Platform-Aware Mission Planning with Task-Level Contracts

Stefan Panjkovic[1,2] , **Alessandro Cimatti**[1] , **Inigo Incer**[3] , **Andrea Micheli**[1] and **Stefano Tonetta**[1]
[1]Fondazione Bruno Kessler, Trento, Italy
[2]University of Trento, Italy
[3]University of Michigan, USA
{spanjkovic, cimatti}@fbk.eu, iir@umich.edu, {amicheli, tonettas}@fbk.eu

Abstract

Automated temporal planning is used to synthesize courses of action for a deterministic abstraction of a system; the produced plans are often modeled as schedules of *tasks* to be executed on the real system under control. A key problem with adopting this architecture is how to ensure the plan is executable and goal-reaching on the real system, which can be arbitrarily complex and possibly non-deterministic.

In this paper, we propose a framework to exploit *task-level contracts*, expressed as assumptions and guarantees at the beginning and end of tasks, to automatically synthesize plans that are guaranteed to be correct on any system satisfying the contracts. Our framework combines a temporal planner to generate candidate plans and a contract reasoner to instantiate and verify the contracts associated with the plan. If the plan is found invalid, we refine the planning problem until a valid plan is found. We present an experimental evaluation on a realistic case-study and on several synthetic problems, showing the applicability of the approach.

1 Introduction

Temporal planning is the problem of synthesizing a schedule of durative actions to reach a desired goal from an initial state, in a domain description subject to temporal constraints. A temporal planning domain, described in languages such as PDDL 2.1 [Fox and Long, 2003] and ANML [Smith *et al.*, 2008], is deterministic, in the sense that an executable plan is associated with exactly one run.

Recently, Panjkovic *et al.* (2025a; 2025b) introduced the "Platform-Aware Mission Planning" (PAMP) problem, an extension to temporal planning where a *mission-level* temporal domain is coupled with the description of an execution *platform* (in the form, for example, of a timed automaton or a timed transition system) upon which mission-level plans can be executed. Notably, the platform can contain a rich set of features, may be nondeterministic, and subject to integrity constraints, so that a valid temporal plan at mission level can be associated with multiple (and possibly invalid) runs at the platform level.

A PAMP problem consists of synthesizing a valid, goal-achieving temporal plan at the mission level that is at the same time guaranteed to be executable and safe w.r.t. an invariant property at the platform level. The PAMP framework integrates, while keeping them separate, (i) an easily solvable, deterministic temporal planning problem, and (ii) the non-determinism in the execution of the plan on the platform, subject to safety constraints. The algorithms proposed in [Panjkovic *et al.*, 2025a; Panjkovic *et al.*, 2025b] leverage the separation of the mission and platform levels by tightly combining an enumeration of temporal mission plans with a platform validity check that corresponds to a model checking problem at the platform level. Despite the effectiveness of the optimizations proposed in [Panjkovic *et al.*, 2025a], solving a PAMP problem in its general form turns out to be significantly more challenging than classical temporal planning.

In this paper, we propose an alternative approach based on the idea of limiting the freedom in which the platform can be executed. The mission plan is no longer allowed to be freely connected to the behavioral model of a platform, with arbitrary parallel and overlapping temporal actions. Rather, the operation of the platform is restricted to a set of predefined *atomic tasks*, each associated with a durative action or a composition of durative actions. The tasks are atomic in the sense that they are activated sequentially and cannot overlap. Each task is associated with (i.e., abstracted into) a *contract*, an assumption-guarantee pair over a set of platform and timing variables. The assumptions indicate the platform-level preconditions, i.e., the set of configurations in which the task can be safely activated, while the guarantees represent the post-conditions of the task. This contract view is intended to capture all the non-deterministic platform executions under restricted conditions. The PAMP problem is therefore reduced to two key subproblems. The first one, named Platform-Aware Mission Planning with Task-Level Contracts (PAMP-TLC), is the problem of finding a temporal plan that is valid according to the temporal planning model and that yields a robust composition of task-level contracts. The second is the problem of ensuring that the platform model satisfies specific requirements induced by the task contracts, which intuitively means that the relevant aspects of the platform are properly characterized in the contracts of the tasks. This ensures that a solution of PAMP-TLC is also a solution for the original PAMP problem.

The approach is structured in the following contributions. We formalize the PAMP-TLC problem in an algebraic contract framework [Incer *et al.*, 2025], and provide an efficient algorithm for the synthesis of valid PAMP-TLC solutions that combines temporal planning and contract reasoning. We characterize the proof obligations a platform must satisfy for a PAMP-TLC solution to be a PAMP solution, and propose an automated approach to verify them on a given platform model. The approach is implemented and experimentally evaluated on a realistic case study taken from [Rouquette *et al.*, 2023] and on several synthetic benchmarks. The comparison with the PAMP algorithm in [Panjkovic *et al.*, 2025a] demonstrates that, despite the introduced limitations in the activation of the platform, the new approach, equipped with the proper task abstractions, retains the ability to solve all the problems in the benchmark set, while at the same time demonstrating superior performance.

Related Work In addition to [Panjkovic *et al.*, 2025a], which we will compare against in the rest of this paper, this work has some similarities with conformant planning [Ghallab *et al.*, 2004], where there is no runtime observation and actions can have non-deterministic effects. To the best of our knowledge, the temporal case is not addressed in the conformant planning literature, and our setting differs in that we consider two models at different layers of abstraction. Viehmann *et al.* [2021] consider the problem of checking whether an abstract sequential plan can be executed on a platform modeled as a timed automaton, where the relationship between the high-level and low-level models is expressed with Metric Temporal Logic (MTL) constraints. However, this work is limited to verifying whether there exists a single platform execution compliant with the plan, while we consider plan robustness w.r.t. all possible platform behaviors. Bozzano *et al.* [2021] propose a formal autonomy framework integrating plan generation, plan execution, monitoring and fault detection identification and recovery that is based on symbolic model-based reasoning. The controlled system is represented as a finite-state non-deterministic planning problem, with resource estimation functions. A form of mission planning based on contracts was previously considered in [Mallozzi *et al.*, 2023], where, given a library of components, the goal is to produce an implementation that satisfies a top-level LTL property, expressed as a contract. In this approach, the components in the library are expressed as LTL contracts, and timing aspects are not considered. There are also several works in the literature on plan execution, where the model of the underlying environment is known, like PRS-style architectures [Ingrand *et al.*, 1992], SkiROS [Rovida *et al.*, 2017] and approaches based on hierarchical task-oriented refinement methods [Patra *et al.*, 2021]. However, these works are fundamentally different from our setting as they are online methods, while we address the problem of providing safety and executability guarantees at planning time.

