Controlling Actions and Time in Parametric Timed Automata

Étienne André[*][†], Michał Knapik[‡], Wojciech Penczek[‡][§] and Laure Petrucci[*]
[*]LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, F-93430, Villetaneuse, France
[†]École Centrale de Nantes, IRCCyN, CNRS, UMR 6597, France
[‡]Institute of Computer Science, PAS, Warsaw, Poland
[§]University of Natural Sciences and Humanities, II, Siedlce, Poland

***Abstract*—Cyber-physical systems involve both discrete actions and real-time that elapses depending on timing constants. In this paper, we introduce a formalism for such systems containing both real-time parameters in linear timing constraints and switchable (Boolean) actions. We define a new approach for synthesizing combinations of parameter valuations and allowed actions, under which the system correctness is ensured when expressed in the form of a safety property. We apply our approach to a railway crossing system example with a malicious intruder potentially threatening the system safety.**

Keywords*—*Time synthesis, action synthesis, parametric timed automata, parametric safety

I. Introduction

Model checking consists in formally verifying whether a model of a system satisfies a property expressed using a formula. Beyond model checking, parameter synthesis consists in synthesizing valuations for some parameters so that the system satisfies the formula. Several recent works tackle parameter synthesis problems as they provide flexibility in the design of real-time systems while still guaranteeing their expected properties.

Among parametric synthesis approaches, two are of particular interest here as they consider different and complementary kinds of parameters: time (*e.g.* [AHV93]) and actions (*e.g.* [KMP15]). Operating on an extension of timed automata (finite state automata extended with clocks) where clocks can be compared to parameters, and properties of the system, synthesis of timing parameters provides a set of constraints the timing parameters should satisfy for the property to hold. On the other hand, starting from an (untimed) automaton and a property, action synthesis examines all possible behaviours in order to deduce which sets of actions should be enabled or disabled for the property to hold.

It is thus quite obvious that these two approaches are complementary as they consider very separate aspects of a system. Therefore, combining them is appealing. Furthermore, such a combination is worthy within a practical design process. Indeed, the designer of a system may be faced with multiple design choices, controlling some actions, or selecting components with specific time characteristics. Our aim by combining action and time synthesis is to provide designers with tools supporting their decision making. For example components with different timings might have different prices while controlling some actions is not always easy to set. Providing constraints on both time and actions allows to improve the choices according to criteria like components cost or feasibility.

Contribution: In this paper, we introduce a framework to synthesize both actions (seen as parameters) and timing constants (seen as another kind of parameters) for safety properties. We propose a procedure that, given a set of discrete states ("locations") to avoid, synthesizes a constraint on action and timing parameters such that, for any timing parameter valuation satisfying this constraint, and for any set of actions enabled or disabled according to the constraint, the system is safe, *i.e.* the discrete states to avoid are not reachable. We choose as a basis formalism of an extension of parametric timed automata (PTA), where actions can be (statically) controlled, *i.e.* may be enabled or disabled (once for all). Our procedure is a semi-algorithm, *i.e.* it is not guaranteed to terminate, but the result is correct whenever it does. We apply our approach to a railway crossing system example with a malicious intruder potentially threatening the system safety. We show that, depending on timing constants chosen, some actions (*e.g.* the fact that the intruder can commit certain actions) may have to be disabled in order to guarantee the system safety.

Related works: Synthesis of timing parameters for PTA has recently drawn a lot of attention: procedures (algorithms or semi-algorithms) were proposed to synthesize all parameters leading to some location [AHV93], preserve the time language [ACEF09], synthesize integer-valued parameters using bounded model checking techniques [KP12], or synthesize bounded valuations satisfying reachability or unavoidability [JLR15], [ALR15].

In addition to the action parameter synthesis proposed in [KMP15], synthesis of discrete parameters is often understood in terms of number of processes; the goal is to prove that a system is correct for any number of processes (possibly beyond a threshold). Common techniques include regular model checking (see *e.g.* [Abd12]) and verification of many identical processes (see *e.g.* [DSZ10]). Action synthesis is also conceptually related to the problem of (discrete) controller synthesis, such as in the Ramadge-Wonham framework [RW87].

This work is partially supported by the CNRS/PAN project "BehaPPi-BMC" (2015–2016), and by the ANR national research program "PACS" (ANR-14-CE28-0002).

This is the author version of the manuscript of the same name published in the proceedings of ACSD 2016. The final publication is available at http://ieeexplore.ieee.org/.

However, little has been done to combine different types of parameters, typically discrete (actions) and continuous (timing) together. A notable exception is [DKRT97] where constraints are derived to ensure the correctness of the bounded retransmission protocol (BRP); one of them involves a discrete parameter (an integer-valued maximum number of retransmissions) multiplied by a continuous timing parameter. However, the procedure proposed seems to be specific to this case study.

Outline of the paper: First, Section II recalls the notions and techniques our approach bases on, *i.e.* Parametric Timed Automata, time parameter synthesis and action synthesis. Then, Section III shows how they can be combined for synthesis of both time and actions. This approach is put into practice on a detailed example in Section IV, using tool support that provides adequate results for an engineer to tune his/her system design. Finally, Section V summarises the contributions and opens some perspectives.

II. Background

In this section, we recall the notions and techniques necessary to build our synthesis approach. First, the basic definitions of *Parametric Timed Automat*a which base on the use of *clocks* and *parameters* are recalled. PTA are extended to APTA (*Action-controllable Parametric Timed Automata*) that constitute a unified framework for time and action synthesis. The synthesis of time parameters leads to expressing *constraints* on their values in order to guarantee that the model satisfies the expected properties. Then, synthesis of actions is presented that allows for pruning the paths that invalidate properties by disabling controllable actions.

A. Clocks, Parameters, Actions, and Constraints

Let $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ denote the sets of Booleans, non-negative integers, integers, non-negative rational numbers, non-negative real numbers, respectively.

Throughout this paper, we assume a set $X = \{x_1, \dots, x_H\}$ of *clocks*, *i.e.* real-valued variables that evolve at the same rate. A *clock valuation* is a function $w : X \to \mathbb{R}_+$. We identify a clock valuation w with the *point* $(w(x_1), \dots, w(x_H))$. We write $X = 0$ for $\bigwedge_{1 \leq i \leq H} x_i = 0$. Given $d \in \mathbb{R}_+$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$. We also use a special zero-clock x_0, always equal to 0 (as in, *e.g.* [HRSV02]).

Let $P = \{p_1, \dots, p_M\}$ be a set of *timing parameters*, *i.e.* unknown timing constants. A *timing parameter valuation* v is a function $v : P \to \mathbb{Q}_+$. We identify a valuation v with the *point* $(v(p_1), \dots, v(p_M))$.

Let *ActVars* be a set of *action variables*. An *action valuation* is a function $\xi : \mathit{ActVars} \to \mathbb{B}$. Given ξ and an action variable α, we say that α is *enabled* in ξ if $\xi(\alpha) = \mathtt{true}$, and *disabled* otherwise. The set of all action valuations is denoted by $\mathit{ActVals}$. With a slight notational abuse we sometimes treat ξ as the set of the actions enabled by this valuation.

In the following, let xplt denote a linear term over $X \cup P$ of the form $\sum_{1 \leq i \leq H} \gamma_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in X$, $p_i \in P$, and $\gamma_i, \beta_j, d \in \mathbb{Z}$. Let plt denote a parametric linear term over P, that is a linear term without clocks ($\gamma_i = 0$ for all i).

