability and expressiveness, even when an oracle is used to provide perfect bounds for the bounded encodings.

2 Background

Temporal Planning. We define the syntax of temporal planning by adapting the formalization used by Gigante et al. [2022], which is similar to PDDL 2.1 level 3 (Fox and Long 2003) and is also compatible with a fragment of the ANML (Smith, Frank, and Cushing 2008) language.

Definition 1 (Temporal Planning Problem)**.** *A temporal planning problem Π is a tuple $\langle P, A, I, G\rangle$, where P is a set of propositions, A is a set of durative actions, $I \subseteq P$ is the initial state and $G \subseteq P$ is the goal condition. A snap (instantaneous) action is a tuple $h = \langle pre(h), eff^+(h), eff^-(h)\rangle$, where $pre(h) \subseteq P$ is the set of preconditions and $eff^+(h), eff^-(h) \subseteq P$ are two disjoint sets of propositions, called the positive and negative effects of h, respectively. We write $eff(h)$ for $eff^+(h) \cup eff^-(h)$. A durative action $a \in A$ is a tuple $\langle a_\vdash, a_\dashv, pre^\leftrightarrow(a), [L_a, U_a]\rangle$, where $a_\vdash$ and $a_\dashv$ are the start and end snap actions, respectively, $pre^\leftrightarrow(a) \subseteq P$ is the over-all condition, and $L_a \in \mathbb{Q}_{>0}$ and $U_a \in \mathbb{Q}_{>0} \cup \{\infty\}$ are the bounds on the action duration.*

A temporal (time-triggered) plan is a set of triples, each specifying a durative action, its starting time and duration.

Definition 2 (Plan)**.** *Let $\Pi = \langle P, A, I, G\rangle$ be a temporal planning problem. A plan for Π is a set of tuples $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$, where, for each $1 \le i \le n$, $a_i \in A$ is a durative action, $t_i \in \mathbb{Q}_{\ge 0}$ is its start time, and $d_i \in \mathbb{Q}_{>0}$ is its duration.*

We use the term *length* of a time-triggered plan π (denoted with $|\pi|$) to denote the number of snap actions in π (i.e. twice the number of durative actions appearing in π).

We assume a semantics without self-overlapping of actions (Gigante et al. 2022), disallowing two instances of the same ground action to overlap in time, and will consider invalid plans that do not satisfy Definition 3.

Definition 3 (Action self-overlapping)**.** *A plan $\{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ is without self-overlapping if there exist no $i, j \in \{1, \ldots, n\}$ such that $a_i = a_j$ and $t_i \le t_j < t_i + d_i$.*

This assumption makes the temporal planning problem decidable, and is commonly adopted in the literature with many papers advocating for the limited practical interest of plans with self-overlapping (Fox and Long 2007).

Semantically, we recall the relevant definitions from Gigante et al. (2022).

Definition 4 (Set of timed snap actions)**.** *A timed snap action (TSA) is a pair $\langle t, h\rangle$, where $t \in \mathbb{Q}_{\ge 0}$ and h is a snap action. Given a plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$, the set of TSAs of π is defined as: $H(\pi) = \{\langle t_1, a_{1\vdash}\rangle, \langle t_1 + d_1, a_{1\dashv}\rangle, \ldots, \langle t_n, a_{n\vdash}\rangle, \langle t_n + d_n, a_{n\dashv}\rangle\}$.*

Definition 5 (Induced parallel plan)**.** *Let π be a plan and let $H(\pi) = \{\langle t'_1, h_1\rangle, \ldots, \langle t'_m, h_m\rangle\}$ be the set of TSAs of π. The induced parallel plan for π is the sequence $\pi^{ind} = \langle\langle t''_1, \{h | \langle t''_1, h\rangle \in H(\pi)\}\rangle, \ldots, \langle t''_k, \{h | \langle t''_k, h\rangle \in H(\pi)\}\rangle\rangle$, which is ordered and grouped with respect to the time index.*

Given $c_i = \langle a_i, t_i, d_i\rangle \in \pi$, we denote by $\pi^{\text{ind}}_\vdash(c_i)$ and $\pi^{\text{ind}}_\dashv(c_i)$ the indexes of the pairs in π^{ind} containing, in the right hand side, the snap actions $a_{i\vdash}$ and $a_{i\dashv}$ relative to c_i, respectively. In order to avoid "race conditions", the semantics prescribes that pairs of interfering events cannot happen simultaneously, and these pairs of events are called "mutex".

Definition 6 (Mutex snap actions)**.** *Two snap actions h and z are mutually exclusive (mutex), denoted by mutex(h, z), if either $pre(h) \cap eff(z) \neq \emptyset$, or $pre(z) \cap eff(h) \neq \emptyset$, or $eff^+(h) \cap eff^-(z) \neq \emptyset$, or $eff^+(z) \cap eff^-(h) \neq \emptyset$.*

Plan validity is defined as a simulation of the plan, where the state of the system is changed by the effects of the events, each event is executable and over-all conditions are satisfied in all the states in which they are active.

Definition 7 (Plan validity)**.** *Let $\Pi = \langle P, A, I, G\rangle$ be a temporal planning problem, $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$ be a plan for Π, and $\pi^{ind} = \langle\langle t'_1, B_1\rangle, \ldots, \langle t'_m, B_m\rangle\rangle$ be its induced plan. Then, π is a valid plan for Π if the following statements hold:*

1. $\forall i \in \{1, \ldots, n\}, L_{a_i} \le d_i \le U_{a_i}$;

2. there are no $h, z \in B_i$, with $h \neq z$, for some $i \in \{1, \ldots, m\}$, such that mutex(h, z).

3. given $s_0 = I$, for all $i \in \{1, \ldots, m\}$, it holds that:

(a) $\bigcup_{h \in B_i} pre(h) \subseteq s_i$;

(b) $s_i = (s_{i-1} \setminus \bigcup_{h \in B_i} eff^-(h)) \cup \bigcup_{h \in B_i} eff^+(h)$;

(c) $G \subseteq s_m$;

4. for all $c = \langle a, t, d\rangle \in \pi$ and all $\pi^{ind}_\vdash(c) \le k < \pi^{ind}_\dashv(c)$, we have $pre^\leftrightarrow(a) \subseteq s_k$;

5. for all $i, j \in \{1, \ldots, m\}$, with $i \neq j$, such that there exist $h \in B_i$ and $z \in B_j$, with mutex(h, z), we have that $|t'_i - t'_j| \ge \varepsilon$;

6. there are no $i, j \in \{1, \ldots, n\}$, with $i \neq j$, such that $a_i = a_j$ and $t_i \le t_j < t_i + d_i$.

Symbolic Timed Transition Systems. To model the platform, we will use timed transitions systems (TTSs) (Cimatti et al. 2019). Here we report the key definitions of TTSs.

Definition 8 (Symbolic Timed Transition System)**.** *A (symbolic) Timed Transition System (TTS) is a tuple $\mathcal{T} = \langle V, \mathcal{X}, \Sigma, I(V), T(V, \Sigma, V'), Z(V)\rangle$, where:*

- *V is a set of state variables;*
- *$\mathcal{X} \subseteq V$ is a set of clock variables;*
- *Σ is a set of input variables;*
- *$I(V)$ is the initial condition;*
- *$T(V, \Sigma, V')$ is the transition condition;*
- *$Z(V)$ is the invariant condition.*

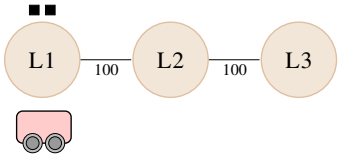

Figure 1: Depiction of an instance of the running example problem.

