Transforming Robotic Plans with Timed Automata to Solve Temporal Platform Constraints

Tarik Viehmann, **Till Hofmann** and **Gerhard Lakemeyer**
Knowledge-Based Systems Group, RWTH Aachen University, Germany
{viehmann, hofmann, lakemeyer}@kbsg.rwth-aachen.de

Abstract

Task planning for mobile robots typically uses an abstract planning domain that ignores the low-level details of the specific robot platform. Therefore, executing a plan on an actual robot often requires additional steps to deal with the specifics of the robot platform. Such a platform can be modeled with timed automata and a set of temporal constraints that need to be satisfied during execution.

In this paper, we describe how to transform an abstract plan into a platform-specific action sequence that satisfies all platform constraints. The transformation procedure first transforms the plan into a timed automaton, which is then combined with the platform automata while removing all transitions that violate any constraint. We then apply reachability analysis on the resulting automaton. From any solution trace one can obtain the abstract plan extended by additional platform actions such that all platform constraints are satisfied. We describe the transformation procedure in detail and provide an evaluation in two real-world robotics scenarios.

1 Introduction

When using planners to solve robotic tasks, a programmer has to model the domain of interest. However, it is often problematic to specify the behavior of a robot based on high-level domain features alone, as lower-level components may impose constraints on the executability of high-level actions. As an example, we consider a simple robot that can move around, and pick up as well as put down objects. For high-level reasoning purposes, it may suffice to only consider the actions `put`, `pick`, and `goto`. When actually operating on the robot, additional details need to be taken care of, e.g., the perception unit needs to be enabled when picking an object. As the camera needs some time to initialize, it needs to be enabled a few seconds before it is used. When moving, the robot should disable its perception unit to save computational resources. In this scenario, the perception control cannot be handled within low-level implementations easily without either wasting time at each pick (turning it on when the robot decides to pick, then wait until it is ready) or wasting resources (turning it on when it is unnecessary). The essential problem here is that the perception handling is dependent on the context determined by the high-level reasoner.

These low-level constraints could be considered directly by the high-level planner. However, this would considerably increase the domain size and thus impair planner performance. Also, it goes against separation of concerns, as the high-level planning domain directly incorporates low-level platform details. Thus, a separation of high-level reasoning and robot-specific actions is desirable [Hofmann *et al.*, 2018].

In this paper, we instead propose a plan transformation based on timed automaton (TA) reachability analysis. We model each robot platform component as a TA and define constraints on the platform model using a subset of Metric Temporal Logic (MTL). The MTL constraints connect the high-level plan with the robot platform, e.g., by requiring that the camera is turned on two seconds before the robot performs the action `pick`. Given a high-level plan, we first construct a TA that represents the plan. Next, we extend the constructed TA by replacing each state by a copy of the platform model, while removing any edges that violate a constraint. By applying reachability analysis on the constructed TA, we obtain a sequence of high-level and low-level actions along with execution times that satisfy all constraints.

In the remainder of this paper, we first summarize related work, before we address some core design decisions of our proposed transformation. Then, we summarize the formalism of MTL and TAs, before we describe how to encode the problem as a reachability task on TAs. We evaluate our approach on logistics robots as well as on a domestic service robot, before we conclude.

2 Related Work

Task and motion planning (TAMP) combines high-level task planning with low-level motion planning, e.g., with an interface between task and motion planner to effectively combine off-the-shelf planners [Srivastava *et al.*, 2014], by extending the FF heuristics for geometric reasoning [Garrett *et al.*, 2015], or by using constraint programming to guide the high-level search with geometric [Gravot *et al.*, 2005; Dantam *et al.*, 2016] and temporal constraints [Erdem *et al.*, 2011]. In contrast to our proposed approach, TAMP never extends but instead constrains the high-level plan. Additionally, the lower level in our approach differs from TAMP; while

TAMP focuses on motion planning, our low-level components are symbolic representations of arbitrary platform components not restricted to manipulation. In a similar fashion, Erdem *et al.* [2016] integrate general feasibility checks into an ASP-based planner, either by checking constraints directly during search, or by constraining the planner afterwards if a feasibility check is violated. Alternatively, external predicates can be directly embedded into PDDL-based planners [Hertle *et al.*, 2012; Dornhege *et al.*, 2012], ASP-based planners [Erdem *et al.*, 2012], and other reasoners such as CCALC [Aker *et al.*, 2011]. There, a low-level component sets the value of a symbolic atom used by the task planner, which allows to integrate platform components, but does not allow for temporal constraints. RMPL [Kim *et al.*, 2001] extends simple temporal networks into temporal plan networks to use a temporal planner to resolve temporal constraints. In contrast to the proposed approach, constraints in RMPL express relations between actions of the plan and do not provide an abstraction of the underlying platform. Konecnỳ *et al.* [2014] use temporal constraints for execution monitoring, but do not use the constraints to augment the original plan. Hofmann and Lakemeyer [2018] formulate metric temporal constraints on GOLOG programs and describe a transformation procedure based on MTL synthesis [Hofmann and Lakemeyer, 2020], but do not provide an implementation.

Concerning *planning via model checking* [Cimatti *et al.*, 1997; Giunchiglia and Traverso, 2000], Li *et al.* [2012] compare the performance of different model checking tools against planning tools, showing that some model checkers can compete with planners. Panek *et al.* [2006a] obtain an order schedule for a lacquer production plant from reachability analysis on TAs extended by weights and communication across different TAs with the tool `TaOpt` [Panek *et al.*, 2006b]. Largouët *et al.* [2016] combine two TA extensions to perform planning in an extension of the transport domain from the International Planning Competition (IPC). They also apply reachability analysis to compute plans. Ziegert and Wehrheim [2013] decouple time-critical software component management from a top-level graph transformation framework to guarantee exclusive access to resources.