2 Background

In this section, we report the background notions for the rest of the paper.

Temporal Planning. We define the syntax of a temporal planning problem, borrowing from [Gigante *et al.*, 2022].

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

Intuitively, the solution to a planning problem is a schedule of actions, each respecting its duration bounds, such that starting from the initial state and applying the effects of every action at the right time, all action conditions are satisfied and the goal is reached in the final state. For the sake of brevity, we do not report the full planning semantics here (a complete account can be found in [Gigante *et al.*, 2022]). We formalize the structure of a plan that will be used for our definitions.

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.*

Definition 3 (Set of timed snap actions)**.** *A timed snap action (TSA) is a pair (t, h), where $t \in \mathbb{Q}_{\geq 0}$ and h is a snap action. Given a plan $\pi = \{\langle a_1, t_1, d_1\rangle, \ldots, \langle a_n, t_n, d_n\rangle\}$, the set of TSAs of π is defined as: $H(\pi) = \{(t_1, a_{1\vdash}), (t_1 + d_1, a_{1\dashv}), \ldots, (t_n, a_{n\vdash}), (t_n + d_n, a_{n\dashv})\}$.*

An important property of a planning problem is whether action self-overlapping is allowed. Gigante *et al.* (2022) proved that temporal planning without action self-overlapping is PSPACE-complete, and we adopt this semantics (i.e., we disallow self-overlapping) for this paper.

Definition 4 (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 $i, j \in \{1, \ldots, n\}$ with $i \neq j$ such that $a_i = a_j$ and $t_i \leq t_j < t_i + d_i$.*

Symbolic Timed Transition Systems. To formalize the comparison between this work and [Panjkovic *et al.*, 2025a], we report here the syntax of a symbolic timed transition system.

Definition 5 (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.*

For brevity, we refer the reader to [Panjkovic *et al.*, 2025a] for the formal semantics. Here, it suffices to say that a TTS models a dynamical system that performs two types of transitions: discrete transitions, where the variables in V change according to the transition relation $T(V, \Sigma, V')$, where V' denotes the values of the variables after the transition is taken; delay transitions, where the clocks increase their values by the same quantity δ and all other variables keep their value.

Given a TTS $\mathcal{T}$, there exist off-the-shelf model-checking algorithms [Cimatti *et al.*, 2019] for proving and disproving properties expressed in Linear Temporal Logic (LTL) [Pnueli, 1977]. In this paper, we use some simple LTL checks as a way to link the PAMP problem to our PAMP-TLC framework.

IO Contracts. In the following, we consider formulas of a given logic L. The logic comes with a background theory and a corresponding satisfaction relation $\models$. Thus, for any formula φ over a set of variables V and any assignment μ to V, we write $\mu \models \varphi$ to denote that φ is satisfied under the assignment μ. In this paper, we focus on the $\mathcal{QF\_LRA}$ (Quantifier-Free Linear Real Arithmetic) logic. In essence, $\mathcal{QF\_LRA}$ formulas are Boolean combinations of inequalities between linear polynomials over real variables: a term is of the form $\sum_i a_i x_i$, where every x_i is a real variable and every a_i is a rational constant; an atom is either a Boolean variable or an expression of the form $\varphi \bowtie c$, where φ is a term, $\bowtie \in \{>, <, \geq, \leq, \neq, =\}$, and c is a rational constant; finally, a formula in $\mathcal{QF\_LRA}$ is a conjunction ($\wedge$), disjunction ($\vee$) or negation ($\neg$) of atoms. Given a formula φ, we denote with $\text{Vars}(\varphi)$ the set of variables appearing in φ. An assignment $\mu : \text{Vars}(\varphi) \to \mathbb{B} \cup \mathbb{R}$ to the variables of φ satisfies φ, denoted $\mu \models \varphi$, if it evaluates φ to true.

In this setting, an Input-Output (IO) Contract is a pair of formulae, denoting an input assumption on some variables I and a guarantee defined over I and output variables O. The following definition adapted from [Incer *et al.*, 2025] provides the syntax for an IO contract.

Definition 6 (IO Contract)**.** *Let $V = V_B \cup V_R$ be a finite set of variables with domain $\mathbb{B} \cup \mathbb{R}$. An IO contract in $\mathcal{QF\_LRA}$ is a tuple $\mathcal{C} = (I, O, a, g)$, where $I, O \subseteq V$ are disjoint sets of input and output variables respectively, and a, g are $\mathcal{QF\_LRA}$ formulas representing the assumptions and guarantees of $\mathcal{C}$ respectively. The assumptions of IO contracts only depend on input variables ($\text{Vars}(a) \subseteq I$), and the guarantees only depend on input and output variables ($\text{Vars}(g) \subseteq I \cup O$).*

An environment of a contract $\mathcal{C} = (I, O, a, g)$ is a set of assignments E to the variables in I such that for each $\mu_E \in E$, $\mu_E : I \to \mathbb{B} \cup \mathbb{R}$, we have that $\mu_E \models a$. An implementation is a set of assignments M to the variables in $I \cup O$ such that for each $\mu_M \in M$, $\mu_M : I \cup O \to \mathbb{B} \cup \mathbb{R}$, we have that $\mu_M \models (a \to g)$.

We define the notions of refinement for IO contracts, and the operations of composition, quotient and merging. We adapt the definitions provided in [Benveniste *et al.*, 2018; Incer *et al.*, 2025] to the case of IO contracts in $\mathcal{QF\_LRA}$.