The state variables in V can be of type Boolean, real or clock. The initial and invariant conditions are expressions over the variables in V (invariant conditions must be convex on clocks). The transition condition is an expression over variables in V, Σ and V', where for each variable $v \in V$, V' contains the *next* version v', representing the value of v after the transition is taken.

A state $s : V \to \{\top, \bot\} \cup \mathbb{R}$ for a given TTS $\mathcal{T} = \langle V, \mathcal{X}, \Sigma, I, T, Z \rangle$ is a total assignment to the state variables in V. We use $s \models \phi$ to mean that the variable values specified by s satisfy the condition ϕ.

Definition 9 (Semantics of TTSs)**.** *The semantics of a TTS $\mathcal{T}$ is defined in terms of a transition system, where the states correspond to the states of $\mathcal{T}$ and the transitions are defined by the following rules:*

- $s \xrightarrow{d} s'$, *for* $d \in \mathbb{R}_{\geq 0}$, *if:*
  - $s'(c) = s(c) + d$, *for all* $c \in \mathcal{X}$;
  - $s'(v) = s(v)$, *for all* $v \in V \setminus \mathcal{X}$;
  - $s \models Z(V)$ *and* $s' \models Z(V)$;
- $s \xrightarrow{a} s'$, *for* $a \in \Sigma$, *if:*
  - $s, a, s' \models T(V, \Sigma, V')$;[1]
  - $s \models Z(V)$ *and* $s' \models Z(V)$.

Essentially, the state of a TTS evolves either through a delay transition, where the values of all the clocks increase by the same amount while all the other variables remain unchanged, or through a discrete transition that changes the values of the variables according to the transition relation $T(V, \Sigma, V')$. In every state, the invariant condition specified by $Z(V)$ must hold. Finally, we define the notions of timed trace and run of a TTS.

Definition 10 (Timed trace)**.** *Let $\mathcal{T}$ be a TTS $\langle V, \mathcal{X}, \Sigma, I(V), T(V, \Sigma, V'), Z(V) \rangle$. A timed action is a pair $\langle t, a \rangle$, where $t \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$. A timed trace is a (possibly infinite) sequence of timed actions $\xi = \langle \langle t_1, a_1 \rangle, \langle t_2, a_2 \rangle, \ldots, \langle t_i, a_i \rangle, \ldots \rangle$, where $t_i \leq t_{i+1}$ for all $i \geq 1$.*

Definition 11 (Run of a TTS)**.** *The run of a TTS $\mathcal{T} = \langle V, \mathcal{X}, \Sigma, I(V), T(V, \Sigma, V'), Z(V) \rangle$ with initial state $s_0 \models I(V)$ over a timed trace $\xi = \langle \langle t_1, a_1 \rangle, \langle t_2, a_2 \rangle, \ldots \rangle$ is the sequence of transitions $s_0 \xrightarrow{d_1} \xrightarrow{a_1} s_1 \xrightarrow{d_2} \xrightarrow{a_2} s_2 \ldots$, where $s \xrightarrow{d} \xrightarrow{a} s''$ indicates the subsequent transitions $s \xrightarrow{d} s'$ and $s' \xrightarrow{a} s''$, $d_1 = t_1$ and $d_i = t_i - t_{i-1}$ for all $i \geq 2$.*

[1]We abuse the notation meaning that a is set to true, and all other variables in Σ are false.

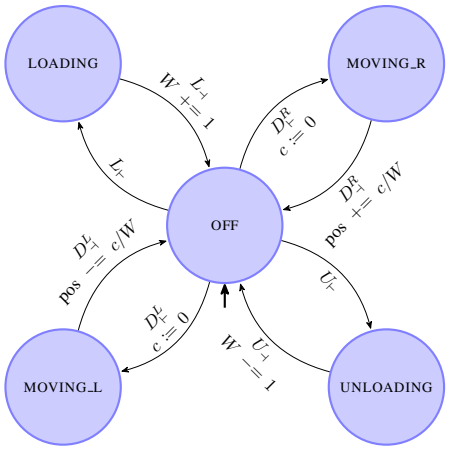

Figure 2: The TTS model of the running example.

3 PAMP Formalization

In this section, we present the formalization of the Platform-Aware Mission Planning (PAMP) problem, where we couple a high-level temporal planning problem with a low-level model of the execution platform and the environment. We generalize the framework presented in (Panjkovic et al. 2025) along three directions: first, we model the platform using a Symbolic Infinite-State Timed Transition System instead of a Timed Automaton, which allows infinite-state variables and more general constraints; second, we extend the notion of safety by allowing planning variables to be used in the property, which can be used to model and check the alignment between the traces at the two levels; finally, we consider an unbounded version of the problem, without assuming that the lengths of the platform traces are bounded w.r.t. the length of the plan to be executed.

The framework models an autonomous system architecture with two layers of abstraction: a *planning layer* describing high-level actions and mission goals, which is represented as a temporal planning problem; a *platform layer* describing low-level details and internal behaviors of the execution platform that is controlled by the planner, which is represented as a Timed Transition System (TTS). The interface between the two layers is modeled by considering the high-level events (start and end events of durative actions) as commands that are sent to the platform triggering discrete transitions: the execution of a time-triggered plan is defined by synchronizing the action start/end commands of the plan with transitions of the platform labeled with the corresponding events. Except for the assumption that the platform obeys to the commands that are scheduled by a plan, the platform is fully non-deterministic: in the time between two high-level commands, the platform can evolve by performing internal transitions and advancing time.

Example. *Figures 1 and 2 show a small running example of the considered framework. There are three locations named L1, L2 and L3 on a line, a truck which is initially in L1, and two packages also in L1. The planning problem consists of 4 actions:* DRIVELEFT(L, L') *and* DRIVERIGHT(L, L')*, which can have a duration between 100 and 500, and move*

the truck from position L to position L' (for simplicity, we represent them here as parametrized actions, but they can be grounded and correspond to the same pair of start/end platform labels); LOAD *and* UNLOAD*, which have a duration of 1, and respectively load and unload a package from the truck at the current location. The goal of the planning problem is to have both packages at location L3. Fig. 2 shows a simple execution platform, where the labels on the transitions correspond to the planning snap actions (e.g. the transition with label $L_\vdash$ is taken when the* LOAD *action is started, while the transition with label $L_\dashv$ is taken when* LOAD *is ended). This TTS models low-level aspects that do not appear in the planning problem description: there is a real variable representing the position on the line (pos), instead of a symbolic location, and the distance that the truck traverses for a* DRIVE *action depends on the duration of the drive and on the weight of the truck, which increases as packages are loaded. When the end event of a* DRIVELEFT *or* DRIVERIGHT *action is triggered ($D^L_\dashv$ or $D^R_\dashv$), the variable corresponding to the position in the platform model is updated by a term c/W, where c is a clock representing the duration of the* DRIVE *action and W is the overall weight of the truck and the packages that it is carrying (W is incremented/decremented every time the end of a* LOAD / UNLOAD *action is triggered). Initially, the weight of the truck is assumed to be 1. We specify a safety property that disallows the unloading of a package at a location that is not within distance 10 from 0, 100 or 200. Moreover, when unloading packages, we want the value of the position variable both at the planning layer and at the platform layer to correspond (L1, L2 and L3 are at positions 0, 100 and 200 respectively).*

Now, we formally define the overall framework and the problem that we are considering. Let $\Pi = \langle P, A, I, G\rangle$ be a temporal planning problem, and let $\mathcal{T} = \langle V, \mathcal{X}, \Sigma, I(V), T(V, \Sigma, V'), Z(V)\rangle$ be a TTS such that $\tau^{a_\vdash}, \tau^{a_\dashv} \in \Sigma$, for all actions $a \in A$. Suppose that $\mathcal{T}$ has a "global" clock $\gamma \in \mathcal{X}$ that is not reset in any transition and has value 0 in the initial state. Let $\rho = \langle (t_1, e_1), \ldots, (t_n, e_n)\rangle$ be a (possibly empty) ordered sequence of timed snap actions of Π, where $t_i < t_{i+1}$ for all $i \in \{1, \ldots, n-1\}$.