3 Plan Transformation: Assumptions

Our goal is to decouple the platform specifics from the abstract reasoning. To narrow down this broad objective, we postulate core assumptions that form the base of the procedure this paper presents. Constraints required by the operating platform should be modeled separately from the abstract reasoning domain. Those constraints are satisfied only after a high-level course of action is determined. This is done by means of a plan transformation that adds platform actions to the existing abstract plan. It specifically does not remove or re-order existing plan actions to avoid tampering with the feasibility of the initial plan. For the same reason, controlling platform specifics must not alter the high-level reasoning states, hence the platform domain and planning domain are disjoint. This enables us to treat the operation of platforms as a process that runs in parallel to the execution of domain actions. It also allows us to treat the plan actions as atomic propositions in MTL. In particular, we do not need to know about the preconditions and effects of the plan actions, because plan actions do not affect platform actions (and vice versa). Instead, the dependencies that the transformation ensures may be fully captured in terms of temporal relations between the domain actions and the platform states. The responsibility of the proposed transformation is therefore to determine an executable plan together with execution start times for each action, based on the temporal restrictions propagated by the reasoner, the control restrictions induced by the models, and the temporal constraints expressing relations between high-level and low-level actions. For simplicity, we assume that the robot platform is modeled with a single TA. If multiple TAs are modeled, they can be combined by a product construction [Alur and Dill, 1994].

4 Foundations

We briefly summarize suitable formalisms, namely MTL and timed automata, to model temporal dependencies.

4.1 Metric Temporal Logic (MTL)

MTL [Koymans, 1990] extends Linear Time Logic (LTL) with timing constraints on the *Until* modality. One commonly used semantics for MTL is a *pointwise semantics* [Henzinger, 1998], in which formulas are interpreted over timed words.

Definition 1 (Timed Words)**.** *A timed word ρ over a finite set of atomic propositions Σ is a finite or infinite sequence $(\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$, where $\sigma_i \subseteq \Sigma$ and $\tau_i \in \mathbb{R}_{\geq 0}$ such that the sequence (τ_i) is monotonically non-decreasing and non-Zeno. The set of timed words over Σ is denoted as $T\Sigma^*$.*

Definition 2 (Formulas of MTL)**.** *Given a set Σ of atomic propositions, the formulas of MTL are built as follows:*

$$\phi ::= \sigma \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathbf{U}_I\, \phi$$

We use the abbreviations $\mathbf{X}_I\phi := (\bot\, \mathbf{U}_I\, \phi)$ (*next*), $\mathbf{F}_I\phi := (\top\, \mathbf{U}_I\, \phi)$ (*future*) and $\mathbf{G}_I\phi := \neg\mathbf{F}_I\neg\phi$ (*globally*). We may omit the interval I if $I = [0, \infty)$.

Definition 3 (Pointwise semantics of MTL)**.** *Given a timed word $\rho = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$ over alphabet Σ and an MTL formula ϕ, $\rho, i \models \phi$ is defined as follows:*

1. *$\rho, i \models p$ iff $p \in \sigma_i$*
2. *$\rho, i \models \neg\phi$ iff $\rho, i \not\models \phi$*
3. *$\rho, i \models \phi_1 \wedge \phi_2$ iff $\rho, i \models \phi_1$ and $\rho, i \models \phi_2$*
4. *$\rho, i \models \phi_1\, \mathbf{U}_I\, \phi_2$ iff there exists j such that (1) $i < j < |\rho|$, (2) $\rho, j \models \phi_2$, (3) $\tau_j - \tau_i \in I$, (4) and $\rho, k \models \phi_1$ for all k with $i < k < j$.*

Following [Henzinger, 1998], we use the strict semantics for the *Until* modality, which can be translated into the non-strict semantics [Ouaknine and Worrell, 2005].

4.2 Timed Automata

Timed automata [Alur and Dill, 1994] extend finite automata by a notion of continuous time. We adopt the definition of Bengtsson and Yi [2004].

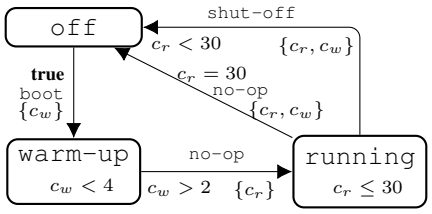

Figure 1: TA $\mathcal{A}_{\texttt{cam}}$ to model a simple perception unit.

Definition 4. *A* timed automaton (TA) *is a tuple* $\mathcal{A} = (L, l_0, E, I)$ *over a finite set of clocks* $\mathcal{C}$ *and a finite alphabet* Σ_E*, where* L *is a finite set of locations (also called states);* $l_0 \in L$ *is a starting location;* $I : L \to \Phi(\mathcal{C})$ *is the invariant of each location; and* $E \subseteq L \times \Phi(\mathcal{C}) \times \Sigma_E \times 2^{\mathcal{C}} \times L$ *is the set of transitions.* $\Phi(\mathcal{C})$ *denotes the set of* clock constraints δ *(also called guards), defined by* $\delta ::= x \mathrel{op} c \mid c \mathrel{op} x \mid \delta \wedge \delta$*, where* $x \in \mathcal{C}$ *is a clock,* $op \in \{<, \leq, =, \geq, >\}$ *and* $c \in \mathbb{Q}$ *is a constant. Instead of* $(l_i, g, a, r, l_j) \in E$*, we write* $l_i \xrightarrow{g,a,r} l_j \in E$ *and call* r *the set of* resets *or* updates*.*

