Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 1751 – 1764

doi:10.3906/elk-2007-170

Control synthesis for parametric timed automata under reachability

Ebru AYDIN GOL*
Department of Computer Engineering, Faculty of Engineering, Middle East Technical University, Ankara, Turkey

Received: 30.07.2020 • **Accepted/Published Online:** 20.01.2021 • **Final Version:** 31.05.2021

Abstract: Timed automata is a fundamental modeling formalism for real-time systems. During the design of such real-time systems, often the system information is incomplete, and design choices can vary. These uncertainties can be integrated to the model via parameters and labelled transitions. Then, the design can be completed by tuning the parameters and restricting the transitions via controller synthesis. These problems, namely parameter synthesis and controller synthesis, are studied separately in the literature. Herein, these are combined to generate an automaton satisfying the given specification by both parameter tuning and controller synthesis, thus exploring all design choices. First, it is shown that the negative decidability results derived for the parameter synthesis problem apply to the proposed problem. Then, a specific version of the problem is studied, where the specification is to reach a target set and parameters can take values from bounded integer sets. An algorithm based on depth first analysis combined with an iterative feasibility check is presented to solve the proposed problem. The correctness and the completeness (under mild assumptions) of the developed algorithm are proven. The findings of the paper are illustrated on an example drawn from scheduling.

Key words: Timed automata, decidability, control, parameter synthesis

1. Introduction

Designing real-time systems with correctness guarantees is a difficult process. Formal mathematical models, such as timed automata (TA), are developed for modeling and verification of real-time systems [1]. A timed automaton extends a finite automaton with a set of real-valued clock variables that measure the time. The clocks can be reset and tested. Thus the constraints over the time passed since the occurrence of an event of interest can be easily represented. Some of the examples of TA models of real-time systems are scheduling of real-time systems [2–4], medical devices [5, 6], and rail-road crossing systems [7].

The correctness of a timed automaton model against a specification can be verified via model checking. It is implemented in various off-the-shelve tools, such as UPPAAL [8], Imitator [9], HyTech [10], and it is applied on industrial case studies [8, 10, 11]. Model-checking can be performed once a complete TA model is obtained, and a negative verification result requires the designer to modify the final, possibly complicated, model. During the design phase, the system information can be incomplete and timing constants can be varied. In parametric timed automata (PTA) such uncertainties are modeled with parameters in place of the timing constants. For PTA, the design is completed via parameter synthesis: find a set of parameters such that the resulting model satisfies the specification. However, almost all nontrivial parameter synthesis problems are undecidable [12]. For example, for a parametric timed automaton, synthesis of parameters for reaching a set

*Correspondence: ebrugol@metu.edu.tr

of locations is undecidable even when the parameters are integer valued. The same problem becomes decidable when a finite upper bound is given for each parameter. Nevertheless, symbolic algorithms without termination guarantees exist for variations of the synthesis problem [12–14], including reachability specifications. While parametric models provide considerable flexibility in the design, the synthesis algorithms are computationally very expensive, for example, see [15] for benchmark examples.

Another approach that is orthogonal to parameter synthesis is controller synthesis, where possible design choices are integrated to the model via labels of the transitions and a control strategy restricts the transitions according to the labels [16–18]. In the pioneering work [16], the authors developed an iterative algorithm for solving synthesis problem for safety specifications (avoid "bad" states at all times). In addition, synthesis of optimal strategies considering location and transition weights has been considered. As summarized in a recent survey, corner point abstractions and game theoretic optimal control approaches are used to solve this problem under reachability specifications for deterministic and nondeterministic timed automata, respectively [17]. Both methods include the computationally expensive step of construction of a finite representation.

As mentioned above, the parameter and control synthesis problems are studied separately in the literature. Here, our goal is to tune the parameters and restrict transitions via controller synthesis such that the resulting automaton satisfies a specification; thus we combine both problems. A variation of parameter and controller synthesis problem is studied in [19], where a safety specification is considered, and the symbolic parameter synthesis method is extended to incorporate symbolic constraints over the transition labels. In [20], the authors present a parametric timed automaton model for an adaptive cruise control system. This model integrates the controller synthesis problem into the parameter synthesis problem via parametric mutually exclusive constraints.