An *AXT-constraint* (or constraint on the action parameters, clock parameters, and timing parameters) over $\mathit{ActVars} \cup X \cup P$ is defined by the following grammar:

$$\phi := \phi \wedge \phi \mid \neg \phi \mid \alpha \mid \mathit{xplt} \sim 0,$$

where $\sim \in \{<, \leq, \geq, >\}$, $\alpha \in \mathit{ActVars}$. Disjunction ($\vee$) is defined as usual; observe that, due to the negation, we allow for non-convex constraints. An *axt-valuation* is a function $at : \mathit{ActVars} \cup X \cup P \to \mathbb{B} \cup \mathbb{R}_+ \cup \mathbb{Q}_+$ such that $at(\alpha) \in \mathbb{B}$ for all $\alpha \in \mathit{ActVars}$, $at(x) \in \mathbb{R}_+$ for all $x \in X$, and $at(p) \in \mathbb{Q}_+$ for all $p \in P$.

A *zone* C is a convex AXT-constraint over $X \cup P$ (hence without action variables) such that each of its linear conjuncts can be written in the form $x_i - x_j \sim \mathit{plt}$, where $x_i, x_j \in X \cup \{x_0\}$. A *guard* g is a zone such that each of its linear conjuncts can be written in the form $x_i \sim \mathit{plt}$.

Given a timing parameter valuation v and a clock valuation w, we denote by $w|v$ the valuation over $X \cup P$ such that for all clocks x, $w|v(x) = w(x)$ and for all timing parameters p, $w|v(p) = v(p)$. Given a zone C, we use the notation $w|v \models C$ to indicate that valuating each clock variable x with $w(x)$ and each timing parameter p with $v(p)$ within C, evaluates to true. We say that C is *satisfiable* if $\exists w, v$ s.t. $w|v \models C$. We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over $X \cup P$ obtained from C by delaying all clocks by an arbitrary amount of time; this can be obtained by adding a new variable to all clocks, ensuring that this variable is non-negative, and eliminating it (see, *e.g.* [ACEF09]). Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. We denote by $C{\downarrow_P}$ the projection of C onto P, *i.e.* obtained by eliminating the clock variables (*e.g.* using Fourier-Motzkin elimination).

A *P-constraint* K is an AXT-constraint over P defined by inequalities of the form $\mathit{plt} \sim 0$. We denote by $\top_P$ (resp. $\bot_P$) the parametric constraint that corresponds to the set of all possible (resp. the empty set of) timing parameter valuations.

An *A-constraint* is an AXT-constraint that contains actions only. An a-valuation is defined naturally from actions to $\mathbb{B}$. We denote by $\top_A$ the A-constraint $\bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \mathtt{true}$, and by $\bot_A$ the A-constraint $\bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \mathtt{false}$. Let $at_\top$ denote the a-valuation assigning $\mathtt{true}$ to all variables.

An *AT-constraint* is an AXT-constraint without clock, *i.e.* with xplt instead of plt in the grammar. An at-valuation is defined similarly to axt-valuation, but without clocks. Given an AT-constraint AT and an at-valuation at, we write $at \models AT$ if the expression obtained by replacing in AT any parameter and action variable by its valuation as in at evaluates to true. We denote by $\top_{AT}$ the at-constraint corresponding to the set of at-valuations $\{at \mid \forall \alpha \in \mathit{ActVars} : at(\alpha) = \mathtt{true}\}$ (*i.e.* the set of valuations for which all timing parameter valuations are possible, and all action variables evaluate to true); we denote by $\bot_{AT}$ the at-constraint $\bot_P \wedge \bigwedge_{\alpha \in \mathit{ActVars}} \alpha = \mathtt{false}$, *i.e.* the constraint satisfied by no timing parameter valuation and only by the false action variables.

B. Action-Controllable Parametric Timed Automata

Parametric timed automata (PTA) extend timed automata with parameters within guards and invariants in place of inte-

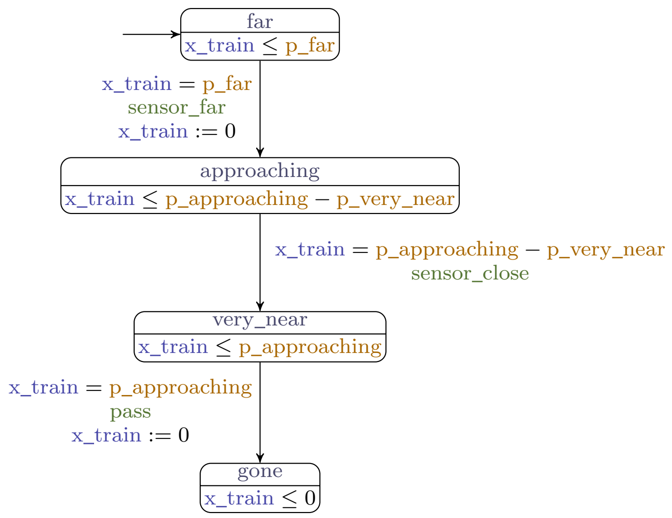


Figure 1: A parametric timed automaton

ger constants [AHV93]. We extend this concept by allowing controllable actions.

Definition 1. *An Action-controllable PTA (APTA) is a tuple $\mathcal{APTA} = (\Sigma, \mathit{ActVars}, L, l_0, X, P, I, E)$, where: i) Σ is a finite set of non-controllable actions, ii) ActVars is a finite set of action variables (that can be controlled), iii) L is a finite set of locations, iv) $l_0 \in L$ is the initial location, v) X is a set of clocks, vi) P is a set of parameters, vii) I is the invariant function, assigning to every $l \in L$ an invariant $I(l)$, viii) E is a set of edges $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and target locations, g is a guard, $a \in \Sigma \cup \mathit{ActVars}$, and $R \subseteq X$ is a set of clocks to be reset.*

A PTA is an APTA without action variables.

a) Valuation of an APTA: Given an APTA $\mathcal{APTA} = (\Sigma, \mathit{ActVars}, L, l_0, X, P, I, E)$ and an a-valuation ξ, we denote by $\xi(\mathcal{APTA})$ the PTA $\mathcal{PTA} = (\Sigma', L, l_0, X, P, I, E')$, where $\Sigma' = \Sigma \cup \{\alpha \in \mathit{ActVars} \mid \xi(\alpha) = \mathtt{true}\}$[1] and $E' = E \setminus \{(l, g, \alpha, R, l') \in E \mid \alpha \in \mathit{ActVars} \wedge \xi(\alpha) = \mathtt{false}\}$.

Example: Figure 1 presents a simple PTA modelling a train circulating on a railway with a gate. This model will be enhanced in the case study of Section IV. It uses 3 parameters (p_far, p_approaching and p_very_near) and 1 clock (x_train). Initially, the train is far from the gate and remains far for as long as p_far. Then it triggers sensor_far and starts approaching the gate, resetting the clock. The approach lasts p_approaching but when only p_very_near time units remain, sensor_close is triggered. Finally, the train can pass the gate and be gone.

b) Valuation of a PTA: Given a PTA $\mathcal{PTA}$ and a timing parameter valuation v, we denote by $v(\mathcal{PTA})$ the *timed automaton (TA)* obtained from $\mathcal{PTA}$ by replacing in $\mathcal{PTA}$ each timing parameter p with its valuation $v(p)$.

Given an at-valuation at, let v be the projection of at onto P, and ξ its projection onto $\mathit{ActVars}$. Then given an APTA $\mathcal{APTA}$, we denote by $at(\mathcal{APTA})$ the TA $v(\xi(\mathcal{APTA}))$.

[1]Strictly speaking, action variables and actions are of different types; we assume that each action variable valuated with `true` is added to the set of actions.

c) Concrete semantics of a TA:

Definition 2 (Semantics of a TA)**.** *Given a PTA $\mathcal{PTA} = (\Sigma, L, l_0, X, P, I, E)$, and a timing parameter valuation v, the concrete semantics of $v(\mathcal{PTA})$ is given by the timed transition system $(Q, q_0, \rightarrow)$, with*

- $Q = \{(l, w) \in L \times \mathbb{R}_+^H \mid w|v \models I(l)\}$, $q_0 = (l_0, X = 0)$
- *$\rightarrow$ consists of the discrete and continuous transition relations:*
  - *discrete transitions: $(l, w) \xrightarrow{e} (l', w')$, if $(l, w), (l', w') \in Q$, there exists $e = (l, g, a, R, l') \in E$, $w' = [w]_R$, and $w|v \models g$.*
  - *delay transitions: $(l, w) \xrightarrow{d} (l, w + d)$, with $d \in \mathbb{R}_+$, if $\forall d' \in [0, d], (l, w + d') \in Q$.*

Moreover we write $(l, w) \overset{e}{\mapsto} (l', w')$ for a sequence of discrete step and time elapsing where $((l, w), e, (l', w')) \in \mapsto$ if $\exists d, w'' : (l, w) \xrightarrow{e} (l', w'') \xrightarrow{d} (l', w')$.