TAs model behavior depending on continuous time with the help of clocks. The current value of clocks is expressed via *Clock assignments* $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$, initially all clocks start at value 0, then they advance continuously and at the same rate. At any given time, a TA $\mathcal{A}$ is in a configuration $\langle l_i, \nu_i \rangle$, which is valid only if the clock values given through ν_i satisfy the constraint $I(l_i)$. A configuration can be changed by either letting d time elapse while not changing l_i, or by taking a transition $e = l_i \xrightarrow{g,a,r} l_j \in E$. Transition e may only be taken if ν_i satisfies g and taking e changes all values of clocks in r to 0. This induces an infinite transition system $\mathfrak{T}(\mathcal{A})$ where the set of states are all possible configurations and the transitions are formed by either delaying for d seconds or taking transitions $e \in E$. A path in $\mathfrak{T}(\mathcal{A})$ is called a *timed trace* and defines a *run* on $\mathcal{A}$ through the labels on the used transitions. Each run induces a timed word $(\sigma_0, \tau_0)(\sigma_1, \tau_1) \ldots \in T\Sigma_E^*$, establishing a connection between MTL and TAs.

The *reachability problem on a TA* can be stated as follows: Given a starting configuration $\langle l, \nu \rangle$ and a destination state l' along with a target clock constraint φ, find a path from $\langle l, \nu \rangle$ to a configuration $\langle l', \nu' \rangle$ where ν' satisfies φ. The problem is PSPACE-complete and thus decidable [Alur and Dill, 1994].

5 Platform Constraints

As motivated above, we are given a plan $P = \langle a_1, \ldots, a_n \rangle$ and a TA $\mathcal{A}_{\mathcal{M}}$ with labeled states, as shown in Figure 1. The plan P describes some agent behavior, while the TA $\mathcal{A}_{\mathcal{M}}$ models the specific robot platform. The edge labels of the platform components describe low-level actions that the robot can take, e.g., `shut-off`. The set of plan actions Σ_P, the low-level actions Σ_E and the state labels Σ_L are assumed to be pairwise disjoint. For a timed word σ we consider it to describe a transformed plan, if for each $\sigma_i \in \sigma$: $|\sigma_i \cap \Sigma_P| \leq 1$, $|\sigma_i \cap \Sigma_E| \leq |\sigma_i \cap \Sigma_L| = 1$ and $\delta_0 \cap \Sigma_P = \emptyset$. Hence, in σ, the platform is always in exactly one state, at most one action is executed, and the first entry contains no action. We connect the plan P and the platform TA $\mathcal{A}_{\mathcal{M}}$ by three kinds of temporal constraints formulated in MTL over $\Sigma = \Sigma_P \cup \Sigma_E \cup \Sigma_L$.

5.1 Requiring Certain Platform Actions

Let us start with constraints postulating how $\mathcal{M}$ has to be controlled during P. Let β_i denote a disjunction over a subset of $\Sigma_P \subseteq \Sigma$ and α_i a disjunction over a subset of $\Sigma_L \subseteq \Sigma$.

$$\mathfrak{uc}(B, \alpha_1, \alpha_2) := \mathbf{G}\big[(\alpha_1 \wedge \mathbf{X}(\alpha_2 \vee (\neg\alpha_1 \mathbf{U} \alpha_2))) \supset (\beta_1 \wedge \neg\alpha_2)\mathbf{U}_{I_1}(\beta_2 \wedge \neg\alpha_2)\mathbf{U}_{I_2} \ldots (\beta_n \wedge \neg\alpha_2)\mathbf{U}_{I_n}\alpha_2\big]$$

where $B := \langle\langle \beta_1, I_1 \rangle, \ldots, \langle \beta_n, I_n \rangle\rangle$. A so-called *until-chain* constraint $\mathfrak{uc}$ states that whenever an action a_1 matching α_1 is followed by a_2 matching α_2, without further occurrences of α_1 or α_2 in between, then between such a matching pair of domain actions a_1 and a_2 the platform has to sequentially remain in the states specified through the β_i in B for a time span given through the intervals I_i.

Satisfying $\mathfrak{uc}(B, \alpha_1, \alpha_2)$, given a plan $P = \langle a_1, \ldots, a_n \rangle$, equates to grounding a timed word $(\{a_1\}, \tau_1) \ldots (\{a_n\}, \tau_n)$ while also extending it by additional platform actions. Consider the TA from Figure 1, $\gamma_e := \mathfrak{uc}(\langle \texttt{running}, [0, \infty) \rangle, \texttt{s\_pick}, \texttt{e\_pick})$ and a plan $\langle \texttt{s\_pick}, \texttt{e\_pick} \rangle$ consisting of a durative pick action represented by a start and end action. Let $\rho := (\{\texttt{s\_pick}\}, \tau_1)(\{\texttt{e\_pick}\}, \tau_2)$ for some $\tau_1, \tau_2 \in \mathbb{Q}_+$. It holds that $\rho \models (\texttt{s\_pick} \wedge \mathbf{X}(\texttt{e\_pick} \vee (\neg\texttt{s\_pick}\mathbf{U}\texttt{e\_pick})))$. Hence, the constraint γ_e requires $(\texttt{running} \wedge \neg\texttt{e\_pick})\mathbf{U}\texttt{e\_pick}$. Assuming $\mathcal{A}_{\texttt{cam}}$ is in state `off` prior to P, we could adapt ρ to satisfy γ_e:

$$\rho' := (\{\texttt{off}\}, 0)(\{\texttt{boot}, \texttt{warm-up}\}, 0)(\{\texttt{running}\}, 3)(\{\texttt{running}, \texttt{s\_pick}\}, 3)(\{\texttt{e\_pick}\}, 3)$$

