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

Anonymous IJCAI'25 Submssion

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 approach 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 *et al.*, 2004]. Temporal planning studies the case where temporal constraints are present and actions have durations.

In a recent paper, Panjkovic *et al.* introduced the Platform-Aware Mission Planning (PAMP) problem to faithfully characterize situations in which automated planning is used to generate plans that are executed on a *platform* where safety constraints must be satisfied [Panjkovic *et al.*, 2025]. The PAMP problem is a generalization of temporal planning where a plan is considered valid only if all the possible executions of the platform controlled by the plan satisfy the safety constraints. In Panjkovic *et al.*, timed automata are used to model the platform and a method is proposed to solve PAMP only in a bounded setting, where traces are assumed to have a maximum length $k|\pi|$ with $|\pi|$ being the length of the plan.

In this paper, we generalize 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 traces can have arbitrary length. We look for plans that ensure the safety constraints on the platform considering traces of any length; moreover, we discuss and uniformly support variations of the semantics for the PAMP problem considering or disregarding behaviors after the end of the plan execution.

All the aforementioned features are uniformly modeled in our PAMP formalism and we propose a new method for tackling this problem. The basic idea of the approach is to generate candidate plans in the form of Simple Temporal Networks (STN) [Dechter *et al.*, 1991], allowing for some 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 we show that our approach is 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 *et al.*, 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 (instanta-*

neous) 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 absolute starting time and its 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* $1 \leq i \leq n$*,* $a_i \in A$ *is a durative action,* $t_i \in \mathbb{Q}_{\geq 0}$ *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).

A time-triggered plan π is a solution plan for the problem Π if each durative action in the plan can be applied at the specified time with the given duration (the preconditions of its start and end snap actions are true at the start and at the end of the action respectively), and if by applying all the effects a final state is reached after the end of the last action in which the goal condition is satisfied. The formal semantics is presented in [Gigante *et al.*, 2022], which we omit here for the sake of brevity.

We assume a semantics without self-overlapping of actions [Gigante *et al.*, 2022], making the temporal planning problem decidable: it is not possible for two instances of the same ground action to overlap in time.

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* $t_i \leq t_j < t_i + d_i$*.*

Symbolic Timed Transition Systems We recall the standard definitions of symbolic Timed Transition Systems [Cimatti *et al.*, 2019].

Definition 4 (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.*

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.

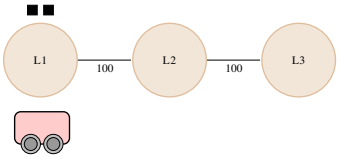


Figure 1: Depiction of an instance of the running example problem.

Given a TTS $\mathcal{T} = \langle V, \mathcal{X}, \Sigma, I, T, Z\rangle$, a state $s : V \to \{\top, \bot\} \cup \mathbb{R}$ of $\mathcal{T}$ 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 5 (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:*
  - $s, a, s' \models T(V, \Sigma, V')$*;*
  - $s \models Z(V)$ *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.

Definition 6 (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 7 (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* $i \geq 2$*.*

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. The definitions 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 to use infinite-state variables and more general constraints; second, we extend the notion of safety by allowing planning variables to be used in the property specification, which can be used to

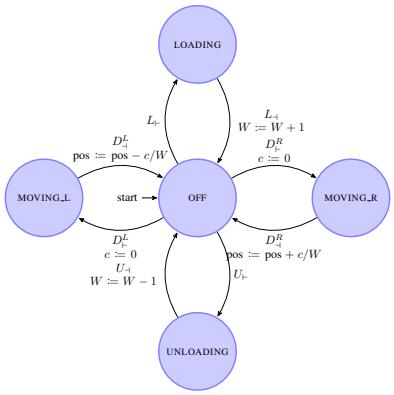


Figure 2: The TTS model of the running example.

model and check the alignment between the traces at the two levels; finally, we consider an unbounded version of the problem, where we do not assume 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* that describes the high-level actions and mission goals, which is represented as a temporal planning problem; a *platform layer* that describes the 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.