Given a TA $v(\mathcal{PTA})$ with the concrete semantics $(Q, q_0, \rightarrow)$, we refer to the states of Q as the *concrete states* of $v(\mathcal{PTA})$. A concrete run of $v(\mathcal{PTA})$ is an alternating sequence of concrete states of $v(\mathcal{PTA})$ and edges starting from the initial concrete state q_0 of the form $q_0 \overset{e_0}{\mapsto} q_1 \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} q_m$, such that for all $i = 0, \dots, m-1$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \mapsto$. Given a state $q = (l, w)$, we say that q is reachable (or that $v(\mathcal{PTA})$ reaches q) if q belongs to a run of $v(\mathcal{PTA})$. By extension, we say that l is reachable in $v(\mathcal{PTA})$.

d) Symbolic semantics of a PTA: Let us now recall the symbolic semantics of PTA. A symbolic state is a pair (l, C) where $l \in L$ is a location, and C its associated parametric zone. The initial symbolic state of $\mathcal{PTA}$ is $s_0 = (l_0, (X = 0)^{\nearrow} \wedge I(l_0))$.

The symbolic semantics exploits the Succ operation. Given a symbolic state $s = (l, C)$ and an edge $e = (l, g, a, R, l')$, $\mathsf{Succ}(s, e) = (l', C')$, with $C' = \big([(C \wedge g)]_R\big)^{\nearrow} \cap I(l')$. That is, we first intersect the constraint of the source symbolic state with the outgoing guard, then reset the clocks of the transition, then let time elapse, and finally intersect with the invariant of the target location.

A symbolic run of a PTA is an alternating sequence of symbolic states and edges starting from the initial symbolic state, of the form $s_0 \overset{e_0}{\Rightarrow} s_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0, \dots, m-1$, $e_i \in E$, and s_{i+1} belongs to $\mathsf{Succ}(s_i, e)$. Given a symbolic state s, we say that s is reachable if s belongs to a run of $\mathcal{PTA}$. In the following, we simply refer to symbolic states belonging to a run of $\mathcal{PTA}$ as symbolic states (or, when clear from the context, just as states) of $\mathcal{PTA}$.

Given a concrete (respectively symbolic) run $(l_0, X = 0) \overset{e_0}{\mapsto} (l_1, w_1) \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} (l_m, w_m)$ (respectively $(l_0, C_0) \overset{e_0}{\Rightarrow} (l_1, C_1) \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} (l_m, C_m)$), its corresponding discrete sequence is $l_0 \overset{e_0}{\Rightarrow} l_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} l_m$. Two runs (concrete or symbolic) are said to be equivalent if their corresponding discrete sequences are equal.

C. Mixed Transition Systems

Mixed Transition Systems [PR06] (MTS) are essentially Kripke structures with transitions labelled by actions.

Definition 3 (MTS). *Let $\mathcal{PV}$ be a set of propositional variables. A* mixed transition system *(MTS) is a 5-tuple $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$, where: i) $\mathcal{S}$ is a non-empty finite set of states, ii) $s^0 \in \mathcal{S}$ is the initial state, iii) Σ is a non-empty finite set of actions, iv) $\mathcal{T} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation, v) $\mathcal{V}_s : \mathcal{S} \to 2^{\mathcal{PV}}$ is a (state) valuation function.*

We write $s \xrightarrow{a} s'$ if $(s, a, s') \in \mathcal{T}$. Let $\chi \subseteq \Sigma$ be a nonempty set of actions and $\pi = (s_0, a_0, s_1, a_1, \ldots)$ be a finite or infinite sequence of interleaved states and actions. By $|\pi|$ we denote the number of the states of π if π is finite, and ω if π is infinite. A sequence π is a *path* over χ iff $s_i \xrightarrow{a_i} s_{i+1}$, $a_i \in \chi$ for each $i < |\pi|$, and either π is infinite or its final state does not have a χ-successor state in $\mathcal{S}$, *i.e.* $\pi = (s_0, a_0, s_1, a_1, \ldots, s_m)$ for some $m \in \mathbb{N}$ and there is no $s' \in \mathcal{S}$ and $a \in \chi$ such that $s_m \xrightarrow{a} s'$. In general, for all $i \in \mathbb{N}$, we denote by π_i the state of π at rank i ($\pi_i = s_i$ in the sequence above).

The set of all paths over χ in a mixed transition system $\mathcal{M}$ is denoted by $\Pi(\mathcal{M}, \chi)$, whereas the set of all paths $\pi \in \Pi(\mathcal{M}, \chi)$ starting from a given state $s \in \mathcal{S}$ is denoted by $\Pi(\mathcal{M}, \chi, s)$. We omit the symbol $\mathcal{M}$ if it is clear from the context, simply writing $\Pi(\chi)$ and $\Pi(\chi, s)$.

D. Parametric Reachability

In [KMP15] a parametric extension of *Action-Restricted Computation Tree Logic* [PR06], ARCTL, is presented. The language of ARCTL consists of CTL-like branching-time formulae where each path quantifier is subscripted with a set of actions. The subscripts are used in path selection, for example: $E_{\{left,right\}}G(E_{\{forward\}}F safe)$ may be read as *"there exists a path over left and right, on which it holds globally that a state satisfying safe is reachable along some path over forward"*. In Parametric ARCTL (denoted by pmARCTL), special variables ranging over sets of actions are allowed in the superscripts. In this paper we deal only with the parametric reachability, *i.e.* with the formulae of type $E_Z F pv$, where $pv \in \mathcal{PV}$. As we do not permit for nesting of modalities, the superscript is usually omitted and we simply write $EFpv$.

We interpret the formulae with respect to action valuations. Formally, given a MTS $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$, and $pv \in \mathcal{PV}$, we say that $EFpv$ holds in $s \in \mathcal{S}$ under an a-valuation $\xi \in ActVals$ iff there exists a path $\pi \in \Pi(\xi, s)$ such that $pv \in \mathcal{V}_s(\pi_i)$ for some $i \in \mathbb{N}$. This is denoted by $\mathcal{M}, s \models_\xi EFpv$.

E. Action Synthesis

Action synthesis for $\varphi \in$ pmARCTL builds an indicator function $f_\varphi \colon \mathcal{S} \to 2^{ActVals}$ such that for all $s \in \mathcal{S}$ we have: $s \models_\xi \varphi$ iff $\xi \in f_\varphi(s)$. Intuitively, $f_\varphi(s)$ consists of all action valuations under which φ holds in state s. As shown in [KMP15], similarly to CTL, pmARCTL has a fixed-point characterisation, which allows for building efficient procedures for action synthesis, based on Binary Decision Diagrams (BDDs). Such a framework for action synthesis was implemented in the standalone tool SPATULA [Kna]. In what follows, given an MTS $\mathcal{M}$ and $pv \in \mathcal{PV}$, by $\mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ we denote the result of the procedure synthesising all minimal sets of actions under which $EFpv$ holds in the initial state of $\mathcal{M}$ (*e.g.* using SPATULA). Formally: $\mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ is the smallest subset of 2^Σ such that: (1) $\xi \in \mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ implies $\mathcal{M}, s^0 \models_\xi EFpv$, and (2) if $\mathcal{M}, s^0 \models_{\xi'} EFpv$, then there exists $\xi \in \mathsf{synthpmARCTL}(\mathcal{M}, EFpv)$ satisfying $\xi \subseteq \xi'$.