5.2 Temporal Constraints on Plan Actions

Furthermore, we may also impose temporal constraints on the actions in the abstract plan, e.g., the duration between two plan actions. For this, we define two macros *IsDomAct* and *PlanOrder*(i), where *IsDomAct* $:= \bigvee_{a \in P} a$ holds iff the current action is a plan action, and *PlanOrder*(i) holds iff the current action is the i-th action in the plan. Given the total number of plan actions n, it can be defined as follows:

$$\textit{PlanOrder}(i) := \begin{cases} \textit{IsDomAct} \wedge \neg\mathbf{F}\,\textit{IsDomAct} & \text{if } i = n \\ \textit{IsDomAct} \wedge \mathbf{F}\,\textit{PlanOrder}(i+1) & \\ \quad \wedge\neg\mathbf{F}\,\textit{PlanOrder}(i) & \text{if } i < n \\ \bot & \text{else} \end{cases}$$

Formulas to constrain the time interval I in which the i-th plan action starts can be simply given as follows:

$$\mathfrak{abs}(i, I) := \mathbf{F}_I\,\textit{PlanOrder}(i)$$

We may also require a certain amount of time passing between plan actions by specifying the time interval I in between the i-th and j-th plan action, with $j > i$:

$$\mathfrak{rel}(i, j, I) := \mathbf{F}\big[\textit{PlanOrder}(i) \wedge \mathbf{F}_I\,\textit{PlanOrder}(j)\big]$$

Continuing with the example from above, we can adapt ρ' to satisfy a constraint $\mathfrak{rel}(1, 2, [15, 20])$ (requiring a time interval $[15, 20]$ between action 1 and action 2) by adapting the grounded times in ρ':

$$\rho'' := (\{\texttt{off}\}, 0)(\{\texttt{boot}, \texttt{warm-up}\}, 0)(\{\texttt{running}\}, 3)(\{\texttt{running}, \texttt{s\_pick}\}, 3)(\{\texttt{e\_pick}\}, 18)$$

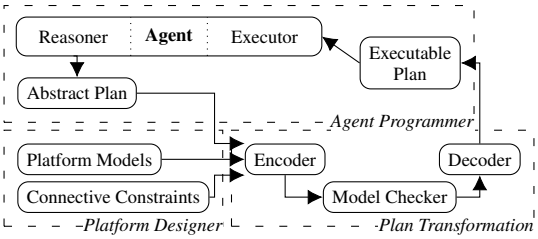

Figure 2: Steps to transform abstract plans into executable ones based on low-level platform specifics.

Listing 1: Transformation Procedure

```
1  A_P := encode_plan(P, C_abs ∪ C_rel)
2  A_base := add_platform_ta(A_P, A_M)
3  A_enc := encode_platform_control(A_base, C_uc):
4    For γ ∈ C_uc:
5      For (a_j, a_j') in get_activations(γ, P)
6        A_base := enforce_uc(A_base, a_j, a_j', γ)
7    return A_base
8  enforce_uc(A_curr, a_j, a_j', uc(⟨β_1, I_1⟩ ... ⟨β_n, I_n, α_1, α_2⟩)):
9    A_context := get_sub_ta(A_curr, a_j, a_j')
10   For i in {1, ..., n}:
11     S_i := copy(A_context)
12     S_i := restrict(A_curr, α_i, I_i)
13   A_temp := combine(S_1, ..., S_n, I_1, ..., I_{n-1})
14   return replace(A_context, A_temp, S_1, S_n, I_n)
```

6 Transformation

Our proposed plan transformation can be summarized as follows: (1) Construct a single TA encoding the satisfaction of constraints during the plan as a reachability problem to a designated goal state (2) Find a symbolic representation of traces reaching that state using a model checker (3) Decode an actual trace from a symbolic one with grounded execution times. An overview of the procedure is shown in Figure 2. Here we focus on the encoding step, which is the core of our approach. The goal of the encoding step is to construct one TA $\mathcal{A}_{\texttt{enc}}$ with a state `fin`, such that every run reaching `fin` corresponds to a timed word satisfying the given constraints (of the form $\mathfrak{uc}, \mathfrak{abs}$ and $\mathfrak{rel}$) whithin the plan $P = \langle a_1, \ldots, a_k \rangle$. Listing 1 gives an overview over the necessary steps `encode_plan`, `add_platform_ta` and `encode_platform_control` to construct $\mathcal{A}_{\texttt{enc}}$.

Consider $P^e = \langle \texttt{s\_goto}, \texttt{e\_goto}, \texttt{s\_pick}, \texttt{e\_pick} \rangle$, a plan consisting of start and end actions for the durative actions `goto` and `pick`, together with constraints:

$$\gamma_1 := \mathfrak{rel}(1, 2, [30, 45)) \quad \gamma_2 := \mathfrak{rel}(2, 3, [0, 0]))$$
$$\gamma_3 := \mathfrak{rel}(3, 4, [15, 20])$$
$$\gamma_4 := \mathfrak{uc}(\langle\langle \texttt{off}, [0, \infty] \rangle, \langle \top, [0, 4] \rangle\rangle, \texttt{s\_goto}, \texttt{e\_goto})$$
$$\gamma_5 := \mathfrak{uc}(\langle\langle \texttt{running}, [0, \infty] \rangle\rangle, \texttt{s\_pick}, \texttt{e\_pick})$$

The constraints γ_1, γ_2 and γ_3 provide the duration of `goto` and `pick` and enforce that `pick` follows without any pause after `goto` finishes; γ_4 states that during a `goto` action the camera should be turned off until the last 4 seconds of its execution, where it may become necessary to boot the camera for a subsequent `pick` action, which per γ_5 requires a running camera for the whole duration. We denote the sets of constraints of the respective types by $C_{\mathfrak{uc}}, C_{\mathfrak{abs}}$ and $C_{\mathfrak{rel}}$.

