Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

Platform-Aware Mission Planning

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

Planning for autonomous systems typically requires reasoning with models at different levels of abstraction, and the harmonization of two competing sets of objectives: high-level mission goals that refer to an interaction of the system with the external environment, and low-level platform constraints that aim to preserve the integrity and the correct interaction of the subsystems. The complicated interplay between these two models makes it very hard to reason on the system as a whole, especially when the objective is to find plans with robustness guarantees, considering the non-deterministic behavior of the lower layers of the system.

In this paper, we introduce the problem of Platform-Aware Mission Planning (PAMP), addressing it in the setting of temporal durative actions. The PAMP problem differs from standard temporal planning for its *exists-forall* nature: the high-level plan dealing with mission goals is required to satisfy safety and executability constraints, for all the possible non-deterministic executions of the low-level model of the platform and the environment. We propose two approaches for solving PAMP. The first baseline approach amalgamates the mission and platform levels, while the second is based on an abstraction-refinement loop that leverages the combination of a planner and a verification engine. We prove the soundness and completeness of the proposed approaches and validate them experimentally, demonstrating the importance of heterogeneous modeling and the superiority of the technique based on abstraction-refinement.

Introduction

A commonly employed architecture to realize autonomous systems consists in using automated planning for the synthesis of plans to achieve given mission goals, which are then executed on the system's platform (and the environment the system operates in). This separation of concerns allows the planner to reason on a high-level model, disregarding details and non-deterministic behaviors of the platform. However, in safety-critical applications, the mission objectives (also called "science objectives", in the space domain) often conflict with the safety constraints dictated by the platform and the environment (also called "engineering constraints"). The former are easily expressible as goals in the planning problem and are existential in nature (i.e., one needs to find any plan achieving the goals), whereas safety constraints are universally quantified over all the possible executions of the given plan (i.e., one needs to prevent any execution violating the safety). To represent this scenario and to offer reasoning guarantees on the autonomous system as a whole, we need to model *both* the planning objectives and the possible platform behaviors, formalizing the link between these levels.

In this paper, we aim at finding solution plans achieving high-level mission goals and offering formal robustness guarantees on the execution of such plans on the platform. We formalize and tackle the "Platform-Aware Mission Planning" (PAMP) problem, which aims at finding a plan guaranteed to achieve the mission objectives, such that all the possible evolutions of the platform controlled by the plan satisfy a set of safety and executability properties, taking into account both the flexibility of the execution platform (the possible choices the platform can operate while obeying a given plan) and the non-determinism from the environment. Concretely, we propose a formal framework, in which a high-level temporal planning representation is coupled with a low-level description of the platform that executes the generated plans. We use a standard temporal planning model adapted from Gigante et al. (2022) and a timed automaton (Alur and Dill 1994) to represent the platform. This framework uses existing models, can be easily instantiated in practice, and mirrors the architecture of real autonomous systems (Gat 1998).

We propose two techniques to solve the PAMP problem. First, we develop a baseline "amalgamated" approach grounded in Satisfiability Modulo Theory (SMT) (Barrett et al. 2009): we combine a standard encoding for temporal planning with a novel encoding for the executability and safety of a symbolic temporal plan. The resulting formula is quantified ($\exists\forall$), mirroring the intuitive quantifier alternation in the problem definition. Second, we propose a much more efficient approach exploiting the subdivision of the framework in two layers of abstraction. The technique uses a state-of-the-art heuristic search temporal planner to generate candidate plans, which are then checked for safety and executability, explaining the conflicts as sequences of events that the planner is required to avoid for subsequent candidates. This check employs a specialized version of the amalgamated SMT encoding. We formally prove the soundness and completeness of the proposed approaches and we develop two scalable case-studies to empirically evaluate

them, showing their empirical effectiveness.

Background

Temporal Planning We start by defining the syntax of a temporal planning problem: we adapt the formal model used by Gigante et al. (2022), which is quite close to PDDL 2.1 level 3 (Fox and Long 2003).

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, $I : P \to \{\top, \bot\}$ 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 (time-triggered) plan is defined as a set of triples, each specifying an action, its 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 $1 \le i \le n$, $a_i \in A$ 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 will call *length* of a time-triggered plan π (denoted with $|\pi|$) the number of snap actions in π (i.e. twice the number of durative actions).

A time-triggered plan π is a solution plan for the problem Π if, starting from the initial state I, 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), which makes 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 $t_i \le t_j < t_i + d_i$.*

This formal model of temporal planning is simplified with respect to concrete planning languages (e.g. for the sake of simplicity we only defined the ground model, while most languages allow a first-order lifted representation for compactness), but it already achieves the full computational complexity of very expressive languages such as ANML (Gigante, Micheli, and Scala 2022).

Timed Automata Here, we recall the standard definitions.

Definition 4 (Clock constraints)**.** *Let $\mathcal{X}$ be a finite set of elements called clocks. A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$, where $x, y \in \mathcal{X}$, $\sim \in \{\le, <, =, >, \ge\}$ and $n \in \mathbb{N}$. We use $\mathcal{C}(\mathcal{X})$ to denote the set of clock constraints on $\mathcal{X}$.*

A Timed Automaton generalizes finite-state automata by means of clock variables that can be reset and track the advancement of time.

Definition 5 (Timed Automaton)**.** *A Timed Automaton (TA) is a tuple $\mathcal{T} = \langle \Sigma, \mathcal{L}, l_0, \mathcal{X}, \Delta, Inv\rangle$, where:*
- *Σ is the alphabet;*
- *$\mathcal{L}$ is a finite set of locations;*
- *$l_0 \in \mathcal{L}$ is the initial location;*
- *$\mathcal{X}$ is a finite set of clocks;*
- *$\Delta \subseteq \mathcal{L} \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{X}} \times \mathcal{L}$ is the transition relation;*
- *$Inv : \mathcal{L} \to \mathcal{C}(\mathcal{X})$ maps each location to its invariant.*

We will write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l'\rangle \in \Delta$.