III. The Mixed Synthesis Problem: Combining Timing and Action Synthesis

A. Objective

Our main goal is to synthesize parameters seen as both timing constants (à la [AHV93]) and switchable actions that can be enabled or disabled once for all (à la [KMP15]).

Action and time synthesis problem:
INPUT: an APTA $\mathcal{APTA}$ and a set of locations L_{bad}
PROBLEM: Synthesise an AT-constraint AT such that, for all $at \models AT$, no location in L_{bad} is reachable in $at(\mathcal{APTA})$.

As this is a safety problem, it is bad-state driven. Hence, in the following, we refer to locations of L_{bad} as *bad locations*, and to the states the location of which belongs to L_{bad} as *bad states*.

B. General Approach

We follow an approach where we first handle a parametric timed model in which we assume all actions to be enabled; then, from the state space of this model, we synthesise actions together with timing parameter constraints. Enabling all actions in the first phase allows for synthesising all timing parameter constraints for all possible actions, and then combining these constraints with action parameters in the second phase.

The steps of this approach are as follows:

1) Model the system using an APTA.
2) Considering the PTA with all actions enabled, generate a sufficient subpart of the state space reaching bad locations.
3) Using this state space seen as an MTS, synthesize actions ensuring unreachability of the bad states.
4) Process the result to get a linear constraint both on timing parameters and actions.

Each of these steps is detailed in the following subsections.

C. A Labelled Parametric State Space

Starting from an APTA model $\mathcal{APTA}$, we first enable all actions, so as to get the PTA $at_\top(\mathcal{APTA})$. We then generate an MTS augmented with parameter constraints. Each state of this MTS corresponds to a symbolic state of the PTA; each transition of this MTS corresponds to a transition of the state space with the action label. We do not generate the entire state space, but locally stop the exploration whenever a bad state is reached (while still exploring other branches).

Algorithm 1 presents our semi-algorithm genMTS to generate the MTS from the PTA $at_\top(\mathcal{APTA})$. It takes as arguments a PTA and the set L_{bad} of bad locations, and builds

Algorithm 1: genMTS($\mathcal{PTA}, L_{bad}$)

```
input  : PTA $\mathcal{PTA}$ with initial state $s_0$
input  : Set of locations $L_{bad}$
output: MTS $\mathcal{M}$
1  $S \leftarrow \emptyset$ ;   $S_{queue} \leftarrow \{s_0\}$ ;
2  while $S_{queue} \neq \emptyset$ do
3      Pick a state $s = (l, C)$ from $S_{queue}$
       /* If $s$ is bad state */
4      if $l \in L_{bad}$ then
           /* Label this state as bad */
5          $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}\}$
6      else
           /* Compute successors */
7          foreach state $s' \in \mathsf{Succ}(s)$ such that $s \overset{a}{\Rightarrow} s'$ do
               /* Enqueue unless met earlier */
8              if $s' \notin S_{queue} \cup S$ then
9                  $S_{queue} \leftarrow S_{queue} \cup \{s'\}$
               /* Add transition to MTS */
10             $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s, a, s')\}$
11 return $\mathcal{M} = (S, \{s_0\}, \Sigma, \mathcal{T}, \mathcal{V}_s)$
```

(a part of) the symbolic semantics of the PTA. Algorithm genMTS maintains two local variables: the set S of processed states, and the set S_{queue} of states to be processed. Observe that if S_{queue} is implemented using a queue, then genMTS can be seen as a breadth-first search algorithm. Additionally, $\mathcal{T}$ denotes the list of transitions of the resulting MTS (initially empty) and $\mathcal{V}_s$ its labelling function (initially assigning to each state no label).

While the set of states to be processed is not empty (line 2), genMTS selects an unprocessed state s (line 3). If this state is a bad state (line 4), it is labelled as such (line 5); we use notation $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}\}$ to denote that function $\mathcal{V}_s$ is updated so that $\mathbf{bad}$ is the (unique) label of state s. If this state is a bad state, its successors are not computed, *i.e.* the algorithm stops exploring this branch. Otherwise, the successors of s are computed (line 7) and, for each of them, they are added to the set of states to be processed (line 9), unless they were met earlier. Finally, the corresponding transitions are added to the MTS (line 10).

Eventually genMTS returns a MTS made of the set S of states, the initial state s_0, all actions of the PTA, the transitions of the state space, and the function $\mathcal{V}_s$ that labels the bad states.

Example 1. *Consider the PTA $\mathcal{PTA}$ in Figure 2a. Assume $L_{bad} = \{l_4, l_5\}$. The MTS generated by* genMTS($\mathcal{PTA}, L_{bad}$) *is given in Figure 2b; note that, for better understanding, for a state (l, C), we give the location l in the upper part, and the constraint projected onto P (i.e. $C{\downarrow_P}$) in the lower part; moreover, we add the label (*$\mathbf{bad}$*) to the upper part after the location name. Observe that locations beyond a bad location in Figure 2a (e.g. l_7) are not part of the MTS, as* genMTS *does not explore states beyond bad states.*

D. Synthesising Action and Timing Parameters

Let us now synthesise the actions and timing parameters ensuring the (un)reachability of the bad states. Let us consider the pmARCTL formula $EF\mathbf{bad}$, *i.e.* the formula stating that there exists a path eventually reaching a bad state. From the parameter valuations reaching the bad state, we can easily retrieve the complement set for which the bad state is unreachable. Applying directly action synthesis techniques of [KMP15] to the MTS obtained from Algorithm genMTS would not be satisfactory: this would yield the minimal sets of actions for which bad states are reachable, independently of the timing parameter valuations.

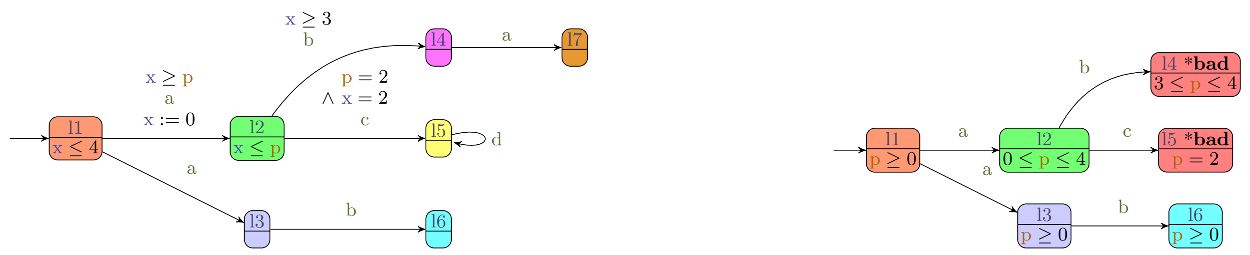
Here, we aim at synthesising an at-constraint on the timing parameters and action parameters guaranteeing the unreachability of L_{bad}. More precisely, for each occurrence of a bad state on a given path, at least one of the actions in each minimal set of actions leading to this state should be disabled, or the timing parameter constraint allowing this reachability should be negated. Let us explain this part in more details in the following.