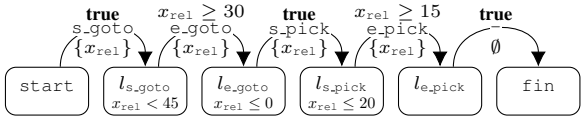

Figure 3: Encoding plan P^e and $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}}$.

6.1 Plan Constraint Encoding

The procedure `encode_plan` constructs a TA $\mathcal{A}_P$ with designated states $a_0 := \texttt{start}$ and $a_{n+1} := \texttt{fin}$ such that traces from `start` to `fin` induce execution time groundings of P respecting $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}}$. The construction is depicted in Figure 3 for our running example. Initially, the plan actions are translated into states as follows:

$$\mathcal{A}_P := (L_P, \texttt{start}, E_P, I_P, X_P)$$
$$L_P := \{l_a \mid a \in P\} \cup \{\texttt{start}, \texttt{fin}\}$$
$$E_P := \{l_{a_i} \xrightarrow{\top, a_j, \emptyset} l_{a_{i+1}} \mid 0 \leq i \leq k\}$$
$$I_P(l) := \top \text{ for all } l \in L_P$$

A fresh clock $x_{\texttt{abs}}$ is introduced to encode $C_{\mathfrak{abs}}$ as follows: Let $lb(x, I) := x \geq a$, if $I = [a, b)$ or $I = [a, b]$ and $lb(x, I) := x > a$ otherwise. Let $ub(x, I)$ be defined analogously for upper bounds. For each $\mathfrak{abs}(k, I_k) \in C_{\mathfrak{abs}}$, we add $ub(x_{\texttt{abs}}, I_k)$ as invariant to l_{k-1}. Similarly, the lower bound of I_k is added as guard $lb(x_{\texttt{abs}}, I_k)$ on the transition reaching l_k. Trivial bounds $x_{\texttt{abs}} \geq 0$ and $x_{\texttt{abs}} < \infty$ are omitted.

Lastly, for each $\mathfrak{rel}(k, k', I) \in C_{\mathfrak{rel}}$ a new clock $x_{k,k'}$ is added that is reset on the incoming transition of l_{a_k}. Then, a clock constraint encoding the lower bound of I is added to the guard on the incoming transition of $l_{a_{k'}}$ and the upper bound is added as invariant on all states l_i, $k \leq i < k'$. This step may be optimized by re-using a clock for different constraints when possible, e.g., a single clock $x_{\texttt{rel}}$ can encode all constraints of the form $\mathfrak{rel}(k, k+1, I)$.

The induced directed graph of the resulting TA $\mathcal{A}_P$ is a simple line and time elapsing in state l_{a_k} corresponds to the time elapsing since starting a_k (and before starting a_{k+1}). The constructed TA indeed encodes the possible action groundings to the plan that satisfy $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}}$:

Theorem 1. *$\rho = (\emptyset, 0)(\{a_1\}, \tau_1)(\{a_2\}, \tau_2), \ldots, (\{a_n\}, \tau_n)$ is a timed word satisfying $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}}$ iff there exists a valid timed trace ξ on $\mathcal{A}_P$ from `start` to `fin` that induces ρ.*

6.2 Platform Encoding

We proceed by incorporating the platform TA into the construction. Listing 1 Line 2 creates a TA $\mathcal{A}_{\texttt{base}}$ based on $\mathcal{A}_P$ such that runs to `fin` in $\mathcal{A}_{\texttt{base}}$ also include arbitrary control according to $\mathcal{A}_{\mathcal{M}}$. $\mathcal{A}_{\texttt{base}}$ consists of $k+1$ fresh copies $\mathcal{A}_{a_k}$ of $\mathcal{A}_{\mathcal{M}}$ (with $a_0 := \texttt{start}$). For each state $l \in L_P \setminus \texttt{fin}$, we add its invariant $I(l)$ to each state in the corresponding copy $\mathcal{A}_l$. For each transition $e = l \xrightarrow{g,a,r} l' \in E_P$, we add transitions from each state s in $\mathcal{A}_l$ to the corresponding copy s' in $\mathcal{A}_{l'}$ (or to `fin`, if $l' = \texttt{fin}$): $s \xrightarrow{g,a,r} s'$. Figure 4 shows such a constructed TA $\mathcal{A}_{\texttt{base}}$ schematically for our P^e and $\mathcal{A}_{\texttt{cam}}$, leaving out all annotations on states and transitions.

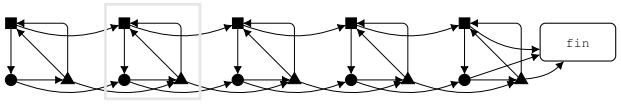

Figure 4: Combining Platform and plan. The activation scope of γ_4 is marked. ■: `off` ▲: `warm-up` ●: `running`

Theorem 2. *Every valid timed trace ξ on $\mathcal{A}_{base}$ from its initial state to `fin` induces a timed word ρ that satisfies $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}}$ and the sequence of platform actions within ρ induce a valid run in $\mathcal{A}_M$.*

6.3 Platform Constraint Encoding

Finally, we encode the until-chain constraints $C_{\mathfrak{uc}}$ (Listing 1 Line 3). By iteratively considering for each $\gamma \in C_{\mathfrak{uc}}$ all corresponding scopes $a_j, a_{j'}$ within the plan, the procedure encodes the platform controlling constraints. Hence the routine `enforce_uc` restricts the traces reaching `fin` between the start of a_j an $a_{j'}$ to remain in states of $\mathcal{A}_{\mathcal{M}}$ as specified in γ.