We denote with $\text{Exec}_\Pi(s, \rho)$, the state of Π that is reached by applying in order all the effects of the snap actions in ρ, starting from the planning state $s \subseteq P$ of Π. For an empty sequence of snap actions, we define $\text{Exec}_\Pi(s, \langle\rangle) = s$.

We define the set of states that are reachable by executing ρ on $\mathcal{T}$ from the initial state r_0, denoted by $\text{Reachable}_\mathcal{T}(r_0, \rho)$, as all the states that belong to a run of $\mathcal{T}$ where all and only the snap actions in ρ are applied, by taking the corresponding transitions at the times specified in ρ: $r_s \in \text{Reachable}_\mathcal{T}(r_0, \rho)$ if and only if there exists a run $r_0 \xrightarrow{d_1} \xrightarrow{\sigma_1} \ldots \xrightarrow{d_k} \xrightarrow{\sigma_k} r_k$, with $0 \le s \le k$, such that there exists an injective function $h : \{0, 1, \ldots, n\} \to \{0, 1, \ldots, k\}$ with the following properties

- $h(0) = 0$ (required to handle the case with $\rho = \langle\rangle$);
- for all $i \in \{1, \ldots, n\}$, for all $j \in \{1, \ldots, k\}$, if $h(i) = j$ then $\tau^{e_i} = \sigma_j$ and $t_i = \sum_{l=1}^{j} d_l$;
- for all $j \in \{1, \ldots, k\}$, if $j \notin \text{Im}(h)$ then for all $e \in \{a_\vdash, a_\dashv : a \in A\}$, $\sigma_j \neq \tau^e$.

We define analogously the set of states that are reachable *after* executing ρ on $\mathcal{T}$ from the initial state r_0, denoted by $\text{ReachableAfter}_\mathcal{T}(r_0, \rho)$ (in the previous definition, we only include in the set the final state r_k of the run).

Example. *Given the sequence $\rho = \langle (0, L_\vdash), (1, L_\dashv), (2, D^R_\vdash), (202, D^R_\dashv)\rangle$, we have that (ignoring the values of the clocks in the states):*

$$\begin{aligned}
&\textit{Reachable}_\mathcal{T}(\langle L = \textit{OFF}, pos = 0, W = 1\rangle, \rho) = \\
&\quad \{\langle L = \textit{OFF}, pos = 0, W = 1\rangle,\\
&\quad \langle L = \textit{LOADING}, pos = 0, W = 1\rangle,\\
&\quad \langle L = \textit{OFF}, pos = 0, W = 2\rangle,\\
&\quad \langle L = \textit{MOVING_R}, pos = 0, W = 2\rangle,\\
&\quad \langle L = \textit{OFF}, pos = 100, W = 2\rangle\}\\
&\textit{ReachableAfter}_\mathcal{T}(\langle L = \textit{OFF}, pos = 0, W = 1\rangle, \rho) = \\
&\quad \{\langle L = \textit{OFF}, pos = 100, W = 2\rangle\}
\end{aligned}$$

Next, we formalize the executability of a plan on a platform represented as a TTS. Intuitively, we say that a time-triggered plan is executable on a TTS if all the snap actions of the plan are applicable at the prescribed times, assuming that the platform applied all the previous commands of the plan. A snap action is applicable if a corresponding transition can be taken at the time specified in the plan.

Formally, given a state r of $\mathcal{T}$, a snap action $a_{\vdash/\dashv}$ is *applicable* in r if and only if there exists a transition $r \xrightarrow{\tau^{a_{\vdash/\dashv}}} r'$ such that $r, \tau^{a_{\vdash/\dashv}}, r' \models T(V, \Sigma, V')$ and $r' \models Z(V)$.

For a plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$, we indicate with $\rho^\pi = \langle (t'_1, e_1), \ldots, (t'_{2n}, e_{2n})\rangle$ the ordered sequence of timed snap actions of π, with $t'_i < t'_{i+1}$ for all $i \in \{1, \ldots, 2n-1\}$. For simplicity, we will assume in this paper that all the valid plans of the considered planning problems do not contain simultaneous events, i.e. snap actions scheduled at the same time: since the semantics of TTS is super-dense (multiple discrete steps can be taken at the same time in a specific order), in order to properly define and check the executability of a plan with simultaneous events for all platform behaviors, all the possible orderings for the sets of simultaneous events would need to be considered. We report at the end of Section 4 how the approach can be extended to handle simultaneous events.

Given a sequence of timed snap actions $\rho = \langle (t_1, e_1), \ldots, (t_n, e_n)\rangle$, we denote with $\rho_i = \langle (t_1, e_1), \ldots, (t_i, e_i)\rangle$ the prefix obtained by considering the first $i \le n$ timed snap actions. We denote with $\rho_0 = \langle\rangle$ the empty sequence.

Definition 12 (Time-triggered plan executability on TTS)**.** *Let Π be a temporal planning problem and let $\mathcal{T}$ be a TTS with initial state $r_0 \models I(V)$. An ordered sequence of timed snap actions $\rho = \langle (t_1, e_1), \ldots, (t_n, e_n)\rangle$ is executable on $\mathcal{T}$ if and only if for all $i \in \{0, \ldots, n-1\}$, for all $r \in \textit{ReachableAfter}_\mathcal{T}(r_0, \rho_i)$, if $r(\gamma) = t_{i+1}$ then e_{i+1} is applicable in r. A time-triggered plan π is executable on $\mathcal{T}$ if its sequence of timed snap actions ρ^π is executable on $\mathcal{T}$.*

Example. *The sequence ρ defined as $\langle (0, L_\vdash), (0.5, D^R_\vdash), (1, L_\dashv), (100.5, D^R_\dashv)\rangle$ is not exe-*

cutable, because location LOADING is reachable with $\gamma = 0.5$ *(this state belongs to* $\mathit{ReachableAfter}_{\mathcal{T}}(\langle L = \mathit{OFF}, pos = 0, W = 1\rangle, \rho)$*) and the transition with label* $D^R_{\vdash}$ *is not applicable (there is no transition from location LOADING with such a label).*

We formalize the safety of a plan w.r.t. a TTS, given a formula φ_B representing a set of bad states B, by requiring that all the states that can be reached by executing $\rho^\pi = \langle (t_1, e_1), \ldots, (t_n, e_n)\rangle$ do not belong to B. Differently from (Panjkovic et al. 2025), we allow variables of the planning problem Π to appear in φ_B: their value is obtained by simulating the application of the snap actions in ρ^π.