Algorithm 2: synthActTime($\mathcal{M}$)

```
input  : MTS $\mathcal{M} = (\mathcal{S}, s^0, \Sigma, \mathcal{T}, \mathcal{V}_s)$
output: AT-constraint guaranteeing unreachability
        of $L_{bad}$
1  $AT \leftarrow \bot_{AT}$ ;   $i \leftarrow 1$
2  foreach $s = (l, C) \in \mathcal{S}$ such that $\mathbf{bad} \in \mathcal{V}_s(s)$ do
       /* Generate a dedicated label for $s$ */
3      $\mathcal{V}_s(s) \leftarrow \{\mathbf{bad}_i\}$
4      $ActsSets \leftarrow \mathsf{synthpmARCTL}(\mathcal{M}, EF\mathbf{bad}_i\}$
5      $AT \leftarrow AT \vee \Big(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets} \big(\bigwedge_{\alpha \in Acts} \alpha\big)\Big)$
6      $i \leftarrow i + 1$
7  return $\neg AT$
```

Algorithm 2 presents our procedure to synthesize an AT-constraint guaranteeing the non-reachability of L_{bad}. The algorithm synthActTime($\mathcal{M}$) maintains two variables: the AT-constraint (initially false), and an integer i (just used to generate unique labels). Note that the AT-constraint in fact ensures the *reachability* of at least one of the bad states; it eventually is negated to ensure non-reachability. For each bad state of the MTS (*i.e.* labelled with $\mathbf{bad}$), we generate a unique label for this state (line 3). Then, we synthesize the set of minimal sets of actions for this unique label to be reachable, *i.e.* such that this state s is reachable. We then refine the AT-constraint as follows: the constraint is augmented (in the sense of the disjunction) with a constraint ensuring the reachability of the current state s, *i.e.* the timing parameter constraint allowing the reachability of this state, together with at least one of the minimal sets of actions allowing its reachability (line 5). The negation of the AT-constraint is eventually returned (line 7).

Example 2. *Consider again the APTA $\mathcal{APTA}$ in Figure 2a, and assume all actions are controllable, i.e. $ActVars = \{a, b, c, d\}$. The MTS $\mathcal{M}$ generated by* genMTS($at_\top(\mathcal{APTA}), L_{bad}$) *is given in Figure 2b. Let us apply* synthActTime *to $\mathcal{M}$. Initially, AT is $\bot_{AT}$. Let us consider as the first bad state the unique state whose location is l_4 in Figure 2b.* synthActTime *adds to that state a label* $\mathbf{bad}_1$*. Then, the result of* synthpmARCTL($\mathcal{M}, EF\mathbf{bad}_1$) *is $\{a, b\}$. Hence,* synthActTime *adds to AT the following at-constraint: $3 \leq p \leq 4 \wedge a \wedge b$. Let us consider the*



(a) A sample PTA (b) The generated MTS

Figure 2: A PTA and its MTS

second bad state, i.e. the unique state whose location is l_5. synthActTime *adds to that state a label* $\mathbf{bad}_2$. *Then, the result of* synthpmARCTL$(\mathcal{M}, EF\mathbf{bad}_2)$ *is* $\{a, c\}$. *Hence,* synthActTime *adds to* AT *the following at-constraint:* $p = 2 \wedge a \wedge c$. *Eventually,* synthActTime *returns the negation of* AT*, i.e.:*

$$(p < 3 \vee p > 4 \vee \neg a \vee \neg b) \wedge (p \neq 2 \vee \neg a \vee \neg c).$$

Sufficient conditions to ensure the satisfaction of this constraint are for example $p = 1$*, or* a *is disabled, or* $p = 2$ *and* c *is disabled.*

Let us now examine a variant of this example where action c *is not controllable. The final result obtained is:*

$$(p < 3 \vee p > 4 \vee \neg a \vee \neg b) \wedge (p \neq 2 \vee \neg a).$$

In this case, $p = 1$*, or* a *is disabled are still sufficient conditions, while* $p = 2$ *is not.*

E. Soundness and Completeness

We now prove that our approach is sound and complete.

Theorem 1 (soundness)**.** *Let* $\mathcal{APTA}$ *be an APTA, and* L_{bad} *be the set of locations to avoid. Assume* genMTS$(at_\top(\mathcal{APTA}), L_{bad})$ *terminates with result* $\mathcal{M}$*. Let* $Res =$ synthActTime$(\mathcal{M})$*. Then, for all* $at \models Res$*, locations* L_{bad} *are unreachable in* $at(\mathcal{APTA})$*.*

Proof: We first need to recall two lemmas relating symbolic and concrete runs (proved in, *e.g.* [HRSV02], [ACEF09]).

Lemma 1. *Let* $\mathcal{PTA}$ *be a PTA, and* v *be a timing parameter valuation. Let* ρ *be a run of* $\mathcal{PTA}$ *reaching a symbolic state* (l, C)*. Then there exists an equivalent run in the TA* $v(\mathcal{PTA})$ *reaching a concrete state* (l, w)*, for some* $w \models C$*, iff* $v \models C{\downarrow_P}$*.*

Lemma 2. *Let* $\mathcal{PTA}$ *be a PTA, and* v *be a timing parameter valuation. Let* ρ *be a run of the TA* $v(\mathcal{PTA})$ *reaching a concrete state* (l, w)*. Then there exists an equivalent run in* $\mathcal{PTA}$ *reaching a symbolic state* (l, C) *such that* $v \models C{\downarrow_P}$*.*

Let $Res =$ synthActTime$(\mathcal{M})$. Let us reason by *reductio ad absurdum*, and assume that a state $s = (l, w)$ with $l \in L_{bad}$ is reachable in $at(\mathcal{APTA})$, for some concrete run. Let ξ be the projection of at on actions. Assume this state is the first bad state along this run (if it is not, then consider the first bad state instead of s). From Lemma 2, we have there exists an equivalent symbolic run in the PTA $\xi(\mathcal{APTA})$ reaching a state (l, C) for some C such that $v \models C{\downarrow_P}$. Since genMTS only stops exploring a branch whenever a symbolic state has no successor or when a bad state is met, then (l, C) was necessarily met by genMTS, and hence is part of $\mathcal{M}$, where it is labelled with $\mathbf{bad}$. Then, in synthActTime$(\mathcal{M})$, the set $ActsSets$ of minimal sets of actions leading to (l, C) is synthesized, and the constraint AT is updated with $\big(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets}(\bigwedge_{\alpha \in Acts} \alpha)\big)$. In the end, synthActTime returns the negation of AT, *i.e.* a conjunction of the constraints of the form $\big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$. This is to say that a valuation satisfying the result of synthActTime either does not satisfy $C{\downarrow_P}$, or at least one action in each set of minimal sets of actions allowing the reachability of (l, C) is disabled. Recall that we assumed $at \models$ synthActTime$(\mathcal{M})$. If the former statement ($at \models \neg C{\downarrow_P}$) is true, then from Lemma 1 no concrete run of $at(\mathcal{APTA})$ reaches a concrete state equivalent to (l, C), which contradicts the hypothesis that (l, w) is reachable in $at(\mathcal{APTA})$. Otherwise, if the latter statement is true, then one action is disabled in any symbolic path that could lead to (l, C), and hence again this contradicts the hypothesis that (l, w) is reachable in $at(\mathcal{APTA})$. ■

Theorem 2 (completeness)**.** *Let* $\mathcal{APTA}$ *be an APTA, and* L_{bad} *be the set of locations to avoid. Assume* genMTS$(at_\top(\mathcal{APTA}), L_{bad})$ *terminates with result* $\mathcal{M}$*. Let* $Res =$ synthActTime$(\mathcal{M})$*. Let* at *be an at-valuation such that locations* L_{bad} *are unreachable in* $at(\mathcal{APTA})$*. Then* $at \models Res$*.*

Proof: Let at be an at-valuation such that locations L_{bad} are unreachable in $at(\mathcal{APTA})$. Let us reason by *reductio ad absurdum*, and assume that $at \not\models Res$. Recall that synthActTime returns the negation of AT, *i.e.* a conjunction of constraints, each corresponding to the handling of one bad state. Since $at \not\models Res$, then there exists at least one such constraint that is not satisfied by at. Consider one of these constraints not satisfied by at; this constraint is of the form $\big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$, where C is the constraint of a symbolic state (l, C) (with $l \in L_{bad}$) of $\top_A(\mathcal{APTA})$, and $ActsSets$ is the set of minimal sets of actions allowing this set to be reachable. Since we have $at \not\models \big(\neg C{\downarrow_P} \vee \bigwedge_{Acts \in ActsSets}(\bigvee_{\alpha \in Acts} \neg\alpha)\big)$ then we have $at \models \big(C{\downarrow_P} \wedge \bigvee_{Acts \in ActsSets}(\bigwedge_{\alpha \in Acts} \alpha)\big)$. That is, $at \models C{\downarrow_P}$ and there exists a minimal set of actions in $ActsSets$ that are all enabled in at. Consider the symbolic run leading to (l, C) using this minimal set of actions. Since $at \models C{\downarrow_P}$, from