Definition 7 (Refinement)**.** *Given two contracts $\mathcal{C}_1 = (I_1, O_1, a_1, g_1)$ and $\mathcal{C}_2 = (I_2, O_2, a_2, g_2)$, we say that $\mathcal{C}_1$ is a refinement of $\mathcal{C}_2$ (written $\mathcal{C}_1 \leq \mathcal{C}_2$), or equivalently that $\mathcal{C}_2$ is a relaxation of $\mathcal{C}_1$, if every environment of $\mathcal{C}_2$ is an environment of $\mathcal{C}_1$ and every implementation of $\mathcal{C}_1$ is an implementation of $\mathcal{C}_2$: $\mathcal{C}_1 \leq \mathcal{C}_2$ iff $(a_2 \to a_1) \wedge ((a_1 \to g_1) \to (a_2 \to g_2))$ is valid.*

The operation of composition returns a specification for a system obtained by composing implementations of the contracts being composed.

Definition 8 (Composition)**.** *Let $\mathcal{C}_1 = (I_1, O_1, a_1, g_1)$ and $\mathcal{C}_2 = (I_2, O_2, a_2, g_2)$ be two IO contracts, such that $I_1 \cap O_2 = \emptyset$. A composition of $\mathcal{C}_1$ and $\mathcal{C}_2$ is a contract $\mathcal{C}_c = (I_c, O_c, a_c, g_c)$ such that:*

1. *$I_c = (I_1 \cup I_2) \setminus O_1$;*
2. *$O_c = O_1 \cup O_2$;*
3. *Combining respective implementations of each contract yields an implementation for the composition: $((a_1 \to g_1) \wedge (a_2 \to g_2)) \to (a_c \to g_c)$ is valid;*
4. *Combining an environment for the resulting composition with an implementation for $\mathcal{C}_2$ yields an environment for $\mathcal{C}_1$ and vice-versa: $(a_c \wedge (a_2 \to g_2)) \to a_1$ and $(a_c \wedge (a_1 \to g_1)) \to a_2$ is valid.*

The quotient operation determines the specification of a subcomponent which, when composed with the provided contract, refines a top-level specification.

Definition 9 (Quotient)**.** *Let $\mathcal{C}_1 = (I_1, O_1, a_1, g_1)$ and $\mathcal{C}_2 = (I_2, O_2, a_2, g_2)$ be two IO contracts, such that $I_1 \cap I_2 = \emptyset$ and $O_2 \subseteq O_1$. A quotient of $\mathcal{C}_1$ by $\mathcal{C}_2$ is a contract $\mathcal{C}_q = (I_q, O_q, a_q, g_q)$ such that:*

1. *$I_q = I_1$;*
2. *$O_q = (O_1 \setminus O_2) \cup I_2$;*
3. *The composition of the quotient with the divisor is a refinement of the dividend: $C_q \parallel \mathcal{C}_2 \leq \mathcal{C}_1$.*

The composition and quotient operations can be implemented in several ways [Benveniste *et al.*, 2018; Incer *et al.*, 2025]. In [Benveniste *et al.*, 2018] the composition between two contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as the Greatest Lower Bound w.r.t. refinement order of all the contracts that satisfy Definition 8 (i.e. the *largest* contract that is a refinement of all the contracts that are a composition of $\mathcal{C}_1$ and $\mathcal{C}_2$), while the quotient is the Least Upper Bound of all the contracts satisfying Definition 9 (i.e. the *smallest* contract that is a relaxation of all the contracts that are a quotient of $\mathcal{C}_1$ by $\mathcal{C}_2$). Several approaches compute a relaxation or refinement of the exact composition or quotient respectively, for efficiency or representational reasons [Incer *et al.*, 2025]. In this paper, we assume that we are given a procedure that computes any composition between two contracts satisfying Definition 8 (denoted with $\mathcal{C}_1 \parallel \mathcal{C}_2$), and a procedure computing the quotient of two contracts satisfying Definition 9 (denoted with $\mathcal{C}_1 / \mathcal{C}_2$). We do not make any assumption on the *strength* of the computed contracts, and we will discuss in Section 4 the impact of this on the overall approach.

One additional operation, *merging*, is used to fuse into a single contract multiple contracts that describe distinct aspects of an object under specification.

Definition 10 (Merging)**.** *Let $\mathcal{C}_1 = (I_1, O_1, a_1, g_1)$ and $\mathcal{C}_2 = (I_2, O_2, a_2, g_2)$ be two IO contracts. The merge of $\mathcal{C}_1$ and $\mathcal{C}_2$ is a contract $\mathcal{C}_1 \bullet \mathcal{C}_2 = (I_m, O_m, a_m, g_m)$ such that $I_m = I_1 \cup I_2$, $O_m = O_1 \cup O_2$, $a_m = a_1 \wedge a_2$, and $g_m = g_1 \wedge g_2$.*

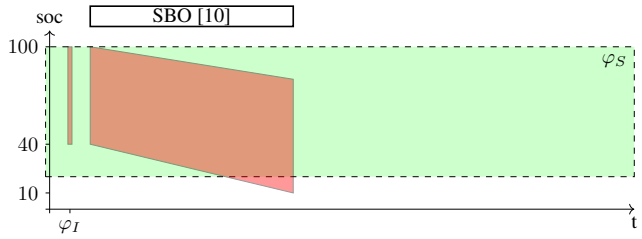

Figure 1: An invalid plan for the example PAMP-TLC problem, where φ_S can be violated.

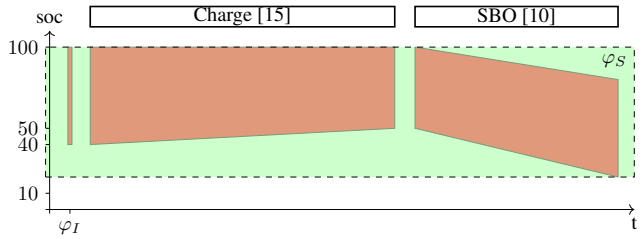

Figure 2: A solution plan for the example PAMP-TLC problem.

3 PAMP with Task-Level Contracts

In this section, we present the formalization of the Platform-Aware Mission Planning with Task-Level Contracts (PAMP-TLC) problem. We couple a high-level temporal planning problem with a low-level model defined using Input-Output (IO) Contracts. In our model, every action of the planning problem and every allowed parallel interleaving of actions is modeled with a contract, describing the assumptions and the guarantees of applying these high-level events on the platform and environment, in terms of low-level variables. Note that we assume a finite set of allowed parallel compositions of planning actions, because at this level of abstraction, composing the contracts of parallel tasks highly depends on the platform specifics.