Definition 6 (State of TAs)**.** *Given a TA $\mathcal{T} = \langle \Sigma, \mathcal{L}, l_0, \mathcal{X}, \Delta, Inv\rangle$, a state of $\mathcal{T}$ is a pair $\langle l, u\rangle$, where $l \in \mathcal{L}$ and $u : \mathcal{X} \to \mathbb{R}_{\ge 0}$ is a clock assignment.*

We use $u \models g$ to mean that the clock values denoted by u satisfy the guard $g \in \mathcal{C}(\mathcal{X})$. For $d \in \mathbb{R}_{\ge 0}$, we use $u + d$ to denote the clock assignment that maps all clocks $c \in \mathcal{X}$ to $u(c) + d$. For $r \subseteq \mathcal{X}$, we use $[r \to 0]u$ to denote the clock assignment that maps all $c \in r$ to 0, and all $c \in \mathcal{X} \setminus r$ to $u(c)$.

Definition 7 (Semantics of TAs)**.** *The semantics of a TA is defined in terms of a transition system, with states of the form $\langle l, u\rangle$ and transitions defined by the following rules:*
- *$\langle l, u\rangle \xrightarrow{d} \langle l, u + d\rangle$ if $u \in Inv(l)$ and $(u + d) \in Inv(l)$, for $d \in \mathbb{R}_{\ge 0}$;*
- *$\langle l, u\rangle \xrightarrow{a} \langle l', u'\rangle$ if $l \xrightarrow{g,a,r} l'$, $u \models g$, $u' = [r \to 0]u$ and $u' \in Inv(l')$.*

Definition 8 (Timed trace)**.** *Let $\mathcal{T} = \langle \Sigma, \mathcal{L}, l_0, \mathcal{X}, \Delta, Inv\rangle$ be a TA. A timed action is a pair $\langle t, a\rangle$, where $t \in \mathbb{R}_{\ge 0}$ 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 $t_i \le t_{i+1}$ for all $i \ge 1$.*

Definition 9 (Run of a TA)**.** *The run of a TA $\mathcal{T} = \langle \Sigma, \mathcal{L}, l_0, \mathcal{X}, \Delta, Inv\rangle$ with initial state $\langle l_0, u_0\rangle$ 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 $\langle l_0, u_0\rangle \xrightarrow{d_1} \xrightarrow{a_1} \langle l_1, u_1\rangle \xrightarrow{d_2} \xrightarrow{a_2} \langle l_2, u_2\rangle \ldots$, where $d_1 = t_1$ and $d_i = t_i - t_{i-1}$ for all $i \ge 2$.*

Problem Definition

In this section, we formalize our composite framework and the Platform-Aware Mission Planning (PAMP) problem.

In our framework, we consider an autonomous system architecture with two layers of abstraction: a *planning layer*, represented as a temporal planning problem, describing the high-level durative actions and a mission goal; and a *platform layer*, represented as a TA, which describes the low-level details and internal actions of the platform that is controlled by the planner. We consider an interface between the two layers where each start and end event of an action of the planning problem is associated with a signal of

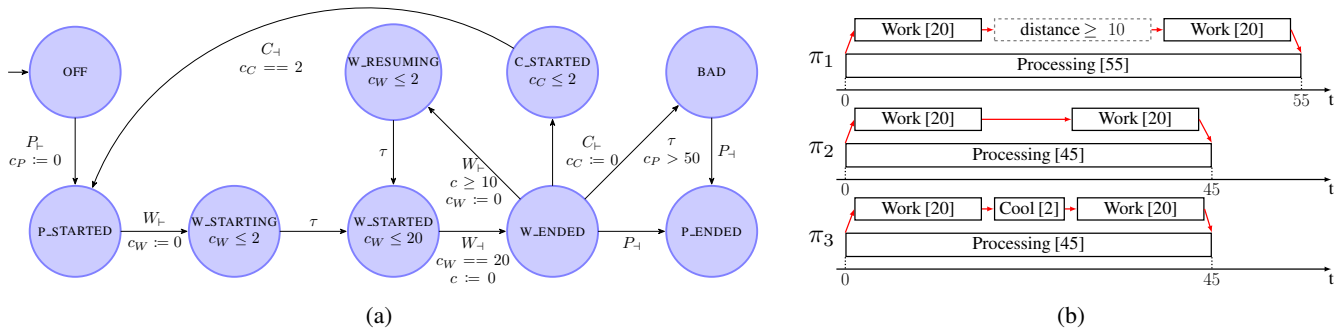


Figure 1: Running example TA platform model (a) and example plans (b). The first two plans violate safety (π_1) and executability (π_2) constraints, while the third one (π_3) is correct for all platform executions.

the TA (a letter of its alphabet), and define the execution of a time-triggered plan by synchronizing the action start/end commands of the plan with transitions of the platform labeled with the corresponding events. In the time between two high-level commands, the platform can freely evolve by performing internal transitions and advancing time.

Example. *Figures 1a and 1b show a small running example of the considered framework. An industrial process needs to be completed by starting a "Process" (P) action and applying in parallel two "Work" (W) actions. In the planning model, we have a Boolean variable* processing*, initially false, and a bounded integer variable* completed-steps*, initially 0 (the integer variable, used for simplicity, can be compiled in our planning model using unary or binary encodings). The "Process" action sets the* processing *variable to true at the start and back to false at the end. The "Work" action has an over-all condition requiring the* processing *variable to be true during the duration of the action, and an end effect that increments the value of the* completed-steps *variable by 1. There is also a "Cooldown" action, which does not have any effect on the variables of the planning model. The goal requires that the "Work" action is applied two times (i.e.* completed-steps $== 2$*). At the platform layer (Fig. 1a), there are transitions with labels corresponding to the start/end events of the high-level durative actions (e.g. $W_\vdash$ corresponds to the start of "Work"), and internal transitions that the platform can take, which are not linked to high-level events (the transitions with label τ). The TA encodes a low-level constraint between successive applications of the "Work" actions that is not modeled in the planning problem: when a "Work" action is performed (reaching the* W_ENDED *location), a component becomes heated and needs to cool down before the next "Work" action can be applied, and this occurs either by waiting 10 time units (transition from* W_ENDED *to* W_RESUMING *with guard $c \geq 10$), or by explicitly applying a "Cooldown" action which cools the component after 2 time units (transitions to* C_STARTED *and* P_STARTED *with labels $C_\vdash$ and $C_\dashv$). Moreover, the process has a deadline of 50 time units, after which the platform can reach an undesirable state (transition from* W_ENDED *to* BAD *with guard $c_P > 50$).*

We start by introducing the notion of states that are reachable by executing a plan on a TA. Intuitively, the states that can be reached by executing a sequence of timed snap actions ρ on a TA $\mathcal{T}$, are 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. We formally define this with a function H, that maps the snap actions of ρ with steps in the run of $\mathcal{T}$ where the transitions with the corresponding labels are taken.