In this paper, we formalize a control synthesis problem for parametric timed automata. We first show that the negative decidability results apply to the most general form of this problem. Then, we focus on a specific version of the problem where the goal is to reach a target set and the parameters are restricted to bounded integer sets. We show that the solution space is finite, thus the synthesis problem is, trivially, decidable. We propose an algorithm based on depth first search over the graph structure of timed automata. Central to the proposed method is the exploration along only realizable automata paths. In particular, a mixed integer linear programming (MILP) based feasibility check is performed for each candidate node (exploration direction), and only feasible nodes are added. Thus, this approach avoids computation of paths that cannot be part of the solution. The correctness of the proposed algorithm, as well as the completeness under mild assumptions are proven. In addition, the completeness for the general case (no additional assumption) is guaranteed with an additional computation step.

The paper is organized as follows. Section 2 presents the necessary notation and background information. Section 3 formally defines the control synthesis for parametric timed automata problem, derives decidability results and finally presents the proposed synthesis algorithm. Section 4 presents a case study inspired from scheduling problem, and illustrate the developed synthesis algorithm. Finally, Section 5 concludes the paper with possible future research directions.

2. Background

Notation The set of natural numbers, real numbers, nonnegative real numbers and positive real numbers are denoted by $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_{>0}$, respectively.

A timed automaton is a finite state machine extended with real-valued clock variables [1, 21, 22]. The constraints over these clocks govern the execution of the automata. For a set of clocks C, a clock constraint is defined with the following grammar $\phi := x \sim c \mid \phi \wedge \phi$ where $\sim \in \{<, >, \geq, \leq\}$, $c \in \mathbb{N}$ is a natural number and $x \in C$ is a clock. A constraint ϕ is called parametric if it contains a parameter in the place of the numeric constant c. A clock valuation is a function $\nu : C \to \mathbb{R}_{\geq 0}$ that assigns a nonnegative real value to each clock. A clock valuation ν satisfies a constraint ϕ, denoted by $\nu \models \phi$, if the constraint evaluates to true when each clock is replaced with the corresponding valuation. Two operations are used over the clock valuations: *delay* and *reset*. For a clock valuation ν and a positive constant $d \in \mathbb{R}_{>0}$ $\nu + d$ is the clock valuation obtained by incrementing each clock by d, $(\nu + d)(x) = \nu(x) + d$ for each $x \in C$. For a clock valuation ν, and a set of clocks $\lambda \subseteq C$, $\nu[\lambda]$ is the clock valuation obtained by resetting each clock from λ to 0, i.e. $\nu[\lambda](x) = 0$ for each $x \in \lambda$ and $\nu[\lambda](x) = \nu(x)$ for each $x \in C \setminus \lambda$.

Definition 1 ((Parametric) timed automata) *A timed automaton $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ is a tuple, where L is a finite set of locations, $l_0 \in L$ is the initial location, Σ is a finite input alphabet, C is a finite set of clocks, $\Delta \subseteq L \times \Sigma \times 2^C \times \Phi(C) \times L$ is a finite transition relation, and $Inv : L \to \Phi(C)$ is an invariant function.*

A transition $e = (l_s, \alpha, \lambda, \phi, l_t) \in \Delta$ is from location l_s to location l_t. The transition can be taken when the current input symbol is α and the clock valuation satisfies ϕ. Upon taking the transition, clocks from λ are reset to 0. A timed automaton is parametric if it contains a parametric constraint (either as an invariant or transition guard). For a parametric timed automaton $\mathcal{A}$, its set of parameters P, and a parameter valuation $v : P \to \mathbb{N}$ that assigns a number to each parameter, a (nonparametric, or concrete) timed automaton $\mathcal{A}(v)$ is obtained by replacing each parameter p with $v(p)$. A path is an interleaving sequence of locations and transitions $\pi = l_0 e_1 l_1 e_2 l_2 \ldots$ such that $e_i = (l_{i-1}, \alpha_i, \lambda_i, \phi_i, l_i) \in \Delta$ for each $i \geq 1$ and $l_i \in L$ for each $i \geq 0$.