The problem that we consider is the generation of a solution plan for the planning problem, such that for every possible outcome of its execution, as described by the contracts associated to the actions, a safety property is maintained.

Example. *Consider a setting in which a spacecraft needs to take some observations of an asteroid. We have a very simple temporal planning problem with durative actions SBO (Small Body Observation) and CHARGE. The SBO action sets a boolean variable obs to true, meaning that the asteroid was observed, and the goal of the planning problem is to have obs set to true. The CHARGE action affects some other variables which are not related to the preconditions of SBO. Suppose that these actions are associated to the following contracts, modeling the fact that there is a low-level soc variable (state of charge), and that SBO and CHARGE have assumptions and guarantees on it:*

- $c_{SBO} = (I_{SBO}, O_{SBO}, a_{SBO}, g_{SBO})$, *where*
  - $I_{SBO} = \{t_1, t_2, soc\}$
  - $O_{SBO} = \{soc'\}$
  - $a_{SBO} \coloneqq (t_2 - t_1 \geq 0) \wedge (soc \geq 3(t_2 - t_1))$
  - $g_{SBO} \coloneqq 2(t_2 - t_1) \leq soc - soc' \leq 3(t_2 - t_1)$
- $c_{CHARGE} = (I_{CHARGE}, O_{CHARGE}, a_{CHARGE}, g_{CHARGE})$, *where*
  - $I_{CHARGE} = \{t_1, t_2, soc\}$
  - $O_{CHARGE} = \{soc'\}$
  - $a_{CHARGE} \coloneqq (t_2 - t_1 \geq 0) \wedge (soc \geq 0)$
  - $g_{CHARGE} \coloneqq \frac{2}{3}(t_2 - t_1) \leq soc' - soc \leq 2(t_2 - t_1)$

In both contracts, the input variables t_1 and t_2 denote the starting time and the ending time of the action, respectively. Basically, c_{SBO} specifies that the duration of the action must be non-negative, and that the initial state of charge must be above $3(t_2 - t_1)$. The guarantee after applying the SBO action is that the decrease in the state of charge will be between $2(t_2 - t_1)$ and $3(t_2 - t_1)$. Instead, the contract c_{CHARGE} guarantees that after applying the CHARGE action, the state of charge will be increased by an amount between $\frac{2}{3}(t_2 - t_1)$ and $2(t_2 - t_1)$. The safety property that we consider is that the state of charge cannot go below 20: $soc \geq 20$.

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. Let Seq_A denote the set of all the totally ordered sequences of snap actions from A. Let V be a finite set of Boolean and real variables, and let V' be the set of primed variables from V. Let $T = \{\tau_1, \tau_2, \ldots\}$ be a set of *time variables* with domain in $\mathbb{R}$. Let $\mathcal{C}$ be a set of IO contracts defined on V, V' and T, where the input variables are from $V \cup T$ and the output variables are from V'. Given a contract $c = (I, O, a, g) \in \mathcal{C}$, we denote with $T_c \subseteq T$ the set of time variables appearing in I.

In our formalization, we associate with a partial function λ a contract to each durative action, and to each allowed parallel interleaving of actions, which is represented by a sequence of start and end snap actions (a single durative action A is represented by the sequence $\langle A_\vdash, A_\dashv \rangle$, where $A_\vdash$ and $A_\dashv$ are the start and end snap actions of A respectively).

Formally, let $\lambda : \text{Seq}_A \rightharpoonup \mathcal{C}$ be a partial function associating sequences of snap actions to contracts in $\mathcal{C}$, such that if $\lambda(\langle h_1, \ldots, h_n \rangle) = c$ then $T_c = \{\tau_1, \ldots, \tau_n\} \subseteq T$ (c has one time variable for each snap action in the sequence).

Given a plan π, we want to identify the maximal groups of parallel actions appearing in it; the plan π is then a sequence of these groups, each associated to a contract by the function λ. We formalize this concept as follows. Let $\pi = \{\langle a_1, t_1, d_1 \rangle, \ldots, \langle a_n, t_n, d_n \rangle\}$ be a plan for Π. We say that a value $t \in \mathbb{R}$ is an *empty time* in π if there doesn't exist $i \in \{1, \ldots, n\}$ such that $t_i \leq t \leq t_i + d_i$. 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 $i \in \{1, \ldots, 2n-1\}$. We say that $\sigma^\pi = \langle (t'_i, e_i), \ldots, (t'_j, e_j) \rangle$ is a *maximal overlapping subsequence* of ρ^π if: 1) there is no empty time in $[t'_i, t'_j]$; 2) $t'_i - \varepsilon$ and $t'_j + \varepsilon$ are empty times for every $\varepsilon \in (0, \delta]$, for a $\delta > 0$. We indicate with $\Sigma^\pi = \langle \sigma^\pi_1, \ldots, \sigma^\pi_r \rangle$, where $\sigma^\pi_i = \langle (t^1_i, e^1_i), \ldots, (t^{|\sigma^\pi_i|}_i, e^{|\sigma^\pi_i|}_i) \rangle$, the sequence of maximal overlapping subsequences of π.

Given a contract $c = (I, O, a, g)$ with $T_c = \{\tau_1, \ldots, \tau_n\}$, we denote with $[c]^i_j$ the contract that is obtained by renaming the variables $\tau_1, \ldots, \tau_n$ to $\tau_i, \ldots, \tau_{i+n-1}$, all the variables $v \in I \setminus T_c$ to v_j and all the variables $v \in O$ to v_{j+1}.

Now, we can define the top-level contract associated to a plan π by composing the contracts associated to the maxi-

mal overlapping subsequences in Σ^π. Before performing the composition, we set the time variables appearing in the contracts to the values specified in the plan (by merging the contracts with a contract encoding the specific timings of the snap actions in π), and we rename the variables in such a way that the output variables of one contract become the input variables of the next contract in the sequence.