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 model consists of 4 actions: DRIVELEFT and DRIVERIGHT, which can have a duration between 100 and 500, and move the truck along the line for the specified duration (it takes 100 units of time to reach an adjacent named location); 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 a low-level aspect that does not appear in the planning problem description: the distance that the truck traverses for a certain DRIVE duration also depends 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 (pos) 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 L1, L2 or L3. Moreover, when unloading packages, we want the value of the position variable both at the planning layer and at the platform layer to be the same.

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 $\text{Exec}_\Pi(\rho, s)$, the state 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 $\text{Exec}_\Pi(\langle\rangle, s) = s$.

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).

Next, we formalize the executability of a time-triggered 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 $r, a, r' \models T(V, \Sigma, V')$ 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. 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 $i \leq n$ timed snap actions. We denote with $\rho_0 = \langle\rangle$ the empty sequence.

Definition 8 (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 $r \in ReachableAfter_{\mathcal{T}}(r_0, \rho_i)$, if $r(\gamma) = t_{i+1}$ then e_{i+1} is applicable in r. A time-triggered plan π of Π is executable on $\mathcal{T}$ if its sequence of timed snap actions ρ^π is executable on $\mathcal{T}$.*

For 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 executable, because location MOVING_R is reachable with $\gamma = 1$ (this state belongs to ReachableAfter$_{\mathcal{T}}(\langle L = \text{OFF}, \text{pos} = 0, W = 1 \rangle, \rho_2)$) and the transition with label $L_\dashv$ is not applicable (there is no transition from location MOVING_R with such a label).

We formalize the notion of safety for 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 $\rho^\pi = \langle (t_1, e_1), \ldots, (t_n, e_n) \rangle$, within time t_n, do not belong to B. Differently from the framework in [Panjkovic *et al.*, 2025], we allow variables of the planning problem Π to appear in φ_B, and their value is obtained by simulating the application of the snap actions in ρ^π up to to point at which a certain state is reached.

Definition 9 (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 $r \in Reachable_{\mathcal{T}}(r_0, \rho)$, $s, r \not\models \varphi_B$, where $s = Exec_\Pi(\rho_i, I)$ with $i \in \{0, 1, \ldots, n\}$ being the maximum index such that $t_i \leq r(\gamma)$.*

For instance, consider the sequence $\rho = \langle (0, L_\vdash), (1, L_\dashv), (2, D^R_\vdash), (202, D^R_\dashv), (203, U_\vdash), (204, U_\dashv) \rangle$. This plan is unsafe, because it creates a mismatch between the position variable used in the planning model and the pos variable defined in the platform model: according to the planning model, the position is set to 200 after the end of the drive, while on the platform model the pos 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 possible internal behavior of the platform that occurs afterwards. Both semantics may be useful in different scenarios, and in Section 6 we carry an evaluation considering both cases.

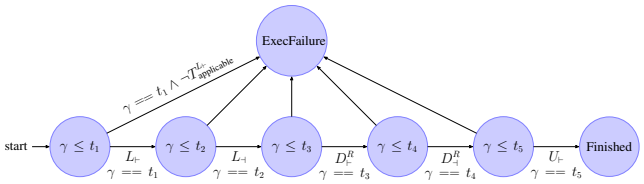


Figure 3: The TTS representing the plan sequence $L_\vdash; L_\dashv; D^R_\vdash; D^R_\dashv; U_\vdash$