Definition 10 (States reachable by plan execution)**.** *Let $\Pi = \langle P, A, I, G\rangle$ be a temporal planning problem, and let $\mathcal{T} = \langle \Sigma, \mathcal{L}, l_0, \mathcal{X}, \Delta, Inv\rangle$ be a TA such that $\tau^{a_\vdash}, \tau^{a_\dashv} \in \Sigma$, for all actions $a \in A$. 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 $i \in \{1, \ldots, n-1\}$. A state r_s is "reachable" by executing ρ on $\mathcal{T}$ from the initial state $r_0 = \langle l_0, u_0\rangle$ 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 $0 \leq s \leq k$, and an injective function $H : \{0, 1, \ldots, n\} \to \{0, 1, \ldots, k\}$ with the following properties:*

1. *$H(0) = 0$ (Required to handle the case with $\rho = \langle\rangle$);*
2. *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 $t_i = \sum_{l=1}^{j} d_l$;*
3. *for all $j \in \{1, \ldots, k\}$, if $j \notin Im(H)$ 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 ReachableAfter$_\mathcal{T}(r_0, \rho)$ (we include in the set only the final state r_k of the run).

In the running example, consider the sequence $\rho = \langle (0, P_\vdash), (1, W_\vdash), (21, W_\dashv)\rangle$. Then we have that

Reachable$_\mathcal{T}$(OFF, ρ) = {OFF, P_STARTED, W_STARTING, W_STARTED, W_ENDED, BAD}

ReachableAfter$_\mathcal{T}$(OFF, ρ) = {W_ENDED, BAD}

For instance, BAD $\in$ ReachableAfter$_\mathcal{T}$(OFF, ρ) since there exists the run OFF $\xrightarrow{d_1=0}\xrightarrow{\sigma_1=P_\vdash}$ P_STARTED $\xrightarrow{d_2=1}\xrightarrow{\sigma_2=W_\vdash}$ W_STARTING $\xrightarrow{d_3=2}\xrightarrow{\sigma_3=\tau}$ W_STARTED $\xrightarrow{d_4=18}\xrightarrow{\sigma_4=W_\dashv}$ W_ENDED $\xrightarrow{d_5=31}\xrightarrow{\sigma_5=\tau}$ BAD and the function H s.t. $H(1)=1$, $H(2)=2$ and $H(3)=4$, satisfying Definition 10.

Next, we formalize the notion of executability of a time-triggered plan on the platform. Intuitively, we say that a time-triggered plan is executable on a TA if every snap action of the plan is applicable at the prescribed time, for any possible internal behavior of the platform, 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 $\langle l, u\rangle$ of $\mathcal{T}$, a snap action $a_{\vdash/\dashv}$ is *applicable* in $\langle l, u\rangle$ if and only if there exists a transition $l \xrightarrow{g,\tau^{a_{\vdash/\dashv}},r} l'$ such that $u \models g$ and $[r \to 0]u \in \text{Inv}(l')$.

For a time-triggered 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 assume 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 TA 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 11 (Time-triggered plan executability on TA)**.** *Let Π be a temporal planning problem and let $\mathcal{T}$ be a TA with initial state $r_0 = \langle l_0, u_0\rangle$. Suppose that $\mathcal{T}$ has a global clock γ that is not reset in any transition and has value 0 in the initial state. 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 $r = \langle l, u\rangle \in \textit{ReachableAfter}_\mathcal{T}(r_0, \rho_i)$, if $u(\gamma) = t_{i+1}$ 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 $\rho = \{(0, P_\vdash), (1, W_\vdash), (21, W_\dashv), (22, W_\vdash), (42, W_\dashv), (45, P_\dashv)\}$, which corresponds to the second plan in Fig. 1b is not executable on the TA of Fig. 1a, because it is possible to reach location W_ENDED with $\gamma = 22$ and $c = 1$ (this state belongs to $\text{ReachableAfter}_\mathcal{T}(r_0, \rho_3)$) and the transition with label $W_\vdash$ is not applicable (the guard $c \geq 10$ is false).

We formalize the notion of safety for a plan w.r.t a TA, given 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.

Definition 12 (Plan safety w.r.t. TA)**.** *Let Π be a temporal planning problem and let $\mathcal{T}$ be a TA with initial state $r_0 = \langle l_0, u_0\rangle$. Suppose that $\mathcal{T}$ has a global clock γ that is not reset in any transition and has value 0 in the initial state. Let $B \subseteq \mathcal{L} \times \mathbb{R}^\mathcal{X}$ be a set of bad states for $\mathcal{T}$. 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. $\mathcal{T}$ if and only if for all states $r = \langle l, u\rangle \in \textit{Reachable}_\mathcal{T}(r_0, \rho)$ such that $u(\gamma) \leq t_n$, $r \notin B$. A time-triggered plan π of Π is B-safe w.r.t. $\mathcal{T}$ if its sequence of timed snap actions ρ^π is B-safe w.r.t. $\mathcal{T}$.*

Consider the running example, and suppose that B is the set of all states with location BAD. The sequence $\rho = \{(0, P_\vdash), (1, W_\vdash), (21, W_\dashv), (32, W_\vdash), (52, W_\dashv), (55, P_\dashv)\}$, which corresponds to the first plan in Fig. 1b, is not B-safe, because it is possible to reach location BAD with $\gamma = 53$ between the application of the last two snap actions $(52, W_\dashv)$ and $(55, P_\dashv)$ (this state belongs to $\text{Reachable}_\mathcal{T}(r_0, \rho)$).