Formally, we define the contract c^π_λ associated with π and the mapping function λ, as $c^\pi_\lambda = [\Phi_1 \bullet \Delta_1]^{k_1}_1 \parallel \ldots \parallel [\Phi_r \bullet \Delta_r]^{k_r}_r$, where $k_i = 1 + \sum_{j=1}^{i-1} |\sigma^\pi_j|$, $\Phi_i = \lambda(\sigma^\pi_i)$, and $\Delta_i = (I_i, O_i, a_i, g_i)$ is an IO contract such that $I_i = \{\tau_1, \ldots, \tau_{|\sigma^\pi_i|}\}$, $O_i = \emptyset$, $a_i = \bigwedge_{j=1}^{|\sigma^\pi_i|} \tau_j = t^j_i$ and $g_i = \top$.

Example. *Consider the plan* $\pi = \{\langle \mathit{CHARGE}, 0, 15\rangle, \langle \mathit{SBO}, 16, 10\rangle\}$. *In this case, since there is no parallelism, we simply have* $\Sigma^\pi = \langle \sigma^\pi_1, \sigma^\pi_2\rangle$, *where* $\sigma^\pi_1 = \langle (0, \mathit{CHARGE}_\vdash), (15, \mathit{CHARGE}_\dashv)\rangle$ *and* $\sigma^\pi_2 = \langle (16, \mathit{SBO}_\vdash), (26, \mathit{SBO}_\dashv)\rangle$. *We have* $\Phi_1 = \lambda(\sigma^\pi_1) = c_{\mathit{CHARGE}}$ *and* $\Phi_2 = \lambda(\sigma^\pi_2) = c_{\mathit{SBO}}$. *By merging with* Δ_1 *and* Δ_2, *we add the assumptions* $(t_1 = 0 \wedge t_2 = 15)$ *and* $(t_1 = 16 \wedge t_2 = 26)$ *to* c_{CHARGE} *and* c_{SBO} *respectively. In the contract* $[\Phi_1 \bullet \Delta_1]^{k_1}_1$, *we perform the renamings* $\mathit{soc} \mapsto \mathit{soc}_1$ *and* $\mathit{soc}' \mapsto \mathit{soc}_2$, *while in the contract* $[\Phi_2 \bullet \Delta_2]^{k_2}_2$ *we perform the renamings* $t_1 \mapsto t_3$, $t_2 \mapsto t_4$, $\mathit{soc} \mapsto \mathit{soc}_2$ *and* $\mathit{soc}' \mapsto \mathit{soc}_3$. *One possible contract* $c^\pi_\lambda = (I^\pi_\lambda, O^\pi_\lambda, a^\pi_\lambda, g^\pi_\lambda)$ *that can be obtained by the composition (satisfying Definition 8) is the following:*

- $I^\pi_\lambda = \{t_1, t_2, t_3, t_4, \mathit{soc}_1\}$
- $O^\pi_\lambda = \{\mathit{soc}_2, \mathit{soc}_3\}$
- $a^\pi_\lambda \coloneqq (t_1 = 0) \wedge (t_2 = 15) \wedge (t_3 = 16) \wedge (t_4 = 26) \wedge \mathit{soc}_1 + \frac{2}{3}(t_2 - t_1) - 3(t_4 - t_3) \geq 0$
- $g^\pi_\lambda \coloneqq (\frac{2}{3}(t_2 - t_1) \leq \mathit{soc}_2 - \mathit{soc}_1 \leq 2(t_2 - t_1)) \wedge (2(t_4 - t_3) \leq \mathit{soc}_2 - \mathit{soc}_3 \leq 3(t_4 - t_3))$

Finally, we define the PAMP-TLC problem, where we want to generate a solution plan π for a temporal planning problem Π, such that the resulting top-level contract c^π_λ refines a contract c_S encoding the desired safety property φ_S. In the following definition, we use the notation $[\varphi]_i$, where φ is a formula in V, to denote the formula obtained by renaming every $v \in \mathrm{Vars}(\varphi)$ to v_i.

Definition 11 (PAMP-TLC)**.** *Let* V *be a set of Boolean and real variables, and let* V' *be the set of primed variables from* V. *Let* $T = \{\tau_1, \tau_2, \ldots\}$ *be a set of time variables with domain* $\mathbb{R}$. *A Platform-Aware Mission Planning with Task-Level Contracts (PAMP-TLC) problem is a tuple* $\Upsilon = \langle \Pi, \mathcal{C}, \lambda, \varphi_I, \varphi_S\rangle$, *where* $\Pi = \langle P, A, I, G\rangle$ *is a temporal planning problem,* $\mathcal{C}$ *is a set of IO contracts defined on variables* V, V' *and* T, $\lambda : \mathit{Seq}_A \to \mathcal{C}$ *is a function associating sequences of snap actions to contracts in* $\mathcal{C}$, $\varphi_I(V)$ *is a formula describing the initial state, and* $\varphi_S(V)$ *is a safety property. A solution for* Υ *is a plan* π *such that:*

1. π *is a valid solution plan for* Π*;*
2. $c^\pi_\lambda \leq c_S$, *where* $c_S = (I_S, O_S, a_S, g_S)$ *is the following IO contract:*
   - $I_S = \{v_1 : v \in V\}$*;*
   - $O_S = \{v_i : v \in V, 1 < i \leq |\Sigma^\pi| + 1\}$*;*
   - $a_S \coloneqq [\varphi_I]_1$*;*
   - $g_S \coloneqq \bigwedge_{i=1}^{|\Sigma^\pi|+1} [\varphi_S]_i$.

Example. *Consider an initial condition* $\varphi_I \coloneqq 40 \leq \mathit{soc} \leq 100$, *and the safety property* $\varphi_S \coloneqq \mathit{soc} \geq 20$. *The plan in which only the SBO action is applied is invalid, as starting from this initial state the contract* c_{SBO} *does not guarantee that* φ_S *is satisfied in the end, as shown in Fig. 1. On the other hand, the plan in Fig. 2 where we apply the CHARGE action before the SBO action is a valid solution for the PAMP-TLC problem, as the contract resulting from the composition (previously shown) refines the following contract* $c_S = (I_S, O_S, a_S, g_S)$ *encoding the safety property* φ_S*:*

- $I_S = \{\mathit{soc}_1\}$
- $O_S = \{\mathit{soc}_2, \mathit{soc}_3\}$
- $a_S \coloneqq 40 \leq \mathit{soc}_1 \leq 100$
- $g_S \coloneqq (\mathit{soc}_1 \geq 20) \wedge (\mathit{soc}_2 \geq 20) \wedge (\mathit{soc}_3 \geq 20)$