Algorithm 1 Unbounded abstraction-refinement algorithm

procedure PLATFORMPLANNING($\Pi, \mathcal{T}, \varphi_B$)
  bad_prefixes $= \{\}$ ▷ Collects the prefixes that need to be avoided
  while True **do**
    $\pi_{\text{STN}} \leftarrow$ PLAN(Π, bad_prefixes)
    pass, bad_prefix, $\pi \leftarrow$ CHECK($\mathcal{T}, \pi_{\text{STN}}, \varphi_B$)
    if pass **then**
      return π
    else
      bad_prefixes $\leftarrow$ bad_prefixes $\cup$ {bad_prefix}
procedure CHECK($\mathcal{T}, \pi_{\text{STN}}, \varphi_B$)
  $e_1, \ldots, e_n \leftarrow$ PATH(π_{STN})
  $\psi_\pi(\vec{t}) \leftarrow \top$
  for $i = 1$ to n **do**
    $\psi_\pi(\vec{t}) \leftarrow \psi_\pi(\vec{t}) \wedge [\pi_{\text{STN}}]_i$
    if $\psi_\pi(\vec{t})$ **is** "unsatisfiable" **then**
      return false, $(e_1, \ldots, e_i), \emptyset$
    while True **do**
      $\mathcal{A}, \varphi_{B'} \leftarrow$ BUILDAUTOMATON($\mathcal{T}, \psi_\pi(\vec{t})$)
      outcome, $\rho \leftarrow$ INVARCHECK($\mathcal{A}, \varphi_B \vee \varphi_{B'}$)
      if outcome **is** "safe" **then**
        if $i == n$ **then**
          return true, (), GETPLAN($\psi_\pi(\vec{t})$, $(e_1, \ldots, e_n)$)
        else
          break ▷ The timings in $\psi_\pi(\vec{t})$ are all valid
      else
        $\left(r_0 \xrightarrow{w_1} \xrightarrow{\lambda_1} \ldots \xrightarrow{w_k} \xrightarrow{\lambda_k} r_k\right) \leftarrow \rho$
        $\phi(\vec{t}, \vec{c}) \leftarrow$ TRACEVALID$_{\mathcal{T},k}(\vec{t}, \vec{c}, \vec{\sigma})\ \left[\vec{\sigma}/\vec{\lambda}\right]$
        $\psi_\pi(\vec{t}) \leftarrow \psi_\pi(\vec{t}) \wedge \neg\exists \vec{c}.\phi(\vec{t}, \vec{c})$
        if $\psi_\pi(\vec{t})$ **is** "unsatisfiable" **then**
          return false, $(e_1, \ldots, e_i), \emptyset$

Definition 10 (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, $\mathcal{T}$ is a TTS with set of variables V, and $\varphi_B(P, V)$ is a formula representing a set of bad states B for $\mathcal{T}$. 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. $\mathcal{T}$.*

For example, the sequence $\rho = \langle (0, L_\vdash), (1, L_\dashv), (2, D^R_\vdash), (402, D^R_\dashv), (403, U_\vdash), (404, U_\dashv), (405, D^L_\vdash), (605, D^L_\dashv), (606, L_\vdash), (607, L_\dashv), (608, D^R_\vdash), (1008, D^R_\dashv), (1009, U_\vdash), (1010, U_\dashv) \rangle$ is a valid solution for the example: it brings both packages to location L3 and it is executable and safe because it carries only 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 many 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, thanks 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 use the TAMER planner [Valentini *et al.*, 2020], a sound and complete approach for temporal planning that returns plans in the form of Simple Temporal Networks (STN) [Dechter *et al.*, 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 which enforce the ordering of the actions and their duration constraints. For model checking, we use NUXMV [Cimatti *et al.*, 2019], a symbolic model checker that is able to prove temporal properties of TTSs.

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 $e_1, \ldots, e_n \rightarrow \text{PATH}(\pi_{\text{STN}})$ 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, that is initialized with $\top$. It iterates over all the prefixes $i \in \{1, \ldots, n\}$, and it first conjuncts 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 infeasible.

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 checking technique is applied 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 satisified (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.

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 of Π and the constraint $\psi_\pi(\vec{t})$:

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

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, and that the time variables $t_1, \ldots, t_n$ keep their initial values:

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

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

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

- when changing the *plan location* from one value to the next, the global clock must have the value of the timing of the next plan 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

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

- 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

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

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

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

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 $\varphi_{B'} := (L = l_{\text{ExecFailure}})$.

Once the TTS $\mathcal{A}$ is built, we perform invariant checking on the 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 $\rho = \left(r_0 \xrightarrow{w_1}\xrightarrow{\lambda_1} \dots \xrightarrow{w_k}\xrightarrow{\lambda_k} r_k\right)$ 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.

5 Related Work

This paper generalizes the PAMP problem, originally defined in [Panjkovic *et al.*, 2025]. PAMP provides *system-level guarantees* on the execution behaviors of generated plans. Our generalization is far from trivial as it allows for infinite-state transition systems in the platform modeling, it relaxes the boundedness assumption on the platform traces and provides a mean to express relations among the planning and platform variables. Moreover, we also uniformly consider variations of the PAMP semantics depending on the behavior of the platform after the end of the plan.