Definition 13 (Plan safety w.r.t. TTS)**.** *Let* $\Pi = \langle P, A, I, G\rangle$ *be a temporal planning problem and let* $\mathcal{T}$ *be a TTS with initial state* r_0*. Let* $\varphi_B(P, V)$ *be a formula representing a set of bad states* B *for* Π *and* $\mathcal{T}$*. Let* $t_0 = 0$*. An ordered sequence of timed snap actions* $\rho = \langle (t_1, e_1), \ldots, (t_n, e_n)\rangle$ *is B-safe w.r.t.* Π *and* $\mathcal{T}$ *if and only if for all states* $r \in \mathit{Reachable}_{\mathcal{T}}(r_0, \rho)$*,* $s, r \not\models \varphi_B$*, where* $s = \mathit{Exec}_{\Pi}(I, \rho_i)$ *and* $i \in \{0, 1, \ldots, n\}$ *is the maximum index s.t.* $t_i \leq r(\gamma)$*.*

Example. *Consider* $\rho = \langle (0, L_{\vdash}), (1, L_{\dashv}), (2, L_{\vdash}), (3, L_{\dashv}), (4, D^R_{\vdash}), (204, D^R_{\dashv}), (205, U_{\vdash}), (206, U_{\dashv}), (207, U_{\vdash}), (208, U_{\dashv})\rangle$*. This plan is not safe, because it is possible to reach state* $\langle L = \mathit{UNLOADING}, pos = 200/3, W = 3\rangle$ *with* $\gamma = 205$ *after starting the first unload action (this state belongs to* $\mathit{Reachable}_{\mathcal{T}}(\langle L = \mathit{OFF}, pos = 0, W = 1\rangle, \rho)$*), and the reached position is not within distance 10 from 0, 100 or 200. Consider the plan* $\pi = \{\langle \textsc{Load}, 0, 1\rangle, \langle \textsc{DriveRight}(L1, L3), 2, 200\rangle, \langle \textsc{Unload}, 203, 1\rangle\}$ *with sequence of timed snap actions* $\pi^\rho = \langle (0, L_{\vdash}), (1, L_{\dashv}), (2, D^R_{\vdash}), (202, D^R_{\dashv}), (203, U_{\vdash}), (204, U_{\dashv})\rangle$*. This plan is unsafe, because it creates a mismatch between the position variable used in the planning model and the position variable defined in the platform model: according to the planning model, the position is set to L3 after the drive, while on the platform model the position variable is set to* $200/2 = 100$*, since the truck is carrying a package.*

This definition of safety is slightly different from the one used in (Panjkovic et al. 2025): in that case, the safety property had to hold only until the time of the last step of the plan, while in this definition the safety property must hold also beyond the application of the last snap action, for any state that can be reached by performing only internal platform transitions after applying all the commands from the plan. Both semantics may be useful in different scenarios, and in Section 6 we carry out an evaluation considering both cases. We will denote with *bounded safety* the notion of safety up to the end of the plan, as defined in (Panjkovic et al. 2025), and with *unbounded safety* the notion of safety extended beyond the end of the plan (Definition 13).

We can now formally define the PAMP problem, where the objective is to find a solution plan for the planning problem, such that it is safe and executable for all the platform traces that are compliant with the plan.

Definition 14 (PAMP)**.** *A Platform-Aware Mission Planning (PAMP) problem is a tuple* $\Upsilon = \langle \Pi, \mathcal{T}, \varphi_B(P, V)\rangle$*, where* Π *is a temporal planning problem with set of variables* P*,* $\mathcal{T}$ *is a TTS with set of variables* V*, and* $\varphi_B(P, V)$ *is a formula representing a set of bad states* B *for* Π *and* $\mathcal{T}$*. A solution for* Υ *is a plan* π *such that: (i)* π *is a valid solution plan for* Π*; (ii)* π *is executable on* $\mathcal{T}$*; (iii)* π *is* B*-safe w.r.t.* Π *and* $\mathcal{T}$*.*

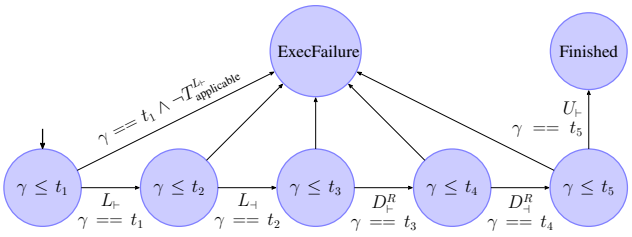

Figure 3: TTS for the plan sequence $L_{\vdash}; L_{\dashv}; D^R_{\vdash}; D^R_{\dashv}; U_{\vdash}$

Algorithm 1 Unbounded abstraction-refinement algorithm

```
1  procedure PlatformPlanning(Π, T, φ_B)
2      bad_prefixes = {}        ▷ Collects the prefixes to be avoided
3      while True do
4          π_STN ← Plan(Π, bad_prefixes)
5          pass, bad_prefix, π ← Check(T, π_STN, φ_B)
6          if pass then return π
7          else bad_prefixes ← bad_prefixes ∪ {bad_prefix}
8  procedure Check(T, π_STN, φ_B)
9      e_1, ..., e_n ← Path(π_STN);  ψ_π(t⃗) ← ⊤
10     for i = 1 to n do
11         ψ_π(t⃗) ← ψ_π(t⃗) ∧ [π_STN]_i
12         if ψ_π(t⃗) is "unsatisfiable" then
13             return false, (e_1, ..., e_i), ∅
14         while True do
15             A, φ_B' ← BuildAutomaton(T, ψ_π(t⃗))
16             outcome, ρ ← InvarCheck(A, φ_B ∨ φ_B')
17             if outcome is "safe" then
18                 if i == n then
19                     return true, (), GetPlan(ψ_π(t⃗), π_STN)
20                 else break ▷ The timings in ψ_π(t⃗) are all valid
21             else
22                 (r_0 --w_1--> --λ_1--> ... --w_k--> --λ_k--> r_k) ← ρ
23                 φ(t⃗, c⃗) ← TraceValid_{T,k}(t⃗, c⃗, σ⃗)[σ⃗/λ⃗]
24                 ψ_π(t⃗) ← ψ_π(t⃗) ∧ ¬∃c⃗.φ(t⃗, c⃗)
25                 if ψ_π(t⃗) is "unsatisfiable" then
26                     return false, (e_1, ..., e_i), ∅
```

Example. *The plan* $\pi = \{\langle \textsc{Load}, 0, 1\rangle, \langle \textsc{DriveRight}(L1, L3), 2, 400\rangle, \langle \textsc{Unload}, 403, 1\rangle, \langle \textsc{DriveLeft}(L3, L1), 405, 200\rangle, \langle \textsc{Load}, 606, 1\rangle. \langle \textsc{DriveRight}(L1, L3), 608, 400\rangle, \langle \textsc{Unload}, 1009, 1\rangle\}$ *with sequence* $\pi^\rho = \langle (0, L_{\vdash}), (1, L_{\dashv}), (2, D^R_{\vdash}), (402, D^R_{\dashv}), (403, U_{\vdash}), (404, U_{\dashv}), (405, D^L_{\vdash}), (605, D^L_{\dashv}), (606, L_{\vdash}), (607, L_{\dashv}), (608, D^R_{\vdash}), (1008, D^R_{\dashv}), (1009, U_{\vdash}), (1010, U_{\dashv})\rangle$ *is a valid solution for the example: it brings both packages to L3 and it is safe and executable because it only carries one package at a time.*

4 Solving Unbounded PAMP

In this section, we present an approach for solving our generalized version of the PAMP problem. The algorithm shares some similarities with the abstraction-refinement approach presented in (Panjkovic et al. 2025). A temporal planner

is used to solve the planning problem (without considering the platform model) and generate candidate solution plans; these plans are then checked for safety and executability on the platform model. At each failed validation check, the planner is informed about a class of plans that needs to be excluded, by analyzing the sequence of discrete choices that determined the validation failure. In (Panjkovic et al. 2025), the validation was performed by producing an encoding of the platform and the candidate plan, which was then checked with an SMT solver. Producing such an encoding was feasible, due to the strong assumption on the boundedness of the platform traces, which allowed a BMC-style encoding of the traces by unrolling the formula representing the transition relation. Since in our formulation of the PAMP problem we do not make such an assumption, in order to perform the validation check we rely on infinite-state model checking: the main idea, is that we represent the produced candidate plans using a TTS, which we then compose with the TTS representing the platform, and then apply a model checking technique on the resulting model to determine whether the safety and executability properties are satisfied.