Relationship between PAMP-TLC and PAMP The Platform-Aware Mission Planning (PAMP) problem [Panjkovic *et al.*, 2025a] considers a similar setting, but the low-level layer is modeled with an explicit timed-transition system (TTS). Every snap action appearing in a plan represents a command that triggers at the scheduled time a special transition in the platform associated to that event. In the time between two high-level commands, the platform is assumed to be non-deterministic, and it can perform internal transitions. The PAMP problem consists in generating a solution plan π for a temporal planning problem, such that it is guaranteed to be safe and executable for any non-deterministic platform behavior compliant with π. The executability notion is formalized by stating that for every plan prefix, for every state r of the platform reachable by applying the plan commands in that prefix at the prescribed times, if the time in r equals the time of the next snap action in the plan, the corresponding transition can be taken in r. The safety notion is defined by stating that every platform state r reachable in a trace compliant with the plan must satisfy a certain safety property φ.

If there is a proper abstraction relationship between the contracts associated to the planning actions and the TTS modeling the platform, we can guarantee that any plan that is a solution for a PAMP-TLC problem, is also a solution for the PAMP problem sharing the same planning problem. Semi-formally, given a contract $c = (I, O, a, g)$ with time variables $T_c = \{\tau_1, \ldots, \tau_n\}$, associated to a sequence of snap actions $\varrho = \langle e_1, \ldots, e_n\rangle$, we say that c is an abstraction of the execution of ϱ on TTS $\mathcal{T}$ with bound L and safety property φ if, in the TTS $\mathcal{T}'$ obtained from $\mathcal{T}$ by setting the initial condition to the assumptions of c, the following three proof obligations hold for any initial state s_0 of $\mathcal{T}'$:

1. $\varrho_{s_0} = \langle (s_0(\tau_1), e_1), \ldots, (s_0(\tau_n), e_n)\rangle$ is executable on $\mathcal{T}'$;
2. in any run compliant with the execution of ϱ_{s_0} on $\mathcal{T}'$, the guarantees of c hold in the interval $[s_0(\tau_n), s_0(\tau_n) + L]$;
3. in any run compliant with the execution of ϱ_{s_0} on $\mathcal{T}'$, if the safety property φ holds in s_0 and at $s_0(\tau_n)$, then it holds up to $s_0(\tau_n) + L$.

Intuitively, the first condition can be used to prove that a solution for PAMP-TLC is executable on $\mathcal{T}$, as all the actions appearing in the plan will be executable considering that the assumptions of the respective contracts will hold at their start; the second condition guarantees that the safety property will

Algorithm 1 Abstraction-refinement algorithm with contract validation

procedure CONTRACTPLANNING(Π, $\mathcal{C}$, λ, φ_I, φ_S)
  bad_seqs = {}
  while True **do**
    $\pi_{\text{STN}} \leftarrow$ PLAN(Π, bad_seqs)
    pass, bad_seqs$_\pi$, $\pi \leftarrow$ CHECK($\mathcal{C}$, λ, π_{STN}, φ_I, φ_S)
    if pass **then return** π
    else bad_seqs $\leftarrow$ bad_seqs $\cup$ bad_seqs$_\pi$
procedure CHECK($\mathcal{C}$, λ, π_{STN}, φ_I, φ_S)
  $\sigma_1^\pi, \dots, \sigma_r^\pi \leftarrow$ maximal_subseqs(π_{STN})
  for $i = 1$ to r **do**
    $\Phi_i \leftarrow \lambda(\sigma_i^\pi)$
    $\Delta_i \leftarrow$ BUILDDURATIONSCONTRACT(π_{STN}, i)
    if $i == 1$ **then**
      $c_1 \leftarrow [\Phi_1 \bullet \Delta_1]_1^{k_1}$
    else
      $c_i \leftarrow c_{i-1} \parallel [\Phi_i \bullet \Delta_i]_i^{k_i}$
    $c_S^i \leftarrow$ BUILDSAFETYCONTRACT(φ_I, φ_S, i)
    $c_Q^i = (I_Q^i, O_Q^i, a_Q^i, g_Q^i) \leftarrow c_S^i / c_i$
    if g_Q^i is satisfiable **then**
      if $i == r$ **then**
        return true, $\emptyset$, GETPLAN(g_Q^i, π_{STN})
      else continue
    else
      $j' \leftarrow \sum_{j=1}^{i} |\sigma_j^\pi|$
      bad_seqs$_\pi \leftarrow (e_1, \dots, e_{j'}) \cup$ BADSUBSEQS(π_{STN}, i)
      return false, bad_seqs$_\pi$, $\emptyset$

hold on the platform at the ending times of the actions, and in the time between two actions (we are assuming that we are given an upper bound L on the maximum distance between two actions); the third condition guarantees that the safety property is preserved during the execution of each action.

The full formalization and proof of this result, as well as the presentation of the necessary formal background on the PAMP problem, is available in the additional material.

The first two proof obligations can be checked by applying directly the validation subroutine of the algorithm for solving the PAMP problem [Panjkovic *et al.*, 2025a], with a safety property ψ expressing the fact that when the global time is in $[s_0(\tau_n), s_0(\tau_n) + L]$, the guarantees of c hold: $\psi \coloneqq (s_0(\tau_n) \leq \gamma \leq s_0(\tau_n) + L) \rightarrow g|_{s_0}$, where γ is the *global clock* of $\mathcal{T}'$ (denoting the time since the start), and $g|_{s_0}$ is the formula obtained by substituting every $v \in V$ in g with $s_0(v)$, and every $v' \in V'$ in g with v. The third proof obligation can be checked with the following LTL property, specifying that if φ holds in the initial state, and eventually in the run the global time is $s_0(\tau_n)$ and φ holds, then in every state of the run with global time up to $s_0(\tau_n) + L$, φ holds: $\varphi \wedge F(\gamma = s_0(\tau_n) \wedge \varphi) \rightarrow G(\gamma \leq s_0(\tau_n) + L \rightarrow \varphi)$.