A transition system is a tuple $T = (S, s_0, \Sigma, \rightarrow)$, where S is a set of states, $s_0 \in S$ is an initial state, Σ is a finite input alphabet and $\rightarrow \subset S \times \Sigma \times S$ is a transition relation. The notation $s \xrightarrow{a} s'$ is used for $(s, a, s') \in \rightarrow$. The semantics of timed automaton is defined as a transition system:

Definition 2 *Let $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ be a timed automaton. The semantics of $\mathcal{A}$ is defined by a transition system $T(\mathcal{A}) = (S, s_0, \Sigma', \rightarrow)$, where*

$S = \{(l, \nu) \mid l \in L, \nu \models Inv(l)\}$ is the set of states,
$s_0 = (l_0, \boldsymbol{0})$ is the initial state such that $\boldsymbol{0}(x) = 0$ for each $x \in C$,
$\Sigma' = \Sigma \cup \mathbb{R}_{\geq 0}$,
the transition relation is defined by the following rules

- *delay: $(l, \nu) \xrightarrow{d} (l, \nu + d)$ for $d \in \mathbb{R}_{>0}$ if $\nu + d \models Inv(l)$*
- *discrete: $(l, \nu) \xrightarrow{a} (l', \nu')$ if there exists $(l, a, \lambda, \phi, l') \in \Delta$ such that $\nu \models \phi$, $\nu' = \nu[\lambda]$ and $\nu' \models Inv(l')$*

We denote a delay transition of duration d followed by a discrete transition under input a by $(l, \nu) \xrightarrow{a}_d (l', \nu')$ (i.e $\exists \nu'' : (l, \nu) \xrightarrow{d} (l, \nu'') \xrightarrow{a} (l', \nu')$) . A run ρ of $T(\mathcal{A})$ is a possibly infinite alternating sequence of

states, delay and discrete transitions originating from s_0:

$$\rho := (l_0, \mathbf{0}) \xrightarrow{a_1}_{d_1} (l_1, \nu_1) \xrightarrow{a_2}_{d_2} (l_2, \nu_2) \xrightarrow{a_2}_{d_2} \dots$$

The set of all runs of $T(\mathcal{A})$ is denoted by $[[\mathcal{A}]]$. A path $\pi = l_0 e_1 l_1 e_2 l_2 \dots$ with $e_i = (l_{i-1}, \alpha_i, \lambda_i, \phi_i, l_i)$ for each $i \geq 1$ is said to be realized by a delay sequence $\mathbf{d} = d_1, d_2, \dots$ if there exists a run $\rho \in [[\mathcal{A}]]$ induced by π and $\mathbf{d}$, i.e., i-th location of ρ is l_i, and i-th transition is taken according to delay d_i and transition e_i. A path π is said to be *realizable*, if there exists a delay sequence $\mathbf{d}$ such that π and $\mathbf{d}$ induce a run ρ of $T(\mathcal{A})$.

Definition 3 (Control strategy) *A control strategy $\mathcal{C} : L \to \Sigma$ of a timed automaton $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ generates an input symbol for each location such that $\mathcal{C}(l) \in \Sigma(l) = \{\alpha \mid (l, \alpha, \lambda, \phi, l') \in \Delta\}$. The timed automaton obtained by executing $\mathcal{A}$ in closed loop with $\mathcal{C}$ is defined as $\mathcal{C}(\mathcal{A}) = (L, l_0, \Sigma, C, \mathcal{C}(\Delta), Inv)$ where $\mathcal{C}(\Delta) = \{e \mid e = (l, \alpha, \lambda, \phi, l') \in \Delta$ and $\alpha = \mathcal{C}(l)\}$.*

Intuitively, the input of the TA $\mathcal{A}$ is determined with respect to the strategy $\mathcal{C}$ for each discrete transition when $\mathcal{A}$ is run in closed loop with $\mathcal{C}$. Note that the control strategy simply restricts the transitions of $\mathcal{A}$.

The control synthesis and parameter synthesis problems are studied against reachability, unavoidability, safety properties, and more complex properties expressed in temporal logics such as computation tree logic (CTL) and metric interval temporal logic (MITL). In this paper, synthesis for reachability properties is studied, and the related definitions are given below. This type of properties are commonly used in scheduling problems [2–4]. In addition, the paper presents some decidability results for MITL and CTL. The interested readers are referred to [23] for more information on temporal logics.

Reachability: For a timed automaton $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$, a subset of its states $L^T \subset L$ is called *reachable* if there exists $\rho = (l_0, \mathbf{0}) \xrightarrow{a_1}_{d_1} (l_1, \nu_1) \xrightarrow{a_2}_{d_2} (l_2, \nu_2) \xrightarrow{a_2}_{d_2} \dots \in [[\mathcal{A}]]$ such that $l_i \in L^T$ for some $i \geq 0$.