Consider $\gamma_4 = \mathfrak{uc}(B_4, \texttt{s\_goto}, \texttt{e\_goto})$ with $B_4 := \langle\langle \texttt{off}, [0,\infty]\rangle, \langle \top, [0,4]\rangle\rangle$. It states that whenever `s_goto` is followed by `e_goto` without occurrences of `s_goto` or `e_goto` in between, then starting at this `s_goto` action, the platform has to be `off` for some time, followed by arbitrary actions for at most four seconds until the matching `e_goto` starts. In other words, while driving, the camera has to be off unless the destination is reached within the next four seconds.

We need to modify each copy $\mathcal{A}_{a_i}$ in $\mathcal{A}_{\texttt{base}}$ that covers the relevant time frame to satisfy the constraint γ_4. Each $\mathcal{A}_{a_i}$ models platform actions in the time frame between two subsequent plan actions. In Line 9, we construct a sub-automaton $\mathcal{A}_{\texttt{context}}$ that contains all states that might be visited between a_j and a'_j, which is shown in Figure 4. An until-chain γ_i defines n sets of states specified by β_i that have to be subsequently visited, which we can enforce by creating n copies S_i of $\mathcal{A}_{\texttt{context}}$ (Line 11). Each S_i is subsequently restricted further (Line 12) to reflect the specification of β_i: First, all states not matching β_i are deleted. Then all those states within S_i are removed that can never be visited while satisfying β_i due to time conflicts imposed by the other constraints.

Since γ_4 describes platform control only between the first and second plan action in P^e, we end up with two copies S_1, S_2 of the initial copy $\mathcal{A}_{\texttt{s\_goto}}$. After restricting the possible states, we consider the temporal constraints of γ, which are given by the intervals I_i. With a fresh clock x_γ and by adding state invariants $ub(x_\gamma, I_i)$ to each state of S_i, the upper bound of I_i can be enforced if x_γ is set to 0 when entering S_i. This is done in the next subroutine `combine` (Line 13), where a single TA is formed out of $S_1, \ldots, S_n$ by connecting states $s \in L_{S_i}$ with $s' \in L_{S_{i+1}}$ as follows: Either s and s' are copies of the same state in $\mathcal{A}_{\texttt{context}}$, then a transition $s \xrightarrow{g_i, \text{copy}, \{x_\gamma\}} s'$ is added, where $g_i = lb(x_\gamma, I_{i+1})$ (*copy transitions*), or there exists a transition $s_{\text{c}} \xrightarrow{g,a,r} s'_{\text{c}}$ in $\mathcal{A}_{\mathcal{M}}$ and s is a copy of s_{c}, s' is a copy of s'_{c}, then $s \xrightarrow{g \wedge g_i, a, r \cup \{x_\gamma\}} s'$ is added (*successor transitions*).

Lastly, the resulting TA $\mathcal{A}_{\texttt{temp}}$ replaces the sub-automaton $\mathcal{A}_{\texttt{context}}$ within $\mathcal{A}_{\texttt{curr}}$ (Line 14). This can be achieved by iterating over all transitions $s \xrightarrow{g,a,r} s'$ of $\mathcal{A}_{\texttt{curr}}$ with either s in $\mathcal{A}_{\texttt{context}}$ and s' not in $\mathcal{A}_{\texttt{context}}$ (incoming transitions) or vice versa (outgoing transitions). Any incoming transition with s' matching β_1 gets the respective copy within S_1 as new destination and x_γ is added to r. Any outgoing transition with s matching β_n gets the respective copy within S_n as new source and $lb(x_\gamma, I_n)$ is added to g. All other incoming and outgoing transitions are deleted.

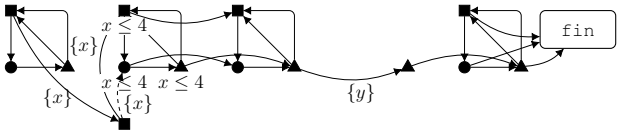

Figure 5: The full encoding of the ongoing example, only annotations related to the encoding of $C_{\mathfrak{uc}}$ are shown, trivial constraints such as $x \geq 0$ and $x < \infty$ are omitted.

Applying this construction to our example yields a TA as shown in Figure 5. When all activations of each $\gamma \in C_{\mathfrak{uc}}$ are encoded, our encoding stops with a TA $\mathcal{A}_{\texttt{enc}}$. The correctness of our encoding is covered by Theorem 3.

Theorem 3. *There exists a timed word ρ based on P augmented by control of $\mathcal{A}_{\mathcal{M}}$ that satisfies $C_{\mathfrak{abs}} \cup C_{\mathfrak{rel}} \cup C_{\mathfrak{uc}}$ iff there exists a run on $\mathcal{A}_{enc}$ from `start` to `fin`.*

7 Evaluation

Using our implementation `taptenc`[1] (**t**imed **a**utomata-based **p**lan **t**ransformation **enc**oding procedure), together with the CLI `verifyta` of the `UPPAAL` model checking suite [Bengtsson *et al.*, 1996], we conducted two experiments on an Intel i7-8565U 1.8 GHz processor with 16 GB memory. `UPPAAL` does not return concrete timed traces when answering reachability queries, but yields a symbolic trace which represents possibly infinitely many concrete traces. Considering difference bound matrices as symbolic representation formalism (such as used by `UPPAAL`), Bøgsted Poulsen and van Vliet [2010] describe how to retrieve the fastest concrete trace out of a given symbolic one. We adopt that approach in our tool to cover the full transformation procedure.