For temporal planning, we assume a sound planner that can return plans in the form of Simple Temporal Networks (STN) (Dechter, Meiri, and Pearl 1991): a solution π_{STN} is characterized by a fixed ordering of snap actions $e_1, \ldots, e_n \leftarrow \text{PATH}(\pi_{\text{STN}})$, where each snap action e_i is associated to a time variable t_i, and a set of constraints between these time variables enforcing the ordering of the actions and their duration constraints. Given a set of plan prefixes, we assume that the search of the planner can avoid plans whose sequence of snap actions starts with a prefix in this set. For model checking, we assume a sound model checker that is able to prove temporal properties of TTSs.

Top-level PAMP procedure. The overall PAMP procedure is detailed in Algorithm 1. First, the planning problem Π is solved, obtaining a set of solution plans π_{STN}, characterized by a fixed ordering of snap actions $e_1, \ldots, e_n \leftarrow \text{PATH}(\pi_{\text{STN}})$ and a set of constraints between the time variables $t_1, \ldots, t_n$ associated to them. Then, the solution π_{STN} is passed to the CHECK procedure, together with a formula encoding the desired safety property φ_B. The CHECK procedure maintains a $\psi_\pi(\vec{t})$ formula, where $\vec{t} = (t_1, \ldots, t_n)$, representing the current region of feasible values for the time variables of the snap actions, initialized with $\top$. We iterate over all the prefixes $i \in \{1, \ldots, n\}$, and we conjoin the subset of constraints of π_{STN} considering only $t_1, \ldots, t_i$ ($[\pi_{\text{STN}}]_i$ is the conjunction of all the constraints containing t_i and one of the previous time variables $t_1, \ldots, t_{i-1}$). It is then checked whether the addition of these constraints keeps the formula $\psi_\pi(\vec{t})$ feasible, otherwise the planner is informed that the sequence of snap actions $e_1, \ldots, e_i$ is invalid.

Then, the algorithm performs a series of loops, refining the formula $\psi_\pi(\vec{t})$ until it either becomes empty, or all the possible timings defined by the constraints satisfy the safety and executability properties of the platform: the intuition is that a new TTS is constructed by composing the platform model and the plan π (with the set of constraints $\psi_\pi(\vec{t})$), a model checker is used to determine whether there is an assignment to $\vec{t}$ such that there exists a trace that violates a property, and if such a trace exists, then the formula $\psi_\pi(\vec{t})$ is refined in such a way that counterexample traces with the same sequence of discrete transitions are no longer possible; then, the procedure is repeated with the new set of constraints to find new counterexample traces (which will take different discrete transitions from the previous ones), until either the property is satisfied (meaning that for any choice in the resulting $\psi_\pi(\vec{t})$ the properties hold), or $\psi_\pi(\vec{t})$ becomes $\bot$, meaning that the plan sequence $e_1, \ldots, e_i$ is invalidated.

TTS construction. For every planning action $a \in A$, we use the Boolean variable $\tau^{a_\vdash}$ (respectively $\tau^{a_\dashv}$) to denote whether a transition with label $\tau^{a_\vdash}$ (respectively $\tau^{a_\dashv}$) is taken by $\mathcal{T}$. First, starting from the platform $\mathcal{T}$, the plan $\rho = (e_1, \ldots, e_n)$ with associated symbolic times $t_1, \ldots, t_n$ and the current formula constraining these times $\psi_\pi(\vec{t})$, we construct a TTS modeling the execution of ρ on $\mathcal{T}$.

An example of such a TTS is shown in Fig. 3. It essentially consists of a sequence of locations, one for every snap action in the plan, and the switch between consecutive locations can only occur when the value of the global clock becomes equal to the timing of the next snap action to apply. If it is not possible to apply a snap action e (represented by the $T^e_{\text{applicable}}$ formula, described afterwards), then the ExecFailure location is reached, which represents the failure of the executability for the given plan.

Let $V_\rho = \{L, \gamma, f_1, \ldots, f_m, t_1, \ldots, t_n, \tau^{e_1}, \ldots, \tau^{e_n}\}$ be a set of variables, where L is the *plan location* variable with values in $\{l_1, \ldots, l_n, l_{\text{end}}, l_{\text{ExecFailure}}\}$, γ is the global clock, $f_1, \ldots, f_m$ are Boolean *fluent variables* one for each proposition in P denoting whether it is true or false, and $t_1, \ldots, t_n$ and $\tau^{e_1}, \ldots, \tau^{e_n}$ are as defined above. Let $\mathcal{X}_\rho = \{\gamma\}$, and let $\Sigma_\rho = \{\sigma\}$, where σ is the *plan event* input variable with values in $\{\epsilon_1, \ldots, \epsilon_n, \epsilon_{\text{internal}}, \epsilon_{\text{fail}}\}$. The variables are initialized according to the initial state and the constraint $\psi_\pi(\vec{t})$:

$$I_\rho(V_\rho) \coloneqq (L = l_1 \wedge \gamma = 0 \wedge \psi_\pi(\vec{t}) \wedge \textstyle\bigwedge_{i=1}^{m} f_i = I_\Pi(f_i))$$

An invariant condition states that the *plan location* variable can keep a certain value only until the time of the next plan snap action to be applied:

$$Z_\rho(V_\rho) \coloneqq \textstyle\bigwedge_{i=1}^{n} (L = l_i \rightarrow \gamma \leq t_i)$$

We define a transition relation $T_\rho\left(V_\rho, \Sigma_\rho, V'_\rho\right)$ with the following constraints:

- the time variables always keep their initial value

$$\textstyle\bigwedge_{i=1}^{n} (t_i = t'_i)$$

- when changing the *plan location* from one value to the next, the global clock must have the value of the timing of the next snap action to be applied, the corresponding transition must be taken by $\mathcal{T}$, and the variables $f_1, \ldots, f_m$ are changed according to the snap action effects

$$\textstyle\bigwedge_{i=1}^{n}(\sigma = \epsilon_i \rightarrow (L = l_i \wedge L' = l_{i+1} \wedge \gamma = t_i \wedge \tau^{e_i} \wedge \bigwedge_{\substack{j \in \{1,\ldots,m\}: \\ f_j \in \text{eff}^+(e_i)}} f'_j = \top \wedge \bigwedge_{\substack{j \in \{1,\ldots,m\}: \\ f_j \in \text{eff}^-(e_i)}} f'_j = \bot))$$

- when the platform $\mathcal{T}$ performs an internal transition, the *plan location* variable remains unchanged

$$\sigma = \epsilon_{\text{internal}} \rightarrow L = L'$$

• when the platform $\mathcal{T}$ performs a transition that is associated to a plan snap action, the *plan event* variable must have the value of the corresponding event

$$\bigwedge_{a \in A}(\tau^{a_{\vdash/\dashv}} \rightarrow \bigvee_{\substack{i \in \{1,\ldots,n\}: \\ e_i \equiv a_{\vdash/\dashv}}} \sigma = \epsilon_i)$$