Lemma 1 we have that there exists an equivalent concrete run in $at(\mathcal{APTA})$. In addition, since all actions along this run are enabled in at, then l is reachable in $at(\mathcal{APTA})$, which violates the assumption that all locations of L_{bad} are unreachable in $at(\mathcal{APTA})$. Hence $at \models Res$. ■

F. Discussion: Approximated Synthesis

Most decision problems are undecidable for PTA (see *e.g.* [And15] for a survey), including the emptiness of the valuation set such as a given set of locations is reachable. Hence, exact synthesis is ruled out. Still, subclasses of PTA such that exact synthesis can be achieved exist. First, for acyclic systems (*i.e.* such that no loops exist, or with a limited number of iterations only), most problems are decidable, and synthesis can often be achieved. Applications include most hardware problems, as empirical knowledge show that most hardware verification problems can be carried out for a limited number of hardware clock cycles (typically two); some scheduling problems can also be analysed over a limited number of periods that can be statically computed beforehand. In addition, reducing the number of clocks and/or of parameters can lead to decidability (see [And15]).

However, in the general case, exact synthesis is not necessarily possible, and algorithms may not terminate. Without surprise, genMTS$(\top_A(\mathcal{APTA}), L_{bad})$ may not terminate, since it computes (a subpart of) the state space of a PTA, which is infinite, and for which no finite abstraction can be computed (since the underlying decision problem is undecidable). However, approximations can be used. A possible approximation is to bound the analysis (*e.g.* using a maximum number of explored states, a maximum exploration depth, or a maximum computation time), in which case the state space explored by genMTS is a subset of the actual state space; in that case, the parameter constraint ensuring the reachability of the bad locations is an under-approximation. Using this constraint in synthActTime is not valid: because of the negation (line 5 in Algorithm 2), synthActTime would yield an over-approximation of the good valuations, violating the soundness (Theorem 1). A safe option is to consider that any unexplored state is a bad state, *i.e.* to add to $C{\downarrow_P}$ at line 5 in Algorithm 2 the projection to P of any state (l', C') the successors of which is unexplored. In that case, the result output by synthActTime is a sound under-approximation of the safe action and parameter valuations.

Note however that, in all our experiments, genMTS always terminated without the need to use approximations.

IV. Application and Evaluation

In this section, we apply the approach presented in Section III to a concrete example of an enhanced version of the classical railway gate controller (see, *e.g.* [AHV93]). We start with the detailed description of our model.

A. Running example: An intruder in a railway gate controller

The system itself comprises two components: the train, and the gate controller. A sensor is located far from the gate, which sends a signal to inform the controller that a train is approaching (so that the controller starts lowering the gate). Another sensor is located close to inform the gate that the train

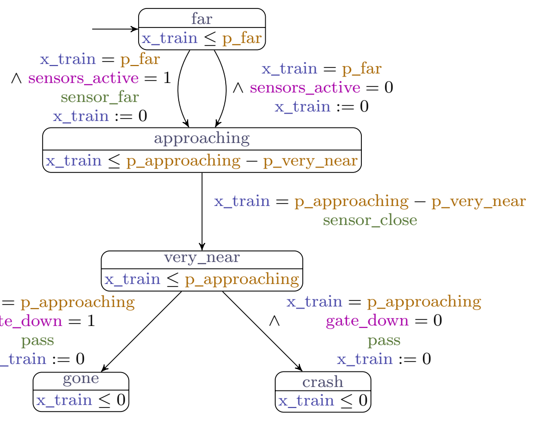


Figure 3: APTA train

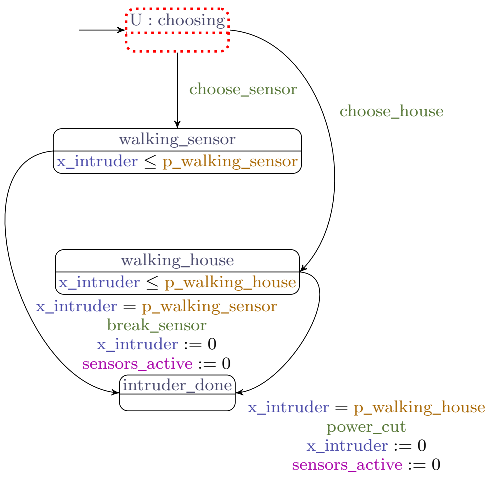


Figure 5: APTA intruder

is very near, triggering an emergency lowering if the gate is not yet closed.

Let us consider that an intruder can sabotage the system by either disabling the far sensor, thus preventing the detection of an approaching train, or cut the power supply in the gate house, hence preventing the gate to lower. Moreover, the gate has a security mechanism that automatically lowers the gate some time after the power has been shut off, and another security mechanism with an emergency gate lowering when the train is very near.

This system is modelled by three APTAs that model the train (Figure 3), the gate (Figure 4) and the intruder (Figure 5) respectively. They comprise parameters (in brown), clocks (in blue), actions (in green), and discrete values[2] (in purple). Each location has a name indicated in the upper part of the node, and its associated invariant is in the lower part.

The parameters allow for analysing a general model, where the values are not yet known at the design time:

- train parameters:

[2]Though not part of the (A)PTA model, discrete values are integer-valued variables often supported by tools, and that are syntactic sugar for locations.

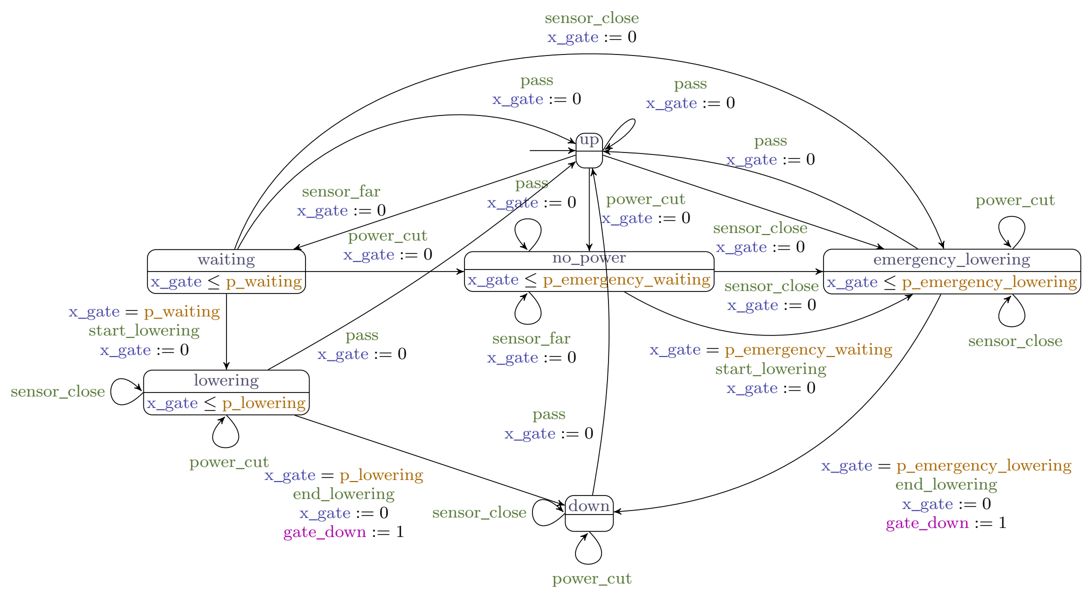


Figure 4: APTA gate

- p_far: time taken between being far and approaching;
- p_approaching: time between the approach stage and passing the gate;
- p_very_near: time between being very near and passing the gate.