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 13 (PAMP)**.** *A Platform-Aware Mission Planning (PAMP) problem is a tuple $\Upsilon = \langle \Pi, \mathcal{T}, B\rangle$, where Π is a temporal planning problem, $\mathcal{T}$ is a TA, and $B \subseteq \mathcal{L} \times \mathbb{R}^\mathcal{X}$ is a set of bad states 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. $\mathcal{T}$.*

The third plan in Fig. 1b is a solution for the example PAMP problem. The application of the "Cool" action between the two "Work" actions makes it fully executable and safe, since the BAD is unreachable ($c_P > 50$ remains false).

Solution Approaches

In this section, we propose two approaches for solving the PAMP problem. We assume that a constant k is given, which represents the maximum possible ratio between the length of a platform trace and the length of the executed plan. Hence, when considering a plan π of length L (the number of snap actions in the plan), we will analyze its safety and executability for platform traces of length up to κL. It is reasonable to assume that such a constant exists and that it can be computed for a platform, as plans have a finite duration and in most practical systems only a finite number of transitions can be taken in a given time.

Encoding-based approach We will now describe our SMT encoding of the PAMP problem (Fig. 2). Consider a temporal planning problem Π, a timed automaton $\mathcal{T}$ modeling the platform, a set of *bad* states B, and a bound h on the length of the plan. We assume that $\mathcal{T}$ contains a global clock γ with initial value 0, that is never reset in any transition. The encoding represents two distinct traces: a *plan trace* with h timed steps, and a *platform trace* with κh timed steps. In each step of a plan trace at most one snap action can be applied (as we discussed in the previous section).

We start by defining the variables of our encoding. For every step $i \in \{1, \ldots, h\}$ of the plan trace, we use the real variable t_i to denote the time associated to step i; for every action a, we use the Boolean variable a_i to denote whether the action a is started at step i, and the real variable d^a_i to represent the duration of action a when started at step i. For every step $i \in \{1, \ldots, \kappa h\}$ of the platform trace, we use the variable l_i to denote the location of $\mathcal{T}$ at step i; for every clock c, we use the variable c_i to represent the value of clock c at step i (the value of the global clock γ at step i is γ_i); finally, for every action a, we use the Boolean variable $\tau^{a_\vdash}_i$

$$\Phi_h : \exists \vec{t}, \vec{a}, \vec{d}.\ \textsc{PlanValid}_{\Pi}\left(\vec{t}, \vec{a}, \vec{d}, h\right) \wedge$$

$$\forall \vec{l}, \vec{c}.\ \bigwedge_{i=0}^{h-1}\Big(\textsc{TraceValid}_{\mathcal{T}}\left(\vec{l}, \vec{c}, h, \kappa\right) \wedge \textsc{compliant}_{\mathcal{T}}\left(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, i, \kappa\right) \rightarrow \textsc{applicable}_{\mathcal{T}}\left(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, i+1, \kappa\right)\Big) \wedge$$

$$\Big(\textsc{TraceValid}_{\mathcal{T}}\left(\vec{l}, \vec{c}, h, \kappa\right) \wedge \textsc{compliant}_{\mathcal{T}}\left(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, h, \kappa\right) \rightarrow \textsc{safety}_{\mathcal{T}}\left(\vec{l}, \vec{c}, B, h, \kappa\right)\Big)$$

$$\textsc{TraceValid}_{\mathcal{T}}\left(\vec{l}, \vec{c}, h, \kappa\right) : \textsc{Init}_{\mathcal{T}}\left(\vec{l}, \vec{c}, 1\right) \wedge \bigwedge_{i=2}^{\kappa h} \textsc{Trans}_{\mathcal{T}}\left(\vec{l}, \vec{c}, i-1, i\right) \qquad \textsc{safety}_{\mathcal{T}}\left(\vec{l}, \vec{c}, B, h, \kappa\right) : \bigwedge_{i=1}^{\kappa h}\left(\gamma_i \leq t_h \rightarrow \neg \textsc{bad}\left(\vec{l}, \vec{c}, B, i\right)\right)$$

$$\textsc{compliant}_{\mathcal{T}}\left(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, h, \kappa\right) : \quad \bigwedge_{a\in A}\bigwedge_{i=1}^{h}\left(a_i \rightarrow \bigvee_{j=1}^{\kappa h}\left(\tau_j^{a_\vdash} \wedge \gamma_j = t_i \wedge \bigwedge_{j'\in\{1,\ldots,\kappa h\},\, j'\neq j}\left(\gamma_{j'} = t_i \rightarrow \neg\tau_{j'}^{a_\vdash}\right)\right)\right)$$

$$\wedge \bigwedge_{a\in A}\bigwedge_{s=1}^{h}\bigwedge_{i=s+1}^{h}\left((a_s \wedge t_s + d_s^a = t_i) \rightarrow \bigvee_{j=1}^{\kappa h}\left(\tau_j^{a_\dashv} \wedge \gamma_j = t_i \wedge \bigwedge_{j'\in\{1,\ldots,\kappa h\},\, j'\neq j}\left(\gamma_{j'} = t_i \rightarrow \neg\tau_{j'}^{a_\dashv}\right)\right)\right) \wedge$$

$$\bigwedge_{a\in A}\bigwedge_{i=1}^{\kappa h}\left(\tau_i^{a_\vdash} \rightarrow \bigvee_{j=1}^{h}(a_j \wedge t_j = \gamma_i)\right) \wedge \bigwedge_{a\in A}\bigwedge_{i=1}^{\kappa h}\left(\tau_i^{a_\dashv} \rightarrow \bigvee_{s=1}^{h}\bigvee_{j=s+1}^{h}(a_s \wedge t_s + d_s^a = t_j \wedge t_j = \gamma_i)\right)$$

$$\textsc{applicable}_{\mathcal{T}}\left(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, h, \kappa\right) : \quad \bigwedge_{a\in A}\left(a_h \rightarrow \bigwedge_{i=1}^{\kappa h}\left(\left(\gamma_i = t_h \wedge \bigwedge_{j=1}^{i-1}\left(\gamma_j < t_h \vee \neg\tau_j^{a_\vdash}\right)\right) \rightarrow \bigvee_{\delta=\langle l_i, g, \tau^{a_\vdash}, r, l_i'\rangle\in\Delta} \textsc{enabled}(\vec{l}, \vec{c}, \delta, i)\right)\right) \wedge$$

$$\bigwedge_{a\in A}\bigwedge_{s=1}^{h-1}\left(a_s \wedge t_s + d_s^a = t_h \rightarrow \bigwedge_{i=1}^{\kappa h}\left(\left(\gamma_i = t_h \wedge \bigwedge_{j=1}^{i-1}\left(\gamma_j < t_h \vee \neg\tau_j^{a_\dashv}\right)\right) \rightarrow \bigvee_{\delta=\langle l_i, g, \tau^{a_\dashv}, r, l_i'\rangle\in\Delta} \textsc{enabled}(\vec{l}, \vec{c}, \delta, i)\right)\right)$$

Figure 2: The bounded encoding of the platform-aware planning problem.