First, we consider a scenario from the *RoboCup Logistics League* (RCLL) [Niemueller *et al.*, 2013]. In the RCLL, robots have to move workpieces between machines and instruct those machines to assemble a product. We apply the transformation to possible production sequences of varying length in a simplified RCLL setting. The domain consists of the actions `goto(m,m')`, `pick(o,m)`, `get-from-shelf(o,m)`, `put(o,m)`, and `pay(o,m)`. The robot platform is modeled via three different timed automata. $\mathcal{A}_{\texttt{perc}}$ offers control over the perception to activate a camera when necessary, trigger object detection to validate grasping attempts as well as providing images to human supervisors during those attempts. $\mathcal{A}_{\texttt{calib}}$ manages the occasional need to calibrate an axis-driven gripper. Depending on the situation, a calibration may be undesirable, e.g., to avoid axis movement when holding something. Lastly, $\mathcal{A}_{\texttt{comm}}(m)$ allows to send instructions to a machine m to modify a product.

[1] https://github.com/TarikViehmann/taptenc

Platform TA	Time (s) trans	load_ta	reach	tracer	# states
perc	0.32	0.11	0.08	0.03	655
calib	0.07	0.04	0.03	0.01	271
comm	0.02	0.01	0.01	0.01	69
perc + calib	0.63	0.85	0.58	0.14	2660
+ 1x comm	1.2	2.4	1.6	0.26	4566
+ 2x comm	2.0	4.0	2.5	0.38	5645
+ 3x comm	4.2	8.7	4.9	0.63	8600
+ 4x comm	13.5	18.1	9.0	1.1	13883

Table 1: Average execution times of five runs on plans of length 50. trans: building the encoding and decoding, load_ta: required preprocessing step of verifyta, reach: reachability analysis, tracer: computation of a concrete trace.

Plan length	Time (s) perc	calib	perc + calib	# states perc	calib	perc + calib
50	.6	.1	2.1	662	269	2574
100	2.0	.5	7.7	1325	527	5513
150	4.9	.1	15.5	1978	769	8297
300	19.2	2.9	53.1	3953	1538	16476

Table 2: Average total transformation time and encoding size of five runs on plans with varying lengths.

As a first experiment, we consider plans of length 50 with a varying number of platform TAs (including multiple instances of $\mathcal{A}_{\texttt{com}}$ for different machines), as shown in Table 1. We can see that our approach is suitable to be used in typical robotics applications, given a limited number of platform components. The size of the encoded TAs is a major factor considering both the time spent by verifyta to load the TA and by taptenc to produce the encoding. Interestingly, the model checking task is not the limiting factor in the conducted tests, which we assume is due to the guidance that the explicit encoding gives to the underlying task. As a second experiment, we examined how the transformation scales as the plan length increases, as shown in Table 2. The results show that the presented approach is able to transform plans with 150 domain actions in reasonable time.

Lastly, we evaluated our approach in a domestic service robot scenario based on [Hofmann *et al.*, 2016]. The robot is tasked to clean up a table full of cups. Clean cups have to be placed on a shelf, dirty ones belong in a dishwasher. In the domain, the robot must (a) align to the furniture properly before picking up a cup, (b) scan surfaces before picking up or putting down a cup, (c) back off an object before moving away from it. We modified the domain to be suitable for temporal planning and considered scenarios with a varying number of cups on the table. We built a simplified domain without the special requirements (a)-(c) and decoupled them in a separate platform TA. Our plan transformation procedure then inserts platform actions to compute a feasible solution. We used TFD [Eyerich *et al.*, 2009] for planning both on the full and the simplified domain and configured it to stop when the first plan was found. The plans obtained from the simple domain were then used as input for taptenc. The results are summarized in Figure 6. From those we conclude that decoupling of platform constraints helped to reduce the overall runtime on complex problem instances significantly, while causing minor computational overhead on small tasks.

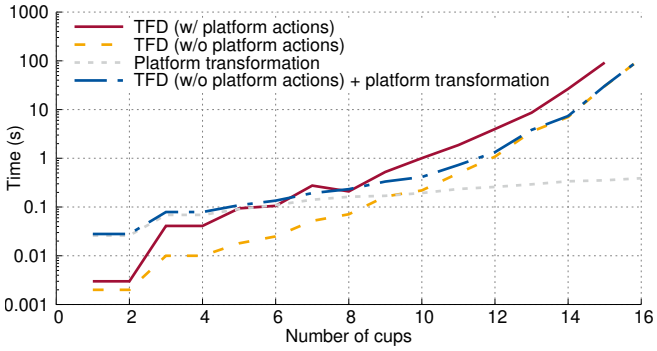

Figure 6: Planning with platform-specific actions versus planning on the simplified household domain followed by a transformation to include platform actions. Planning time was limited to 300s.

8 Conclusion

We developed a procedure to transform an abstract plan into an executable action sequence that considers all platform specifics that were ignored in the abstract domain. Such an abstract plan may be determined by a planner or by any other reasoning system that produces action sequences. Having platform components decoupled from the high-level domain allows to adapt low-level specifics without changing the high-level reasoner, as different concerns are clearly separated. It also simplifies the high-level domain, which may improve the reasoner's performance. We represent platform components as timed automata (TAs) and express the connection between such automata models and the abstract domain with constraints from a subset of Metric Temporal Logic (MTL). The proposed transformation first combines the abstract plan and the platform components into a single TA and then applies reachability analysis to determine an execution trace. We evaluated the approach on real-world scenarios from the RoboCup Logistics League and from a domestic service robot setting. Both scenarios showed that our approach is able to transform an abstract plan in reasonable time, even when dealing with large problems. The proposed approach also outperforms an alternative approach, where the platform details are directly encoded into the planning problem.