4 Solving PAMP-TLC

In this section, we present an approach for solving the PAMP-TLC problem. A temporal planner is used to solve the planning problem and enumerate candidate solution plans. In order to check that a plan satisfies the low-level safety property, we compute the top-level contract associated to the plan by composing the contracts of each action (and each allowed interleaving) in the plan, and then by computing the quotient between a contract encoding the safety property and the plan contract. This determines (if it exists) a scheduling of the actions that satisfies the safety property.

For temporal planning, we assume a sound planner that can return 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 $e_1, \dots, e_n$, where each snap action e_i is associated to a time variable t_i representing the moment in which it is applied, and a set of constraints between time variables enforcing the duration constraints and the ordering. We assume that the planner search can avoid plans containing a given set of actions sequences. In principle, any heuristic-search planner can be instrumented to satisfy these requirements by using plan de-ordering methods [Bäckström, 1998] and by checking the prefixes of plans for pruning.

The approach is detailed in Algorithm 1. First, the planning problem Π is solved, obtaining a set of solution plans π_{STN} characterized by a fixed sequence of snap actions $e_1, \dots, e_n$ and a set of constraints between the time variables $t_1, \dots, t_n$ associated to them. Then, the solution π_{STN}, and the formulas φ_I and φ_S encoding the initial state and safety property are passed to the CHECK procedure. In the CHECK procedure, we first determine the sequence $\sigma_1^\pi, \dots, \sigma_r^\pi$ of maximal overlapping subsequences of π_{STN}. We iterate over all prefixes $i \in \{1, \dots, r\}$, and we consider the contract $\Phi_i = \lambda(\sigma_i^\pi)$ that is associated to σ_i^π. We compute the contract Δ_i, whose assumption is the conjunction of all the constraints in π_{STN} (expressed as $\mathcal{QF\_LRA}$ formulas) containing time variables in $\{t_1, \dots, t_i\}$, and whose guarantee is $\top$. We merge these contracts and rename the resulting contract obtaining $[\Phi_i \bullet \Delta_i]_i^{k_i}$, which is such that the input and output variables have index i and $i+1$ respectively, and the time variables start with index $k_i = 1 + \sum_{j=1}^{i-1} |\sigma_j^\pi|$, i.e. the next index after the time variables of the previous contracts in the sequence. We then compose this contract with the contract considered in the previous step: $c_i = c_{i-1} \parallel [\Phi_i \bullet \Delta_i]_i^{k_i}$. We compute the contract c_S^i encoding the safety property for the first i steps: the assumption is $[\varphi_I]_1$ and the guarantee is $\bigwedge_{j=1}^{i+1} [\varphi_S]_j$. We then compute the quotient $c_Q^i = c_S^i / c_i$, whose output variables are the time variables for the subsequence of the plan $\sigma_1^\pi, \dots, \sigma_i^\pi$. By the properties of the quotient operation, the values for the time variables satisfying the guarantees of c_Q^i are such that $c_i \leq c_S^i$. If g_Q^i is satisfiable, we extract a value for the time variables, for which it is guaranteed that the plan π satisfies the safety property φ_S. Otherwise, the current prefix of snap actions is not schedulable in a way that guarantees the safety property, and therefore we return to the planner this invalid prefix, which can then be used for pruning in the continuation of the planning search. In addition to that, we analyze this prefix with the BADSUBSEQS procedure, to determine whether there is a shorter subsequence that is guaranteed to violate the safety property, which is a stronger information that the planner can learn. For an invalid prefix $\langle e_1, \dots, e_i \rangle$, we consider subsequences $\langle e_j, \dots, e_i \rangle$, where j goes from i to 1, and we determine whether the composition of the respective contracts can refine the safety contract, starting from any initial state. If that is not the case, we avoid in the planning search every plan containing such a subsequence.

Theorem 1. *Let* $\Upsilon = \langle \Pi, \mathcal{C}, \lambda, \varphi_i, \varphi_S \rangle$ *be a PAMP-*

Domain	PAMP			PAMP-TLC			
	coverage	avg loops	avg time	coverage	avg loops	avg time	avg time + cc
factory	1	2.0	11489.7	**10**	3.3	369.0	374.2
fix1	**15**	14.3	8141.9	**15**	5.7	8151.0	8155.0
fix2	**16**	13.2	7980.6	**16**	5.1	7978.1	7983.1
jpl1	5	1.4	6621.8	**8**	2.0	4327.6	4359.5
jpl2	3	1.0	7653.0	**8**	2.1	4579.9	4614.2

Table 1: Coverage table reporting the number of solved instances, average time (in seconds) and iterations *on the solved instances*. The column "avg time + cc" includes the proof-obligation checking time.

TLC problem, and let π be a plan returned by $\textsc{ContractPlanning}(\Pi, \mathcal{C}, \lambda, \varphi_I, \varphi_S)$. *Then, π is a valid solution for Υ.*

Proof. (Sketch) We show that π satisfies Definition 11.

1. Let π_{STN} be the last plan in STN form returned by $\textsc{Plan}(\Pi, \text{bad\_seqs})$. Because of the soundness of the planner, every plan that has the specified ordering of snap actions and a timing for each snap action that satisfies the STN constraints of π_{STN} is a valid solution for Π. The plan π is returned by $\textsc{GetPlan}(g_Q, \pi_{\text{STN}})$ when iteration $i = r$ of the $\textsc{Check}(\mathcal{C}, \lambda, \pi_{\text{STN}}, \varphi_I, \varphi_S)$ procedure is reached. Since $i = r$, the assumptions of c_i contain all the constraints in π_{STN}. Since $c_Q = c_S / c_i$, by the properties of the quotient operation the value for the timings extracted from the guarantee g_Q of c_Q will satisfy the assumptions of c_i, which include the STN constraints. Since the ordering of the snap actions is preserved, we can conclude by the soundness of the planner that π is a solution for Π.
2. Since π is obtained by extracting a value for the timings of the snap actions from g_Q, we have that $c_\lambda^\pi \leq c_Q \parallel c_r$. By the properties of the quotient operation, we have that $c_Q \parallel c_r \leq c_S$. Therefore, $c_\lambda^\pi \leq c_S$. □

The specific way in which the composition and quotient operations are implemented can have an impact on the performance of the algorithm (while the soundness is always preserved): looser implementations can lead to the invalidation of plans that could have passed the refinement check with the safety contract, if a tighter implementation was used.