- gate parameters:
  - p_waiting: time between sensor activation and beginning of gate lowering;
  - p_emergency_waiting: time between power shutdown and beginning of gate lowering;
  - p_lowering: time to lower the gate;
  - p_emergency_lowering: time to lower the gate in emergency mode.
- intruder parameters:
  - p_walking_sensor: time to walk and disable the sensor;
  - p_walking_house: time to walk and cut the power.

Clocks are associated with the train (x_train), the gate (x_gate) and the intruder (x_intruder).

Let us now detail the train APTA in Figure 3. Initially, the train is far, and remains far until it has travelled for p_far time. At that date, it becomes approaching. Note that there are two similar transitions from location far to location approaching. One considers the sensor is active, and thus triggers the sensor_far action, while the other caters for the inactive sensor case (disabled by the intruder). In both cases, the train clock is reset to count time elapsed from then on. When the train becomes very_near, the APTA changes location again, by triggering sensor_close. Note that here there is only one transition as the intruder cannot tamper with the close sensor. Then the last part of the train approach takes place, and when it reaches the gate, it results in two different states according to the status of the gate. If gate_down is false, there is a crash, otherwise the train resumes its journey. Both end states block time to avoid unnecessary state space explosion during the analysis.

The intruder (Figure 5) starts in an urgent location where (s)he has to choose immediately between the two sabotage possibilities. Although the actions are different (break_sensor and power_cut), they both deactivate the sensors as the communication between sensors and gate is broken.

The gate APTA is presented in Figure 4. Initially the barrier is up, and can change on reception of a sensor_far or a sensor_close signal or detecting a power_cut, leading to locations waiting, emergency_lowering or no_power, respectively. In all cases, the clock starts ticking. After some time, the gate starts lowering until it is down. The power_cut can also happen anytime, leading to the no_power location. After a power shutdown, the emergency procedure states that if the power is not back an emergency_lowering takes place. It is also the case if the sensor_close is triggered while the train should still be approaching. Finally, when the train has passed the crossing, the barrier gets back up.

In the initial configuration, the train is far, the intruder is ready for choosing and the gate is up. The discrete variables are also initialised: gate_down $= 0$, sensors_active $= 1$.

B. *Tool Support*

Our toolchain, depicted in Figure 6, consists of three tools coordinated by the interface IF.

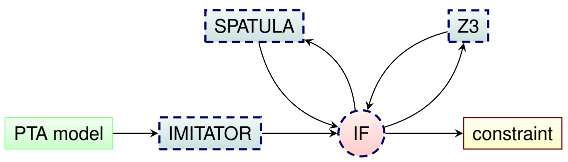


Figure 6: Tool support

The synthesis process starts with the generation of the state space (Section III-C) by using IMITATOR [AFKS12], a tool for parameter synthesis for real-time systems modelled by PTA. The version used is 2.7.3 (build 1338). IF accepts the output of IMITATOR and prepares a set of programs for SPATULA [KMP15], a tool implementing the action synthesis for pmARCTL. In order to transform the timed state space graph into a mixed transition system, IF performs constraint simplification and analysis, employing the Z3 SMT-solver [dMB08].

C. Experimental Results and Interpretation

We synthesise constraints under which the location crash is avoidable, despite of the presence of the intruder, depending on the extent of the capabilities of the latter, *i.e.* the enabled actions. For practical purposes, our toolchain accepts the set of *controllable actions*, denoted by $Cntr$. Intuitively, these are the only actions that are allowed to be switched on and off in the considered model. As the goal of analysis is the safety from the activities of the malicious intruder, we assume that its actions constitute the set of controllable transitions, *i.e.* $Cntr = \{\text{choose\_house}, \text{choose\_sensor}, \text{break\_sensor}, \text{power\_cut}\}$.

Firstly, the input model is handled by IMITATOR to produce the MTS according to Algorithm 1. In repeated experiments, this step took approximately $2s$, on average. The MTS is then additionally processed by IF with the help of the Z3 solver. Finally, the program IF follows Algorithm 2 to build the set of at-constraints under which a state labeled with crash is reachable. To this end, IF prepares a set of programs for SPATULA, which corresponds to calling the external procedure synthpmARCTL in Algorithm 2. The second part of the process took approximately $1s$, on average.

Timing parameter constraints	Actions
$K_1 = \{k_1, k_2, k_3, k_4, k_5\}$	$Acts_1 = \{\text{choose\_house}, \text{power\_cut}\}$
$K_2 = \{k_6, k_7\}$	$Acts_2 = \{\text{choose\_sensor}\}$
$K_3 = \{k_8, k_9\}$	$Acts_3 = \{\text{choose\_house}\}$
$K_4 = \{k_{10}, k_{11}, k_{12}, k_{13}, k_{14}\}$	$Acts_4 = \{\text{break\_sensor}, \text{choose\_sensor}\}$

Table I: Constraints for reachability of the crash location

$$\begin{aligned}
k_1 = {}& \text{p\_walking\_house} \geq \text{p\_far} \wedge \text{p\_walking\_sensor} > 0 \\
&\wedge \text{p\_lowering} \geq \text{p\_emergency\_lowering} \wedge \text{p\_very\_near} > 0 \\
&\wedge \text{p\_emergency\_lowering} \geq \text{p\_very\_near} \wedge \text{p\_far} > 0 \\
&\wedge \text{p\_very\_near} + \text{p\_emergency\_waiting} \\
&+ \text{p\_walking\_house} \geq \text{p\_far} + \text{p\_approaching} \\
&\wedge \text{p\_far} + \text{p\_approaching} \geq \text{p\_walking\_house} + \text{p\_very\_near} \\
&\wedge \text{p\_far} + \text{p\_waiting} \geq \text{p\_walking\_house}
\end{aligned}$$

Figure 7: Exemplary parameter constraint k_1

In Table I we collect the result of the synthesis for the reachability of the crash location. The left column of the table contains sets of indices of constraints on time parameters. Due to the space limits we present only exemplary constraint in Figure 7. The right column of the table contains sets of actions. For a given row $1 \leq i \leq 4$, a state labeled with crash can be reached under a given at-valuation at if it satisfies one of constraints from K_i and enables all the actions from $Acts_i$. The information presented in Table I is exhaustive, *i.e.* crash is reachable iff at satisfies:

$$ReachCrash := \bigvee_{i=1}^{4} \Big(\bigvee_{k \in K_i} k \wedge \bigwedge_{a \in Acts_i} a \Big). \tag{$\clubsuit$}$$

Hence the system is safe from the malicious attempts of the intruder iff the at-constraint $AvoidCrash := \neg ReachCrash$ is satisfied. Let $1 \leq i \leq 4$. With a slight notational abuse let us denote $\overline{K_i} := \bigwedge_{k \in K_i} \neg k$ and $\overline{Acts_i} := \neg \bigwedge_{a \in Acts_i} a$. Observe that $\overline{K_i}$ represents the set of parameter valuations that invalidate all of the constraints of K_i. We interpret $\overline{Acts_i}$ as all the subsets of $Cntr$ that do not subsume $Acts_i$. We can now write:

$$AvoidCrash := \bigwedge_{i=1}^{4} \big(\overline{K_i} \vee \overline{Acts_i}\big). \tag{$\spadesuit$}$$

For the sake of presentation and readability, we specialize the considered problem by assuming from now on the following values of the train and gate parameters: $\text{p\_far} = 3$, $\text{p\_approaching} = 2$, $\text{p\_very\_near} = 1$, $\text{p\_waiting} = 1$, $\text{p\_emergency\_waiting} = 1$, $\text{p\_lowering} = 2$, and $\text{p\_emergency\_lowering} = 1$. For each constraint k we denote by $\ddot{k}$ the result of substituting the above parameters with the appropriate values. Then, the constraints $\ddot{k}_1, \ldots, \ddot{k}_{14}$ can be presented as shown in Figure 8.