(respectively $\tau_i^{a_\dashv}$) to denote whether a transition with label $\tau^{a_\vdash}$ (respectively $\tau^{a_\dashv}$) will be taken by $\mathcal{T}$ at step i.

We will use the notation $\vec{t}$ to denote the set of variables of the form t_i, and analogously for $\vec{a}$, $\vec{d}$, $\vec{l}$ and $\vec{c}$.

The formula Φ_h represents time-triggered plans of length up to h that satisfy Definition 13: the plan must be a valid solution for Π ($\textsc{PlanValid}_{\Pi}$); for all possible traces of $\mathcal{T}$, the plan must be executable, i.e. for all plan prefixes i from 0 to $h-1$, if a trace of $\mathcal{T}$ is valid ($\textsc{TraceValid}_{\mathcal{T}}$) and all the snap actions up to i have been applied ($\textsc{compliant}_{\mathcal{T}}$), then the $(i+1)$-th snap action will be applicable at the prescribed time ($\textsc{applicable}_{\mathcal{T}}$); finally, for all possible traces of $\mathcal{T}$, all the states of $\mathcal{T}$ that can be visited by executing the plan do not intersect the set of bad states B, i.e. if a trace of $\mathcal{T}$ is valid and all the snap actions of the plan have been applied, then the safety property is satisfied ($\textsc{safety}_{\mathcal{T}}$).

The formula $\textsc{PlanValid}_{\Pi}$ is a standard bounded encoding of temporal planning (Shin and Davis 2005), which we omit for the sake of brevity. Similarly, the formula $\textsc{TraceValid}_{\mathcal{T}}$ is a standard unrolling of the transition relation of the timed automaton $\mathcal{T}$ up to step κh, where we denote with $\textsc{init}_{\mathcal{T}}$ the formula for the initial state of $\mathcal{T}$, and with $\textsc{trans}_{\mathcal{T}}$ the transition relation of $\mathcal{T}$.

The formula $\textsc{compliant}_{\mathcal{T}}$ encodes the fact that a platform trace applies all the snap actions of a plan up to step h, and that no transition corresponding to a snap action is triggered at the wrong time or without the action being present in the plan (it characterizes the traces that appear in the definition of $\text{Reachable}_{\mathcal{T}}$): when an action a is started at step i (a_i), there exists a step j in the platform trace where a transition with the corresponding label is triggered ($\tau_j^{a_\vdash}$), the value of the global clock corresponds to the time at which a is started ($\gamma_j = t_i$), and there are no multiple occurrences of the corresponding label at the same time; the same applies for actions ending at step i, which were started at a previous step s ($a_s \wedge t_s + d_s^a = t_i$); if a transition with label $a_\vdash$ is taken at step i ($\tau_i^{a_\vdash}$), then there must exist a step j in the plan trace at which a is started and the times are the same ($a_j \wedge t_j = \gamma_i$), and similarly for transitions with label $a_\dashv$.

The applicability of snap actions is encoded by $\textsc{applicable}_{\mathcal{T}}$. If the action a is started at step h, then for all the steps i in the platform trace where the value of the global clock corresponds to the time at which a is started and the corresponding transition has not already been taken in a previous step j at the current time ($\gamma_j < t_h \vee \neg\tau_j^{a_\vdash}$), there must exist a transition from the current location l_i with label $\tau^{a_\vdash}$ that is enabled (ENABLED encodes the fact that the guard is true under the current clock evaluation and that the invariant of the reached location is true after the necessary clocks are reset). The applicability of ends is handled similarly.

Finally, the formula $\textsc{safety}_{\mathcal{T}}$ states that for all the steps in the platform trace that occur before the end of the plan ($\gamma_i \leq t_h$), the current state is not included in the set of bad states B (BAD encodes the set $B \subseteq \mathcal{L} \times \mathbb{R}^{\mathcal{X}}$).

The overall procedure (PAMP-ENC) builds the formulae Φ_h for increasing bounds h and checks them with an SMT solver: if it returns UNSAT, then there is no safe and executable plan within bound h and the bound is increased; if a model is returned, it corresponds to a solution to the PAMP problem, as it satisfies the planning constraints and the executability and safety properties for all platform traces.

We now show the soundness and completeness of the ap-

proach. Here we provide proof sketches for the theorems, while the full details are included in the additional material.

Theorem 1 (Soundness and completeness of encoding-based algorithm)**.** *For every PAMP problem* $\Upsilon = \langle \Pi, \mathcal{T}, B \rangle$ *and every bound* κ*:*

1. *if* PAMP-ENC$(\Pi, \mathcal{T}, B, \kappa)$ *terminates and returns plan* π*, then* π *is a valid solution for* Υ *(soundness);*
2. *if there exists a solution for* Υ*, then* PAMP-ENC$(\Pi, \mathcal{T}, B, \kappa)$ *will eventually terminate and return a solution for* Υ *(completeness).*

Proof. (Sketch) (1) Suppose that the procedure returns plan π at step h. Let μ be the model of Φ_h from which π was extracted. We need to prove that π satisfies Definition 13:

1. First, since μ satisfies PLANVALID$_\Pi(\vec{t}, \vec{a}, \vec{d}, h)$, which is a standard bounded encoding of Π, the plan π extracted from μ is a solution for Π of length up to h.
2. Second, we need to prove that π is executable on $\mathcal{T}$, i.e. it satisfies the requirements of Definition 11. Let $r_0 = \langle l_0, u_0 \rangle$ be the initial state of $\mathcal{T}$ and let $\rho^\pi = \langle (t_1, e_1), \ldots, (t_n, e_n) \rangle$ be the ordered sequence of timed snap actions of π. Consider a prefix $i \in \{0, \ldots, n-1\}$ of ρ^π and a state $r = \langle l, u \rangle \in \text{ReachableAfter}_\mathcal{T}(r_0, \rho^\pi_i)$ such that $u(\gamma) = t_{i+1}$ and r is reachable within κh steps starting from r_0, i.e. there exists a run $r_0 \xrightarrow{d_1} \xrightarrow{\sigma_1} \ldots \xrightarrow{d_k} \xrightarrow{\sigma_k} r_k \equiv r$ with $k \leq \kappa h$. Then, it can be shown that e_{i+1} is applicable in r: the run reaching r together with the model μ satisfy TRACEVALID$_\mathcal{T}(\vec{l}, \vec{c}, h, \kappa)$ and COMPLIANT$_\mathcal{T}(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, i, \kappa)$; from the encoding Φ_h this implies that APPLICABLE$_\mathcal{T}(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, i+1, \kappa)$ is satisfied, and this implies that e_{i+1} is applicable in r. Therefore, by Definition 11, π is executable on $\mathcal{T}$.
3. Third, we need to prove that π is B-safe w.r.t. to $\mathcal{T}$, i.e. it satisfies the requirements of Definition 12. Let $r_0 = \langle l_0, u_0 \rangle$ be the initial state of $\mathcal{T}$ and let $\rho^\pi = \langle (t_1, e_1), \ldots, (t_n, e_n) \rangle$ be the ordered sequence of timed snap actions of π. Consider a state $r = \langle l, u \rangle \in \text{Reachable}_\mathcal{T}(r_0, \rho^\pi)$ such that $u(\gamma) \leq t_n$ and r is reachable within κh steps starting from r_0, i.e. there exists a run $r_0 \xrightarrow{d_1} \xrightarrow{\sigma_1} \ldots \xrightarrow{d_k} \xrightarrow{\sigma_k} r_k$, with $k \leq \kappa h$ and $r = r_i$ for some $i \in \{0, \ldots, k\}$. Then, it can be shown that $r \notin B$: the run reaching r together with the model μ satisfy TRACEVALID$_\mathcal{T}(\vec{l}, \vec{c}, h, \kappa)$ and COMPLIANT$_\mathcal{T}(\vec{t}, \vec{a}, \vec{d}, \vec{l}, \vec{c}, h, \kappa)$; from the encoding Φ_h this implies that SAFETY$_\mathcal{T}(\vec{l}, \vec{c}, B, h, \kappa)$ is satisfied, and this implies that $r \notin B$.

(2) Let π be the solution plan for Υ that exists by assumption. Since PLANVALID$_\Pi(\vec{t}, \vec{a}, \vec{d}, h)$ is a standard bounded encoding of the temporal planning problem Π with completeness guarantees, there exists a step h for which there is a model $\mu \models$ PLANVALID$_\Pi(\vec{t}, \vec{a}, \vec{d}, h)$ such that π can be extracted from it. We can then show that $\mu \models \Phi_h$, which implies that PAMP-ENC$(\Pi, \mathcal{T}, B, \kappa)$ terminates at a step lower or equal than h (because $\mu \models \Phi_h$). □

Abstraction-refinement approach In our second approach, based on an abstraction-refinement loop, we consider the planning problem and the validation problem separately: a temporal planner generates solution plans considering only the planning problem, and then the produced candidate plans are checked for executability and safety at the platform layer. Since we are considering time explicitly, it is not feasible to exclude single time-triggered plans at each validation check, as the planner, that is not aware of the platform constraints, can in most cases just slightly change the timing of the actions and the same problem would occur at the platform layer. Instead, at each failed validation check we want to exclude classes of plans, by determining that a certain sequence of discrete choices is infeasible, for any possible scheduling of the chosen snap actions.

For solving the planning problem, we rely on the TAMER temporal planner (Valentini, Micheli, and Cimatti 2019), which is a sound and complete approach for temporal planning that is able to return plans expressed as Simple Temporal Networks (STN) (Dechter, Meiri, and Pearl 1991): a returned solution π_{STN} is characterized by a fixed ordering of snap actions $e_1, \ldots, e_n \leftarrow$ PATH(π_{STN}), with each snap action e_i associated to a symbolic time t_i. The times are ordered increasingly, with additional constraints between pairs of start/end snap actions representing the duration constraints of the corresponding durative actions. The planning algorithm implements an explicit-state heuristic-search approach, that works by exploring all the possible ordered sequences of snap actions, and updating in each state a STN whenever a new snap action is added to the sequence. If the set of STN constraints of a state becomes infeasible, then it can be pruned, as it means that the chosen sequence of discrete events cannot be scheduled while respecting the temporal constraints of the problem. If a goal state is reached, then all the time-triggered plans that satisfy the STN constraints of that state are valid solution plans, and a specific solution can be extracted by solving the constraints.

The main idea of our approach is to validate on the platform the set of STN constraints π_{STN} produced by the planner, using an encoding similar to the one of the previous approach (Fig. 2). If there exists a solution to the STN constraints, that satisfies the executability and safety notions for all the platform traces, then it corresponds to an answer to the PAMP problem. Otherwise, we determine the shortest prefix $e_1, \ldots, e_i$ of the sequence of snap actions PATH(π_{STN}), such that by considering only the STN constraints of $e_1, \ldots, e_i$, there does not exist a way to schedule them while guaranteeing executability and safety for all platform traces. If such a prefix is found, it can be learned by the planner, and all the states that are found during exploration whose path starts with a learned prefix can be pruned.