5 Experimental evaluation

We developed the presented approach in a solver written in Python based on Pacti, a Python framework for carrying out compositional system analysis and design by reasoning on IO Contracts [Incer *et al.*, 2025]. For temporal planning we use the TAMER planner [Valentini *et al.*, 2020]. Finally, for checking the contracts w.r.t. a platform modeled as a TTS we use NUXMV [Cimatti *et al.*, 2019], a symbolic model checker that is able to prove temporal properties of TTSs.

We experimentally evaluated the new approach against the PAMP implementation in [Panjkovic *et al.*, 2025a] both on the JPL case-study modeled as IO contracts presented in [Rouquette *et al.*, 2023], where we added two versions of the planning problem to show our synthesis capabilities, and a simple platform model compliant with the contracts to compare against PAMP. Moreover, we added the contract layer to the benchmark set used in [Panjkovic *et al.*, 2025a].

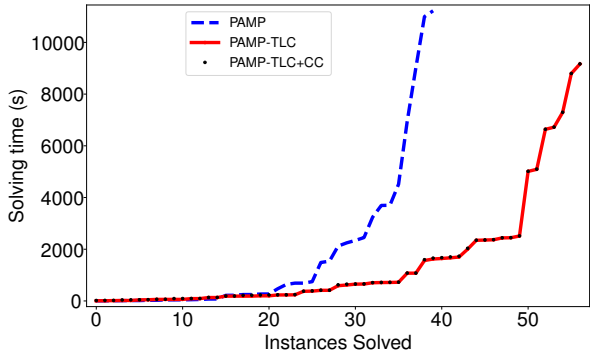

Figure 3: Cactus plot of the experiment. A point (x, y) of the line for a solver indicates that the solver took y seconds to solve its x-th harder instance. PAMP-TLC+CC (which has practically identical performance as PAMP-TLC) considers the cost of proof-obligation checking for every instance.

All the experiments were performed on a cluster of identical machines with AMD EPYC 7413 24-Core Processor and running Debian GNU/Linux 13. We used a timeout of 14400 seconds and a memory limit of 20GB.

Table 1 reports the coverage table comparing our approach PAMP-TLC with PAMP. It reports the number of solved instances for each domain, the average number of abstraction-refinement loops that were necessary in each domain before finding a valid solution plan, and the time required by PAMP-TLC with the additional check that each contract is a correct abstraction w.r.t. the platform model. The results show that PAMP-TLC generally solves more instances compared to PAMP, and with faster solving times, as highlighted by the cactus plot in Fig. 3. The number of required iterations for validating the plans produced by the planner is also lower for PAMP-TLC. Moreover, the cost of checking that a contract is a correct abstraction for the platform models is very low, between 5 and 30 seconds for the considered benchmarks.

These results highlight that by having a more abstract representation of the platform, using contracts to model the effect of applying each planning action on the controlled platform, we can generate robust plans more effectively, compared to reasoning on the platform model directly. With the additional checking that each contract is a correct abstraction of the execution of the corresponding action on the platform, the returned plans are also valid solutions for the PAMP problems, and the cost of these checks is shown to be very low.

6 Conclusions

In this paper, we presented a novel approach to tackle the Platform-Aware Mission Planning (PAMP) problem, by exploiting a Task-Level abstraction provided in the form of Input-Output contracts. We showed a planning schema that finds plans that are valid according to this abstraction, and we defined the proof obligations that need to be checked on a concrete platform for the plan to be valid according to the PAMP semantics. As future work, we plan to explore the problem of synthesis for the contracts from a prototypical platform implementation.

Acknowledgments

Andrea Micheli has been partially supported by the STEP-RL project funded by the European Research Council under GA n. 101115870. The initial collaboration for this paper was established during the Contract Languages Workshop at the Lorentz Center (March 2024).

References

[Bäckström, 1998] Christer Bäckström. Computational aspects of reordering plans. *J. Artif. Intell. Res.*, 9:99–137, 1998.

[Benveniste *et al.*, 2018] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. *Contracts for System Design*, volume 12. Now Publishers Inc., Hanover, MA, USA, 2018.

[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.*, 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.

[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.

[Incer *et al.*, 2025] Inigo Incer, Apurva Badithela, Josefine B. Graebener, Piergiuseppe Mallozzi, Ayush Pandey, Nicolas Rouquette, Sheng-Jung Yu, Albert Benveniste, Benoît Caillaud, Richard M. Murray, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Pacti: Assume-guarantee contracts for efficient compositional analysis and design. *ACM Trans. Cyber Phys. Syst.*, 9(1):3:1–3:35, 2025.

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

[Mallozzi *et al.*, 2023] Piergiuseppe Mallozzi, Inigo Incer, Pierluigi Nuzzo, and Alberto L. Sangiovanni-Vincentelli. Contract-based specification refinement and repair for mission planning. In *11th IEEE/ACM International Conference on Formal Methods in Software Engineering, FormaliSE 2023, Melbourne, Australia, May 14-15, 2023*, pages 29–38. IEEE, 2023.

[Panjkovic *et al.*, 2025a] Stefan Panjkovic, Alessandro Cimatti, Andrea Micheli, and Stefano Tonetta. Generalizing Platform-Aware Mission Planning for Infinite-State Timed Transition Systems. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 843–852, 10 2025.

[Panjkovic *et al.*, 2025b] Stefan Panjkovic, Alessandro Cimatti, Andrea Micheli, and Stefano Tonetta. Platform-Aware Mission Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 35(1):93–101, Sep. 2025.

[Patra *et al.*, 2021] Sunandita Patra, James Mason, Malik Ghallab, Dana Nau, and Paolo Traverso. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence*, 299:103523, 2021.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE, 1977.

[Rouquette *et al.*, 2023] Nicolas Rouquette, Inigo Incer, and Alessandro Pinto. Early design exploration of space system scenarios using assume-guarantee contracts. In *2023 IEEE 9th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 15–24, 2023.

[Rovida *et al.*, 2017] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S. Polydoros, Bjarne Großmann, Ronald P. A. Petrick, and Volker Krüger. *SkiROS—A Skill-Based Robot Control Platform on Top of ROS*, pages 121–160. Springer International Publishing, Cham, 2017.

[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.