$$\begin{aligned}
&\ddot{k}_1 = \text{p\_walking\_sensor} > 0 \wedge 4 \geq \text{p\_walking\_house} \geq 3, \\
&\ddot{k}_2 = \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} = 3, \\
&\ddot{k}_3 = \text{p\_walking\_sensor} > 0 \wedge 5 \geq \text{p\_walking\_house} \geq 4, \\
&\ddot{k}_4 = \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} = 4, \\
&\ddot{k}_6 = \text{p\_walking\_house} > 0 \wedge \text{p\_walking\_sensor} \geq 5, \\
&\ddot{k}_8 = \text{p\_walking\_sensor} > 0 \wedge \text{p\_walking\_house} \geq 5, \\
&\ddot{k}_{10} = \text{p\_walking\_house} > 0 \wedge 4 \geq \text{p\_walking\_sensor} \geq 3, \\
&\ddot{k}_{12} = \text{p\_walking\_house} > 0 \wedge 5 \geq \text{p\_walking\_sensor} \geq 4, \\
&\ddot{k}_{14} = \text{p\_walking\_house} > 0 \wedge \text{p\_walking\_sensor} = 4, \\
&\ddot{k}_5 = \ddot{k}_3, \ddot{k}_7 = \ddot{k}_6, \ddot{k}_9 = \ddot{k}_8, \ddot{k}_{11} = \ddot{k}_{10}, \ddot{k}_{13} = \ddot{k}_{12},
\end{aligned}$$

Figure 8: Parameter constraints under partial substitution

In a natural way we define $\overline{\ddot{K}_i} := \bigwedge_{k \in K_i} \neg \ddot{k}$, for all $1 \leq i \leq 4$ (see Figure 9). Let us show how we can employ these constraints to analyse safety of the system under selected capabilities of the intruder. Assume that the intruder can gain an access to the gate house and cut the power supply, but (s)he is not aware of the sensor, or the sensor cannot be disabled. This corresponds to enabling $Acts_1 = \{\text{choose\_house}, \text{power\_cut}\}$ in the model. As this set subsumes $Acts_3$ (see Table I), by Condition ♠ we have $AvoidCrash = \overline{\ddot{K}_1} \wedge \overline{\ddot{K}_3} = \text{p\_walking\_house} \leq 3$.

D. Performance Evaluation

To the best knowledge of the authors, there is no other tool allowing for mixed action-parameter synthesis for APTA.

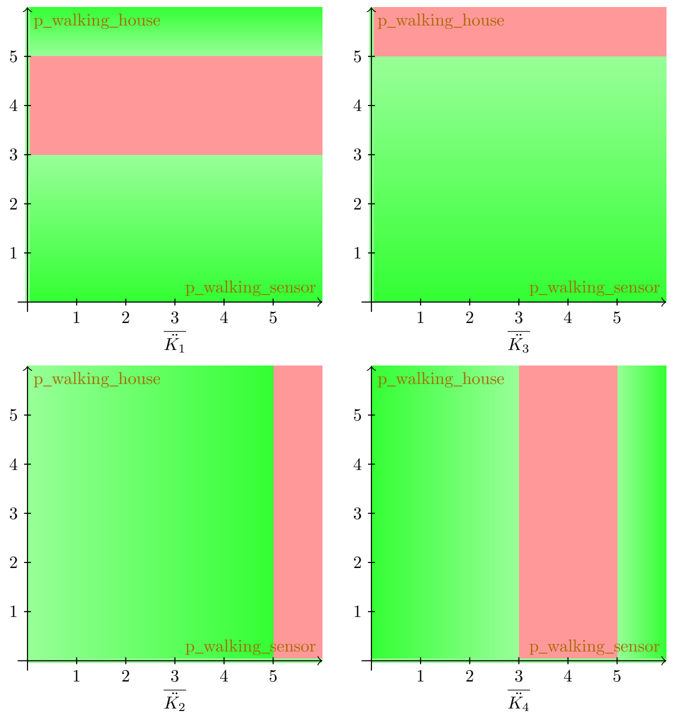


Figure 9: Safety regions

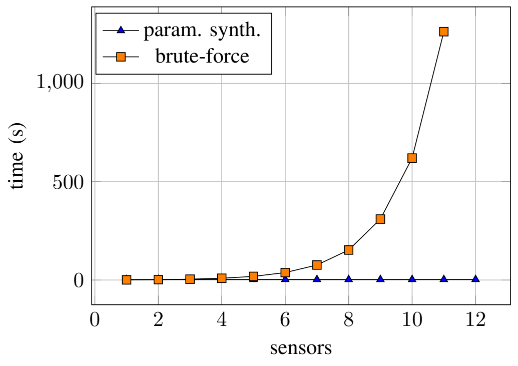


Figure 10: Parametric synthesis vs brute-force

Therefore, following the methodology presented in [KMP15], in Figure 10 we compare the efficiency of our approach to the brute-force solution, where IMITATOR is iteratively called on all the non-empty subsets of the controllable actions. We scale the benchmark with respect to the number of actions that allow to choose the sensor. The advantage of our approach is clearly visible and follows from the fact that SPATULA is tailored towards handling a large number of action variables. It should be noted, however, that our earlier experience with synthesis based on Binary Decision Diagrams (see [KMP15], [JKPL12]) allows us to hypothesise that the practical complexity of the solution presented in this paper is also exponential in the number of actions, albeit with a much smaller steepness than in the case of brute-force.

All the experiments have been performed on an Intel i5 quad-core 2.6 GHz machine with 4 GiB RAM, running Linux 3.13 operating system.

V. Conclusion and Perspectives

We presented an original framework to synthesise two kinds of parameters, *i.e.* action parameters and timing parameters, in a unified manner. The resulting constraint aims at helping designers to tune their system depending on cost or availability constraints, and at providing designers with tools supporting their decision making. We exemplified our approach on a proof-of-concept case study.

So far, we have considered only reachability-like properties. Although some properties go beyond the class of reachability properties, a quite large number of properties (including time properties such as deadlines) fall into the class of properties that can be encoded into reachability testing (see [ABBL98] for a study of its expressive power). Our longer-term goal is of course to generalize our results to deal with more complex, ideally full pmARCTL.

References

[ABBL98] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. The power of reachability testing for timed automata. In *FSTTCS*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 1998.

[Abd12] Parosh Aziz Abdulla. Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118, 2012.

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[ALR15] Étienne André, Didier Lime, and Olivier H. Roux. Integer-complete synthesis for bounded parametric timed automata. In *RP*, volume 9328 of *Lecture Notes in Computer Science*, pages 7–19. Springer, 2015.

[And15] Étienne André. What's decidable about parametric timed automata? In *FTSCS*, volume 596 of *Communications in Computer and Information Science*, pages 1–17. Springer, 2015.

[DKRT97] Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 1997.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[JKPL12] Andrew V. Jones, Michał Knapik, Wojciech Penczek, and Alessio Lomuscio. Group synthesis for parametric temporal-epistemic logic. In *AAMAS*. IFAAMAS, 2012.

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

[KMP15] Michał Knapik, Artur Męski, and Wojciech Penczek. Action synthesis for branching time logic: Theory and applications. *ACM Trans. on Embedded Comput. Syst.*, 14(4), 2015.

[Kna] Michał Knapik. michalknapik.github.io/spatula.

[KP12] Michał Knapik and Wojciech Penczek. Bounded model checking for parametric timed automata. *Transactions on Petri Nets and Other Models of Concurrency*, 5:141–159, 2012.

[PR06] C. Pecheur and F. Raimondi. Symbolic model checking of logics with actions. In *MoChArt*, volume 4428 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2006.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.