• when a *fluent variable* changes, the *plan event* variable must be associated to a snap action that affects such fluent

$$\bigwedge_{i=1}^{m} f_i \neq f_i' \rightarrow \bigvee_{\substack{j \in \{1,\ldots,n\}: \\ f_i \in \text{eff}(e_j)}} (\sigma = \epsilon_j)$$

We define with T_{fail} the transition condition that sets the *plan location* variable to $l_{\text{ExecFailure}}$, which occurs when a certain snap action must be applied (the global clock reaches the timing of the application), but the transition to the next *plan location* value cannot be taken (expressed using the negation of the existentially quantified transition relation):

$$\sigma = \epsilon_{\text{fail}} \rightarrow \bigvee_{j=1}^{n}(L = l_j \wedge L' = l_{\text{ExecFailure}} \wedge \gamma = t_j \wedge \neg(\exists \vec{v'}.T_{\mathcal{T}}(V_{\mathcal{T}}, \Sigma_{\mathcal{T}}, V_{\mathcal{T}}') \wedge T_\rho(V_\rho, \Sigma_\rho, V_\rho'))[\sigma/\epsilon_j])$$

Finally, we can define the TTS $\mathcal{A} = \langle V_{\mathcal{A}}, \mathcal{X}_{\mathcal{A}}, \Sigma_{\mathcal{A}}, I_{\mathcal{A}}(V_{\mathcal{A}}), T_{\mathcal{A}}(V_{\mathcal{A}}, \Sigma_{\mathcal{A}}, V_{\mathcal{A}}'), Z_{\mathcal{A}}(V_{\mathcal{A}})\rangle$, where $V_{\mathcal{A}} = V_{\mathcal{T}} \cup V_\rho$, $\mathcal{X}_{\mathcal{A}} = \mathcal{X}_{\mathcal{T}} \cup \mathcal{X}_\rho$, $\Sigma_{\mathcal{A}} = \Sigma_{\mathcal{T}} \cup \Sigma_\rho$, $I_{\mathcal{A}} = I_{\mathcal{T}} \wedge I_\rho$, $T_{\mathcal{A}} = T_{\mathcal{T}} \wedge T_\rho \wedge T_{\text{fail}}$ and $Z_{\mathcal{A}} = Z_{\mathcal{T}} \wedge Z_\rho$. The invariant condition is $\varphi_{B'} \coloneqq (L = l_{\text{ExecFailure}})$.

Model checking and refinement. Once the TTS $\mathcal{A}$ is built, we check the invariant property $\varphi_B \vee \varphi_{B'}$, i.e. we check whether the platform $\mathcal{T}$ controlled by the plan can violate either the safety or the executability property. If the property is satisfied, then we either move to the next plan prefix, or if we already reached the end we can extract a specific time-triggered plan by solving the set of constraints $\psi_\pi(\vec{t})$ (since the invariant check passed, all the timing in $\psi_\pi(\vec{t})$ are valid).

Otherwise, the model checker returns a trace $\rho = \left(r_0 \xrightarrow{w_1}\xrightarrow{\lambda_1} \ldots \xrightarrow{w_k}\xrightarrow{\lambda_k} r_k\right)$ that violates the safety or executability property. Now, we want to remove a region of values from $\psi_\pi(\vec{t})$, for which a trace with the same discrete transitions of ρ is still a valid counterexample for the invariant properties. Let $\text{TRACEVALID}_{\mathcal{T},k}(\vec{t}, \vec{c}, \vec{\sigma})$ be the formula representing the unrolling of the transition relation of $\mathcal{A}$ for k steps (where k is the length of trace ρ), where $\vec{c}$ is the set of clocks of $\mathcal{A}$, and $\vec{\sigma}$ is the set of all the other variables of $\mathcal{A}$. Consider the formula $\phi(\vec{t}, \vec{c}) = \text{TRACEVALID}_{\mathcal{T},k}(\vec{t}, \vec{c}, \vec{\sigma})\left[\vec{\sigma}/\vec{\lambda}\right]$, where we fix the values of the discrete variables according to the trace ρ. Now, by existentially quantifying over the clocks $\vec{c}$ and negating the resulting formula, we obtain a formula encoding the times $\vec{t}$ for which a trace with discrete choices $\vec{\lambda}$ is no longer feasible (for any delay of the clock variables $\vec{c}$). Therefore, we update $\psi_\pi(\vec{t})$ by setting it to $\psi_\pi(\vec{t}) \wedge \neg\exists\vec{c}.\phi(\vec{t}, \vec{c})$. In the end, we check whether $\psi_\pi(\vec{t})$ has become unsatisfiable, in which case we return a bad prefix to the planner. Otherwise, we repeat the validation process with the new $\psi_\pi(\vec{t})$, and we are guaranteed that an eventual new counterexample trace will make different discrete choices.

Theorem 1. *Let $\Upsilon = \langle \Pi, \mathcal{T}, \varphi_B\rangle$ be a PAMP problem, and let π be a plan returned by* $\text{PLATFORMPLANNING}(\Pi, \mathcal{T}, \varphi_B)$. *Then, π is a valid solution for Υ.*

Proof. We need to show that π satisfies Definition 14:

1. Let π_{STN} be the last plan in STN form returned by $\text{PLAN}(\Pi, \text{bad\_prefixes})$. Because of the soundness of the planner, every plan that has the ordering of snap actions given by $e_1, \ldots, e_n \leftarrow \text{PATH}(\pi_{\text{STN}})$, and a value for the variables representing the time of the snap actions $t_1, \ldots, t_n$ that satisfies the STN constraints of π_{STN}, is a valid solution for Π. The plan π is returned by $\text{GETPLAN}(\psi_\pi(\vec{t}), (e_1, \ldots, e_n))$ when iteration $i = n$ of the $\text{CHECK}(\mathcal{T}, \pi_{\text{STN}}, \varphi_B)$ procedure is reached. Since we are at iteration $i = n$, $\psi_\pi(\vec{t})$ is a conjunction of all the constraints in π_{STN} and eventually other formulas that were used for the refinement (line 28 of Algorithm 1); hence, $\psi_\pi(\vec{t})$ implies the constraints of π_{STN}. Since the ordering of the snap actions is $e_1, \ldots, e_n$, we can conclude by the soundness of the planner that π is a solution for Π.
2. Suppose, for the sake of contradiction, that π is not executable on $\mathcal{T}$. Let ρ^π be the sequence of snap actions of π. By Definition 12, there exists $i \in \{0, \ldots, n-1\}$ and $r \in \text{ReachableAfter}_{\mathcal{T}}(r_0, \rho_i^\pi)$ such that $r(\gamma) = t_{i+1}$ and e_{i+1} is not applicable in r. By definition of $\text{ReachableAfter}_{\mathcal{T}}$, there exists a run $r_0 \xrightarrow{d_1}\xrightarrow{\sigma_1} \ldots \xrightarrow{d_k}\xrightarrow{\sigma_k} r_k = r$ such that there exists an injective function $h : \{0, 1, \ldots, i\} \rightarrow \{0, 1, \ldots, k\}$ with the properties described in Section 3. Consider the last TTS $\mathcal{A}$ that is returned by $\text{BUILDAUTOMATON}(\mathcal{T}, \psi_\pi(\vec{t}))$ at iteration i, for which the $\text{INVARCHECK}(\mathcal{A}, \varphi_B \vee \varphi_{B'})$ returns "safe". We can show that there exists a run $r_0' \xrightarrow{d_1}\xrightarrow{\sigma_1} \ldots \xrightarrow{d_k}\xrightarrow{\sigma_k} r_k' = r'$ of $\mathcal{A}$ such that $r'(\gamma) = t_{i+1}$ and e_{i+1} is not applicable in r': the run satisfies the transition relation $T_{\mathcal{A}} = T_{\mathcal{T}} \wedge T_\rho \wedge T_{\text{fail}}$ of $\mathcal{A}$ since the original run satisfies $T_{\mathcal{T}}$; T_ρ is satisfied because of the properties of function h, specifying that the transitions corresponding to the snap actions are taken only when the value of the global clock equals the value of the time of the snap action; $r'(\gamma) = t_{i+1}$ because the run has the same delay transitions, and e_{i+1} is not applicable since there is no transition with the corresponding label that can be taken in $\mathcal{T}$. The transition from r' to a state with location ExecFailure is possible, as the fact that e_{i+1} is not applicable satisfies the constraints of T_{fail}. Therefore, there exists a run from an initial state of $\mathcal{A}$ to a state with location ExecFailure, but this is a contradiction since $\text{INVARCHECK}(\mathcal{A}, \varphi_B \vee \varphi_{B'})$ returned "safe". Thus, π is executable on $\mathcal{T}$.
3. Suppose, for the sake of contradiction, that π is not B-safe w.r.t. Π and $\mathcal{T}$. Let ρ^π be the sequence of snap actions of π. By Definition 13, there exists $r \in \text{Reachable}_{\mathcal{T}}(r_0, \rho^\pi)$ such that $s, r \models \varphi_B$, where $s = \text{Exec}_\Pi(I, \rho_i)$ with i being the maximum index such that $t_i \leq r(\gamma)$. The proof then follows the previous reasoning: there exists a run of $\mathcal{A}$ which satisfies φ_B, but this is a contradiction as the invariant checking procedure did not find any run to a state satisfying φ_B. Thus, π is B-safe w.r.t. Π and $\mathcal{T}$. □