The overall procedure is detailed in Algorithm 1. The planning problem Π is solved, and we obtain a set of solution plans π_{STN}, characterized by a fixed order of snap actions $e_1, \ldots, e_n \leftarrow$ PATH(π_{STN}), together with a set of temporal constraints between their associated times $t_1, \ldots, t_n$. The solution π_{STN} is then passed to CHECK, together with the set of bad states B. The CHECK procedure iterates over all the prefixes $i \in \{1, \ldots, n\}$, and builds the formula ψ^i_π, which is the subset of constraints of π_{STN} considering only $t_1, \ldots, t_i$ ($[\pi_{\text{STN}}]_i$ is the conjunction of all the con-

straints containing t_i and one of the times in $\{t_1, \ldots, t_{i-1}\}$). The constraints ψ_π^i are then used to produce the formula Φ_i, which is an encoding of the PAMP problem, considering only candidate plans represented by ψ_π^i: in the formula of Fig. 2, PLANVALID_Π is replaced with ψ_π^i, while the forall formula is simplified considering the specific discrete choices $e_1, \ldots, e_i$ that are made by each plan represented by ψ_π^i (for each step $j \in \{1, \ldots, i\}$, the truth value of all the variables a_i is known and can be substituted in the formula). The formula Φ_i is then provided to an SMT solver: if it is unsatisfiable, then we can deduce that the prefix $e_1, \ldots, e_i$ is not valid for the platform, for any possible scheduling of the snap actions that respects the planning constraints, and therefore this path can be "learned" by the planner and used for pruning; if it is satisfiable, then the next prefix can be considered, and if the whole plan was being considered then a final solution can be extracted from such a model.

Theorem 2 (Soundness and completeness of abstraction-refinement algorithm)**.** *For every PAMP problem* $\Upsilon = \langle \Pi, \mathcal{T}, B \rangle$ *and every bound* κ*:*

1. *if* PAMP-REF$(\Pi, \mathcal{T}, B, \kappa)$ *terminates and returns plan* π*, then* π *is a valid solution for* Υ*;*
2. *if there exists a solution for* Υ*, then* PAMP-REF$(\Pi, \mathcal{T}, B, \kappa)$ *will eventually terminate and return a solution for* Υ*.*

Proof. (Sketch) (1) Suppose that the procedure returns plan π. π is a valid solution to the planning problem Π, because of the soundness of the TAMER planner (Valentini, Micheli, and Cimatti 2019) and the correctness of the encoding of the STN constraints (that replace the formula $\text{PLANVALID}_\Pi(\vec{t}, \vec{a}, \vec{d}, h)$ in the Φ_h encoding). The plan then satisfies an analogous encoding of the forall subformula of Φ_h (which is simplied taking into account the specific action choices made by TAMER), and this guarantees the executability and safety properties (the proof follows the same reasoning of soundness in Theorem 1).
(2) If a solution π to Υ exists, then because of the completeness of the TAMER planner and the fact that the excluded prefixes do not satisfy the Φ_h encoding, the plan π will be eventually returned by TAMER (unless a different solution is found earlier for Υ). The plan then satisfies an analogous encoding of the forall subformula of Φ_h, which confirms the executability and safety properties of π (the proof follows the soundness in Theorem 1). □

Algorithm 1: Abstraction-refinement algorithm

```
1  procedure PAMP-REF(Π, 𝒯, B, κ)
2      bad_prefixes = {}
3      while True do
4          π_STN ← PLAN(Π, bad_prefixes)
5          pass, π ← CHECK(𝒯, π_STN, B)
6          if pass then return π
7          else
8              bad_prefixes ← bad_prefixes ∪ π
9  procedure CHECK(𝒯, π_STN, B)
10     e_1, ..., e_n ← PATH(π_STN)
11     ψ_π^0 ← ⊤
12     for i = 1 to n do
13         ψ_π^i ← ψ_π^{i-1} ∧ [π_STN]_i
14         Φ_i ← ENCODE(ψ_π^i, 𝒯, B, i, κ)
15         μ ← SOLVE(Φ_i)
16         if μ is UNSAT then return ⊥, (e_1, ..., e_i)
17         else
18             if i = n then return ⊤, EXTRACTPLAN(μ)
```

Related Work

In this paper, we define the PAMP problem, aiming at offering *guarantees* at system-level on the execution behaviors of the plan. To the best of our knowledge, this problem is novel. There is a wide literature on how to handle execution of plans in autonomous systems (Cashmore et al. 2015; Zanetti et al. 2023), but here the point is to model the interaction between a planner and the underlying platform and construct plans that are provably safe by construction.

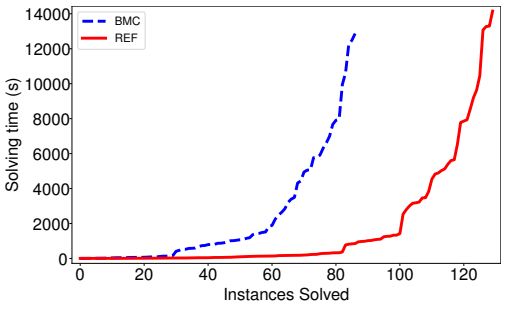
Bozzano, Cimatti, and Roveri (2021) propose a formal framework for on-board autonomy that relies on symbolic model-based reasoning, and integrates plan generation, plan execution, monitoring, fault detection identification and recovery, and run-time diagnosis functionalities. The controlled system is modeled as a finite-state non-deterministic planning problem enriched with resource estimation functions. In our case, we focus on a timed model for the system and the platform modeling is much richer than the estimation functions: we allow for generic timed automata models.

Viehmann, Hofmann, and Lakemeyer (2021) assume that an abstract sequential plan is given and model the platform layer as a timed automaton. A set of constraints expressed in Metric Temporal Logic (MTL) define the relationship between the two layers. The paper is limited to the problem of checking if there is one execution of the platform satisfying a given plan (it is an $\exists\exists$ quantification), while in our case, we provide a formal model for the plan generation part and tackle a plan generation problem that requires a universally-quantified validation on the platform ($\exists\forall$). Moreover, we differ on the interface between the high and low abstraction layers: we use the labels in the timed automata to model "commands" that are sent by the plan, while Viehmann, Hofmann, and Lakemeyer use MTL formulae to constrain the possible traces. Finally, we also consider executability constraints.