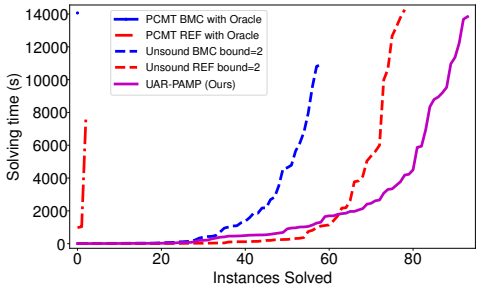
The PAMP problem shares strong connections with conformant planning [Ghallab *et al.*, 2004], where actions can have non-deterministic effects and no runtime observation is granted. To the best of our knowledge, no conformant planner tackles the temporal case, and our approach differs in that we consider the platform modeled as a timed transition system, hence 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 *et al.*, 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 Infinite Timed Transition Systems labels to represent "commands" sent by the plan, whereas Viehmann *et al.* employ MTL constraints to restrict possible traces.

Finally, Bozzano *et al.* (2021) present an autonomy framework leveraging symbolic, model-based reasoning to integrate plan generation, execution, monitoring and FDIR. Their approach models the controlled system as a finite-state nondeterministic planning problem enhanced with resource estimation functions. Our work employs a infinite-state model for the system and features a richer platform representation.

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

(a)



(b)

	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

(c)

Figure 4: Experimental results. Coverage table (a) and cactus plot (b) for the bounded safety semantics and coverage table for UAR-PAMP on domains without and with variable relations (c).

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 *et al.*, 2022]. We used the nuXmv IC3IA [Cimatti *et al.*, 2014] solver as a backend for model-checking and TAMER as planner. 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 possible to specify which notion of safety to use: safety up to the end of the plan, as defined in [Panjkovic *et al.*, 2025], which we will refer to as *bounded safety*; safety also beyond the end of the plan, for any state that can be reached by performing only internal transitions in the platform after applying all the commands from the plan, which we will call *unbounded 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 certain number of times the charge action.

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. Fig. 4a 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 for 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. 4b, which demonstrates the runtime performance. Finally, Fig. 4c 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, higlighting that the choice of the semantics for the safety specification does not significantly affect the performance of the approach.

7 Conclusions

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

In future work, we plan to investigate the use of parametric synthesis techniques as an alternative solution method and to consider other representation models for the platform, such as contracts.

References

[Bozzano *et al.*, 2021] Marco Bozzano, Alessandro Cimatti, and Marco Roveri. A comprehensive approach to on-board autonomy verification and validation. *ACM Trans. Intell. Syst. Technol.*, 12(4), aug 2021.

[Cimatti *et al.*, 2014] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

[Cimatti *et al.*, 2019] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. Extending nuxmv with timed transition systems and timed temporal properties. In Isil Dillig and Serdar Tasiran, editors, *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*, pages 376–386. Springer, 2019.

[Cimatti *et al.*, 2022] Alessandro Cimatti, Alberto Griggio, and Stefano Tonetta. The VMT-LIB language and tools. In *SMT*, volume 3185 of *CEUR Workshop Proceedings*, pages 80–89. CEUR-WS.org, 2022.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 1991.

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

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[Gigante *et al.*, 2022] Nicola Gigante, Andrea Micheli, Angelo Montanari, and Enrico Scala. Decidability and complexity of action-based temporal planning over dense time. *Artif. Intell.*, 307:103686, 2022.

[Panjkovic *et al.*, 2025] Stefan Panjkovic, Alessandro Cimatti, Andrea Micheli, and Stefano Tonetta. Platform-aware mission planning. *CoRR*, abs/2501.09632, 2025.

[Smith *et al.*, 2008] David Smith, Jeremy Frank, and William Cushing. The anml language. In *KEPS 2008*, 2008.

[Valentini *et al.*, 2020] Alessandro Valentini, Andrea Micheli, and Alessandro Cimatti. Temporal planning with intermediate conditions and effects. In *AAAI 2020*, 2020.

[Viehmann *et al.*, 2021] Tarik Viehmann, Till Hofmann, and Gerhard Lakemeyer. Transforming robotic plans with timed automata to solve temporal platform constraints. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2083–2089. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.