Handling simultaneous snap actions. For the sake of simplicity, we assumed that the plans generated by the planner could not contain simultaneous snap actions (snap actions scheduled at the same time), but the framework and the approach can be easily extended to handle such plans. If a plan has a set of snap actions that need to be applied simultaneously, we can obtain the strongest guarantee on the robustness of the plan by requiring that the executability and safety constraints are satisfied for all the possible total orderings of the simultaneous snap actions in the plan. In order to handle this requirement, we can model such behaviors in the TTS representing the plan. Suppose, for example, that a sequence of timed snap actions ρ contains (t, e) and (t, e'): we add two transitions from the state with *plan location* corresponding to time t, with the guard that the global clock must be t, that do not modify the *plan location*, apply the snap action, and set a boolean variable to true, which represents the fact that the snap action was applied; we also add two transitions that change the *plan location* to $l_{\text{ExecFailure}}$, which can be taken only if the snap actions have not been applied so far (the boolean variable is false) and the transition that applies them cannot be taken. The transition that changes the *plan location* to the next value can only occur if both the boolean variables corresponding to (t, e) and (t, e') are true (i.e. they were applied). The model checker can choose any order for the simultaneous snap actions, and if the property holds, then any possible ordering is safe and executable.

5 Related Work

The PAMP problem has been originally defined in (Panjkovic et al. 2025), where solution approaches are presented that provide *system-level guarantees* on the execution behaviors of generated plans, considering a platform modeled as a Timed Automaton. These techniques adopt the very strong assumption that platform traces are bounded w.r.t. the length of the plans, i.e. it is assumed that platform traces can be of length at most $k|\pi|$, for a given constant k and a plan π. In many scenarios, it is true that such a constant k exists, as a platform may only be able to perform a bounded number of internal transitions between two commands of the plan. However, this constant k can be very large, especially for platforms that can perform many internal transitions with short delays in between, and choosing a wrong value for k impacts the soundness of the approach, as potentially longer counterexample traces could be missed. Having a large bound k can have a huge impact on the performance of these approaches, as they rely on the solving of quantified SMT formulas that scale linearly with k. Our approach relaxes the boundedness assumption by relying on model-checking for the verification of the safety and executability constraints. It is also more expressive, as it allows for infinite-state transition systems in the platform modeling and it provides a mean to express relations among the planning and platform variables. This is crucial in order to check if some undesired misalignment can occur between the state of the planning problem and the state of the platform.

The PAMP problem shares strong connections with conformant planning (Ghallab, Nau, and Traverso 2004), where actions can have non-deterministic effects and no runtime observation is granted. To the best of our knowledge, no conformant planner tackles the temporal case, and our approach differs in that we consider the platform modeled as a timed transition system, allowing for (possibly unbounded) behaviors during and among planning actions; moreover, we provide guarantees on the execution safety on the platform.

A closely related work is (Viehmann, Hofmann, and Lakemeyer 2021), which models the platform layer as a timed automaton and assumes an abstract sequential plan is provided. The authors use Metric Temporal Logic (MTL) to express constraints between the planning and platform layers. However, their work focuses on verifying whether a single platform execution satisfies a given plan (an $\exists\exists$ problem). Our approach, by contrast, includes a formal framework for plan generation and addresses a universally-quantified validation of platform behavior ($\exists\forall$). Moreover, our interface between abstraction layers differs: we use TTS labels to represent "commands" sent by the plan, whereas Viehmann, Hofmann, and Lakemeyer employ MTL constraints to restrict possible traces.

Bozzano, Cimatti, and Roveri (2021) present an autonomy framework using symbolic, model-based reasoning to integrate plan generation, execution, monitoring and FDIR. The approach models the controlled system as a finite-state non-deterministic planning problem with resource estimation functions. Our work employs a infinite-state model for the system and features a richer platform representation.

Finally, there are several works on plan execution with a known model of the underlying environment: PRS-style architectures (Ingrand, Georgeff, and Rao 1992) are able to perform execution-time reasoning and planning in dynamic domains; approaches based on hierarchical task-oriented refinement methods (Patra et al. 2021) integrate planning and acting systems by using the same operational models; SkiROS (Rovida et al. 2017) is an approach developed on top of ROS that integrates planning with low-level robotics control, offering a GUI to supervise the execution of a plan. These approaches are fundamentally different from PAMP as they operate in an online setting, while we focus on providing robustness guarantees at planning time.

6 Experimental Evaluation

We developed the presented technique in a solver written in Python based on pyVMT, a python framework for handling VMT (Verification Modulo Theory) problems (Cimatti, Griggio, and Tonetta 2022). For temporal planning, we use the TAMER planner (Valentini, Micheli, and Cimatti 2020): the planning algorithm is an explicit-state heuristic-search approach, that explores all the possible total orderings of sequences of snap actions, and updates in each visited state a STN every time a new snap action is added to the sequence. A state is pruned whenever the set of STN constraints becomes unfeasible (which means that the selected sequence of actions cannot be scheduled while respecting all the constraints), and if a goal state is reached, then all the time-triggered plans with that sequence of snap actions and a scheduling which satisfies the STN constraints are valid solution plans for Π. We also adapted TAMER to support the avoidance of a set of bad plan prefixes during its search.