Acknowledgements

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants *GL-747/23-1*, *GRK 2236/1*, and under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612.

This work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215).

We thank Stefan Schupp for an insightful initial discussion on timed automata modelling.

References

[Aker *et al.*, 2011] Erdi Aker, Ahmetcan Erdogan, Esra Erdem, and Volkan Patoglu. Causal reasoning for planning and coordination of multiple housekeeping robots. In *Logic Programming and Nonmonotonic Reasoning*, 2011.

[Alur and Dill, 1994] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2), 1994.

[Bengtsson and Yi, 2004] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Springer, 2004.

[Bengtsson *et al.*, 1996] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, 1996.

[Bøgsted Poulsen and van Vliet, 2010] Danny Bøgsted Poulsen and Jonas van Vliet. *Concrete Delays for Symbolic Traces*. Master's thesis, Aalborg University, 2010.

[Cimatti *et al.*, 1997] Alessandro Cimatti, Enrico Giunchiglia, Fausto Giunchiglia, and Paolo Traverso. Planning via model checking: A decision procedure for AR. In *Recent Advances in AI Planning*, 1997.

[Dantam *et al.*, 2016] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems XII*, 2016.

[Dornhege *et al.*, 2012] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Springer, 2012.

[Erdem *et al.*, 2011] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*, 2011.

[Erdem *et al.*, 2012] Esra Erdem, Erdi Aker, and Volkan Patoglu. Answer set programming for collaborative housekeeping robotics: Representation, reasoning, and execution. *Intelligent Service Robotics*, 5(4), 2012.

[Erdem *et al.*, 2016] Esra Erdem, Volkan Patoglu, and Peter Schüller. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Communications*, 29(2), 2016.

[Eyerich *et al.*, 2009] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of ICAPS*, 2009.

[Garrett *et al.*, 2015] Caelan Garrett, Tomás Lozano-Pérez, and Leslie Kaelbling. FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the 11th International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015.

[Giunchiglia and Traverso, 2000] Fausto Giunchiglia and Paolo Traverso. Planning as Model Checking. In *Recent Advances in AI Planning*, 2000.

[Gravot *et al.*, 2005] Fabien Gravot, Stephane Cambon, and Rachid Alami. aSyMov: A Planner That Deals with Intricate Symbolic and Geometric Problems. In *Proc. of ISRR*, 2005.

[Henzinger, 1998] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In *Proc. of CONCUR*. Springer, 1998.

[Hertle *et al.*, 2012] Andreas Hertle, Christian Dornhege, Thomas Keller, and Bernhard Nebel. Planning with semantic attachments: An object-oriented view. In *Proc. of ECAI*, 2012.

[Hofmann and Lakemeyer, 2018] Till Hofmann and Gerhard Lakemeyer. A Logic for Specifying Metric Temporal Constraints for Golog Programs. In *Proc. of CogRob*, 2018.

[Hofmann and Lakemeyer, 2020] Till Hofmann and Gerhard Lakemeyer. Controller synthesis for Golog programs over finite domains with metric temporal constraints. Poster at KR, 2020.

[Hofmann *et al.*, 2016] Till Hofmann, Tim Niemueller, Jens Claßen, and Gerhard Lakemeyer. Continual Planning in Golog. In *Proc. of AAAI*, 2016.

[Hofmann *et al.*, 2018] Till Hofmann, Victor Mataré, Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. Constraint-Based Online Transformation of Abstract Plans into Executable Robot Actions. In *AAAI Spring Symposium: SIRLE*, 2018.

[Kim *et al.*, 2001] Phil Kim, Brian C Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. of IJCAI*, 2001.

[Konecnỳ *et al.*, 2014] S. Konecnỳ, Sebastian Stock, Federico Pecora, and Alessandro Saffiotti. Planning domain + execution semantics: A way towards robust execution. In *AAAI Spring Symposium on Qualitative Representations for Robots*, 2014.

[Koymans, 1990] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4), 1990.

[Largouët *et al.*, 2016] Christine Largouët, Omar Krichen, and Yulong Zhao. Temporal Planning with extended Timed Automata. In *Proc. of ICTAI*, 2016.

[Li *et al.*, 2012] Y. Li, J. Sun, J. S. Dong, Y. Liu, and J. Sun. Planning as Model Checking Tasks. In *Proc. of SEW*, 2012.

[Niemueller *et al.*, 2013] Tim Niemueller, Daniel Ewert, Sebastian Reuter, Alexander Ferrein, Sabina Jeschke, and Gerhard Lakemeyer. RoboCup Logistics League Sponsored by Festo: A Competitive Factory Automation Testbed. In *RoboCup Symposium*, 2013.

[Ouaknine and Worrell, 2005] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proc. of LICS*, 2005.

[Panek *et al.*, 2006a] Sebastian Panek, Sebastian Engell, and Olaf Stursberg. Scheduling and planning with timed automata. In *Computer Aided Chemical Engineering*, volume 21. Elsevier, 2006.

[Panek *et al.*, 2006b] Sebastian Panek, Olaf Stursberg, and Sebastian Engell. Efficient synthesis of production schedules by optimization of timed automata. *Control Engineering Practice*, 14, 2006.

[Srivastava *et al.*, 2014] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of ICRA*, 2014.

[Ziegert and Wehrheim, 2013] Steffen Ziegert and Heike Wehrheim. Temporal reconfiguration plans for self-adaptive systems. In *Software Engineering*, 2013.