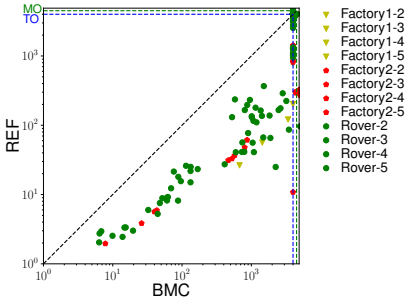
The problem that we address is also strongly related to conformant planning (Ghallab, Nau, and Traverso 2004), but in a temporal setting and for a model of the actions given by the timed automaton in the platform level. In fact, we are looking for a plan that is capable of succeeding irrespectively of the non-determinism of the platform. We are not aware of any paper concerning conformant temporal planning, but our encodings into SMT resemble the approach in (Cimatti, Roveri, and Bertoli 2004) for finite-state conformant planning. A key difference with respect to conformant planning is that in our case the non-determinism originates from the platform and can involve multiple steps not visible at planning level, whereas in conformant planning one assumes that either the initial state or action effects are non-deterministic (i.e., there is a "lockstep" between the planning

Algorithm / Domain	BMC	REF
Factory1-k2	4	9
Factory1-k3	1	2
Factory1-k4	0	0
Factory1-k5	0	0
Factory2-k2	10	10
Factory2-k3	3	6
Factory2-k4	0	3
Factory2-k5	0	0
Rover-k2	35	44
Rover-k3	18	25
Rover-k4	8	18
Rover-k5	8	13
Total	87	130

(a)



(b)



(c)

Figure 3: Experimental results: coverage table (a), cactus (b) and scatter (c) plots. The k values represent the bound on platform traces w.r.t. plan lengths. BMC is the encoding-based approach, while REF is the algorithm based on abstraction-refinement.

choices and the nondeterministic outcomes).

Experimental Evaluation

We developed a solver written in Python based on pySMT (Gario and Micheli 2015) implementing both the presented approaches. The solvers accept temporal planning problems written in either PDDL 2.1 or ANML, and platform models written in timed SMV (Cimatti et al. 2019), which allows to model TAs in a symbolic setting. We experimentally evaluated both approaches on two novel sets of benchmarks, ROVER and FACTORY, which are both available in the additional material. In ROVER, there are n locations $l_0, \ldots, l_{n-1}$ connected by edges, and a robot which is initially at location l_0. The robot can move between consecutive locations in 1 unit of time, while moving between non-consecutive locations takes 100 units of time. The robot can also communicate at each location. The goal of the planning problem is to communicate while at certain locations, and reach location l_{n-1} in the end. At the platform layer, modeled as a network of TAs, there is a task component (synchronized with the high-level communicate action) that controls a communication component: when a message is sent, the communication component moves to a *standby* location if no other message is sent within 30 units of time; if this happens, the task needs to resume the component by transitioning to the *resuming* location, before sending the next message. In this problem, we include a safety property by requiring that the platform never transitions to the *resuming* location, to avoid consuming excessive energy for the resumption process. Therefore, solution plans will be required to only travel between consecutive locations when the first message is sent, so that the communication component does not need to be resumed. We scale the instances by increasing the number of locations, by considering all the possible combinations of locations in which to send messages, and by increasing the bound κ.

FACTORY is the same domain of the running example shown in Fig. 1b and Fig. 1a. We consider two different ways of modeling the deadline: either at the planning layer (domain FACTORY1), by having a durative "Process" action that needs to be run in parallel with all other actions in the plan, or at the platform layer (domain FACTORY2), by having a component that synchronizes with the task associated with the "Work" action, and that disables the synchronization after the deadline has passed. The instances are scaled by increasing the number of required Work actions, by considering different deadlines, and by increasing the bound κ.

We performed all the experiments 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. The experimental results are shown in Fig. 3. We can observe that both approaches are effective at solving the tested benchmarks, with the approach based on abstraction-refinement having a wider coverage and faster solving times. This is expected, especially if the number of necessay refinement loops is low, as heuristic-search based planners are typically much faster at finding solution plans compared to encoding-based approaches, and checking the executability and safety of a STN plan is computationally much cheaper compared to combining the check with the full encoding of the planning problem. In the tested benchmarks, the number of necessary loops in the second approach ranged from 1 to 8. It is evident from the coverage table that the bound κ on the length of the platform traces greatly influences the performance of both approaches, as platform traces are universally quantified in the encoding formula. A future direction is to try to check the executability and safety notions using an unbounded technique, possibly proving the non-existence of bad traces.

Conclusions

In this paper, we formally defined the "Platform-Aware Mission Planning" problem, motivated by the need of synthesizing plans that not only achieve the mission objectives, but also ensure executability and the satisfaction of safety properties during execution. We devised an amalgamated method and a decomposition approach that can solve the problem, and showed the superiority of the latter experimentally.

As future work, we plan to generalize our model to the case of hybrid automata, allowing the representation of continuous behaviors and resources in the platform. Moreover, we are interested in other problems that can be defined in the formal framework we proposed, such as synthesizing plans that guarantee other formal properties like diagnosability.

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).

References

Alur, R.; and Dill, D. L. 1994. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2): 183–235.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability Modulo Theories. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, 825–885. IOS Press. ISBN 978-1-58603-929-5.

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).

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 333–341. AAAI Press.

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.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1): 127–206.

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*.

Gario, M.; and Micheli, A. 2015. pySMT: a Solver-Agnostic Library for Fast Prototyping of SMT-Based Algorithms. In *SMT Workshop*.

Gat, E. 1998. On three-layer architectures. *Artificial intelligence and mobile robots*, 195: 210.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier. ISBN 978-1-55860-856-6.

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.

Gigante, N.; Micheli, A.; and Scala, E. 2022. On the Expressive Power of Intermediate and Conditional Effects in Temporal Planning. In Kern-Isberner, G.; Lakemeyer, G.; and Meyer, T., eds., *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022*.

Shin, J.-A.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*.

Valentini, A.; Micheli, A.; and Cimatti, A. 2019. Temporal Planning with Intermediate Conditions and Effects. *CoRR*, abs/1909.11581.

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.

Zanetti, A.; Moro, D. D.; Vreto, R.; Robol, M.; Roveri, M.; and Giorgini, P. 2023. Implementing BDI Continual Temporal Planning for Robotic Agents. In *IEEE International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2023, Venice, Italy, October 26-29, 2023*, 378–382. IEEE.