Domain	Unsound PCMT BMC (bound=2)	Unsound PCMT REF (bound=2)	PCMT BMC with Oracle	PCMT REF with Oracle	Bounded Sem. UAR-PAMP (Ours)
Driverlog1	0	0	0	0	7
Driverlog2	0	0	0	0	9
Factory1	4	9	0	0	8
Factory2	10	10	0	0	3
Fix1	5	8	1	1	11
Fix2	5	8	0	1	16
Rover	35	44	0	1	40
Total	59	79	1	3	94

Table 1: Coverage table for the bounded safety semantics

	Bounded Sem. UAR-PAMP	Unbounded Sem. UAR-PAMP
Driverlog1	7	8
Driverlog2	9	10
Factory1	8	8
Factory2	3	2
Fix1	11	11
Fix2	16	16
Rover	40	42
BatteryVars	14	14
Driverlog1Vars	5	5
Driverlog2Vars	6	6
Total	119	122

Table 2: Coverage table for UAR-PAMP

For model checking, we use NUXMV (Cimatti et al. 2019), a symbolic model checker that is able to prove temporal properties of TTSs. In particular, we exploit the IC3IA algorithm (Cimatti et al. 2014), which combines search-based, deductive and abstraction techniques to iteratively build proofs while disproving candidate counterexamples.

The solver accepts temporal planning problems written in PDDL2.1 or ANML, and platform models written in timed SMV (Cimatti et al. 2019). It is also possible to specify whether to use the *bounded* or *unbounded* notion of safety.

We experimentally evaluated the new approach on the benchmark set used in (Panjkovic et al. 2025) and three novel sets of benchmarks, DRIVERLOG, FIX and BATTERY. DRIVERLOG is similar to the running example used in this paper, and it is scaled by increasing the number of locations and the number of packages to be delivered. FIX describes a scenario in which a certain amount of objects needs to be fixed, but at the platform layer a single *fix* action may not be sufficient to repair an object. There is however a bound, over which it is guaranteed that the fix succeeds, but this information is not present in the planning model. We scale the instances by increasing the number of objects and the number of necessary *fix* actions to repair an object. Finally, in BATTERY, an energy storage device needs to be fully recharged, but the amount of charge after every *recharge* action can have some uncertainty according to the platform model. The safety property specifies that the absolute difference between the *planning charge variable* and the *platform charge variable* must not be larger than a threshold, and the only way to guarantee this property, is to apply a sufficient number of times the charge action. We scale the instances by increasing the number of required recharges.

All the experiments were performed on a cluster of identical machines with AMD EPYC 7413 24-Core Processor and running Ubuntu 20.04.6. We used a timeout of 14400 seconds and a memory limit of 20GB. Table 1 is the coverage table comparing our approach (adopting the *bounded safety* semantics) UAR-PAMP with the bounded approaches of (Panjkovic et al. 2025) (which we denote with PCMT BMC and PCMT REF). For each domain, we manually found the bound on the number of platform internal transitions between two consecutive commands from the planner, and used such bound as an "Oracle" for the approaches of (Panjkovic et al. 2025). In this way, the bounded approaches of (Panjkovic et al. 2025) are guaranteed to find PAMP plans that are valid also for the unbounded case, but we remark that this is not a fair comparison, as the "oracle" bound is not easily computable in general. We also ran those approaches using the smallest bound 2. Clearly, by choosing bound 2, the resulting approaches are unsound in general, as given a plan there may exist a platform trace invalidating that plan of length greater than twice the length of the plan. This actually happens in many of our domain instances: the unsound approaches find plans, which are not valid according to the unbounded semantics nor for the approaches using oracle bounds. The results clearly show that UAR-PAMP solves significantly more problems than PCMT BMC and PCMT REF, even when adopting 2 as the bound (which is the absolute best case for the bounded approaches). The dominance of UAR-PAMP is also evident in the cactus plot in Fig. 4, which demonstrates the runtime performance. Finally, Table 2 compares our approach in both the bounded and unbounded safety semantics for all the domains, including the ones with variable relations. The results show that the number of solved instances is very similar, highlighting that the choice of the semantics for the safety specification does not significantly affect the performance.

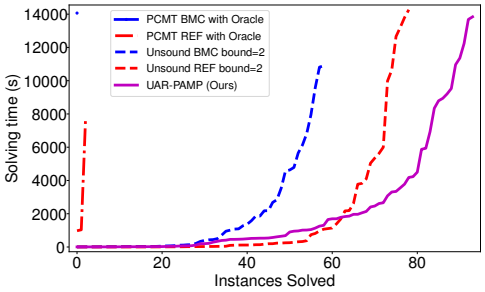

Figure 4: Cactus plot for the bounded safety semantics

7 Conclusions

We presented a significant generalization of the Platform-Aware Mission Planning (PAMP) problem to consider infinite-state execution platform models: we defined the problem over Timed Transition Systems (TTS) instead of Timed Automata, we allowed custom relations between planning fluents and platform variables, and we consider unbounded traces instead of bounded ones. Our solution method combines a temporal planner with an infinite-state model-checker, and we show that it is consistently more efficient than the existing approach for bounded PAMP problems, despite being strictly more expressive.

Acknowledgments

This work has been partly supported by the PNRR project iNEST – Interconnected Nord-Est Innovation Ecosystem (ECS00000043) funded by the European Union NextGenerationEU program and by the STEP-RL project funded by the European Research Council (grant n. 101115870). We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

References

Bozzano, M.; Cimatti, A.; and Roveri, M. 2021. A comprehensive approach to on-board autonomy verification and validation. *ACM Trans. Intell. Syst. Technol.* 12(4).

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2014. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, 46–61. Springer.

Cimatti, A.; Griggio, A.; Magnago, E.; Roveri, M.; and Tonetta, S. 2019. Extending nuxmv with timed transition systems and timed temporal properties. In Dillig, I., and Tasiran, S., eds., *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, 376–386. Springer.

Cimatti, A.; Griggio, A.; and Tonetta, S. 2022. The VMT-LIB language and tools. In *SMT*, volume 3185 of *CEUR Workshop Proceedings*, 80–89. CEUR-WS.org.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*.

Fox, M., and Long, D. 2007. A note on concurrency and complexity in temporal planning.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Gigante, N.; Micheli, A.; Montanari, A.; and Scala, E. 2022. Decidability and complexity of action-based temporal planning over dense time. *Artif. Intell.* 307:103686.

Ingrand, F., and Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artif. Intell.* 247:10–44.

Ingrand, F.; Georgeff, M.; and Rao, A. 1992. An architecture for real-time reasoning and system control. *IEEE Expert* 7(6):34–44.

Morris, P. H. 2016. The mathematics of dispatchability revisited. In Coles, A. J.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, 244–252. AAAI Press.

Panjkovic, S.; Cimatti, A.; Micheli, A.; and Tonetta, S. 2025. Platform-aware mission planning. In *Proceedings of the International Conference of Automated Planning and Scheduling ICAPS 2025 (https://doi.org/10.48550/arXiv.2501.09632)*.

Patra, S.; Mason, J.; Ghallab, M.; Nau, D.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence* 299:103523.

Rovida, F.; Crosby, M.; Holz, D.; Polydoros, A. S.; Großmann, B.; Petrick, R. P. A.; and Krüger, V. 2017. *SkiROS—A Skill-Based Robot Control Platform on Top of ROS*. Cham: Springer International Publishing. 121–160.

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *KEPS 2008*.

Valentini, A.; Micheli, A.; and Cimatti, A. 2020. Temporal planning with intermediate conditions and effects. In *AAAI 2020*.

Viehmann, T.; Hofmann, T.; and Lakemeyer, G. 2021. Transforming robotic plans with timed automata to solve temporal platform constraints. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2083–2089. International Joint Conferences on Artificial Intelligence Organization. Main Track.