3. Results

In this section, we first formally define the control synthesis for parametric timed automata problem. Then in the first subsection, we derive decidability results considering restrictions on parameter ranges and specifications. In the second subsection, for a restricted version of the problem that is shown to be decidable, we present a synthesis algorithm and prove its correctness.

Problem 3.1 *Given a parametric timed automaton $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$, its set of parameters P, an interval I_p for each parameter $p \in P$, and a property ψ,*

*a) **[decision]** Is there a control strategy $\mathcal{C}$ and a parameter valuation v pair such that $\mathcal{C}(\mathcal{A}(v))$ satisfies ψ?*

*b)**[synthesis]** Generate a control strategy $\mathcal{C}$ and a parameter valuation v such that $\mathcal{C}(\mathcal{A}(v))$ satisfies ψ if one exists.*

In literature, the parameter and control synthesis problems are studied separately. The parameter synthesis problem is studied from several aspects: safety, reachability, temporal logic formulas are considered as the specification (ψ), intervals in real numbers, intervals in integers and bounded intervals are considered for parameter intervals (I_p). In addition the restricted versions of the problem with respect to the number of parametric clocks and parametric constraints are considered in terms of decidability. The next section analyses Problem 3.1-a) in terms of decidability with respect to these results.

3.1. Decidability analysis for controller synthesis for parametric timed automata

A recent paper surveying the results on decision problems over parametric timed automata shows that almost all non-trivial decision problems are undecidable [12]. The negative decidability results are based on a reduction from halting problem of a two-counter machine that is known to be undecidable. On the other hand, decidability results are obtained when either the parameters are restricted to bounded integer sets or the number of parametric clocks and constraints are bounded. Problem 3.1 extends the classical parameter synthesis problem with the ability of restricting transitions via controller synthesis. We show that the negative decidability results applies to this problem as well. In particular, we first show that Problem 3.1-a) is undecidable for reachability.

Theorem 1 *The controller and parameter synthesis problem is undecidable for reachability.*

Proof Consider the class of PTA that has a single input, i.e. $\Sigma = \{\alpha\}$. For this class, the only feasible controller is $(C)(l) = \alpha$ for each $l \in L$. Consequently, the problem reduces to the parameter synthesis problem, which is known to be undecidable. If the controller and parameter synthesis problem was decidable, the result would apply to this sub-class. Thus, it is undecidable. □

The same argument from the proof of Theorem 1 applies to other properties such as safety and unavoidability for which the parameter synthesis problem is known to be undecidable. Thus, we conclude that Problem 3.1-a) is undecidable for reachability, safety and unavoidability properties.

The controller space is finite since both L and Σ are finite sets, and its size is upper bounded by $\mid \Sigma \mid^{|L|}$. Consequently, if the parameter synthesis problem is decidable, then the corresponding controller and parameter synthesis problem is also decidable, since it is sufficient to enumerate all possible control strategies and solve the parameter synthesis problem for each of them. For example, the parameter synthesis problem is decidable for the considered properties when each parameter is integer valued and restricted to a finite set. Thus, Problem 3.1-a) is decidable for these properties when $I_p = [l_p, u_p] \subset \mathbb{N}$ with a finite upper bound $u_p < \infty$ for each $p \in P$. In particular, the problem can be solved by enumerating each parameter valuation and controller synthesis pair and performing model-checking on the resulting TA. However, due to the exponential nature of the solution space and the model checking complexity, the greedy approach would be infeasible for any practical problem. In the subsequent section, we present an efficient algorithm for reachability property considering bounded and integer valued parameters.

3.2. Synthesis algorithm for reachability property

In this section, we present an algorithm to solve Problem 3.1 when $I_p = [l_p, u_p] \subset \mathbb{N}$ is a finite set of integers for each $p \in P$ and the specification ψ is $Reach(L_T)$ where $L_T \subset L$. First, we argue that the problem cannot be directly reduced to parameter synthesis under reachability via Example 1.

Example 1 *Consider the parametric timed automaton shown in Figure 1. It has four locations $L = \{l_0, l_1, l_2, l_3\}$ and four transitions $\Delta = \{e_1, e_2, e_3, e_4\}$. The parameter domains are $p_1 \in [7, 8]$ and $p_2 \in [2, 3]$, and the target location is l_3 ($L_T = \{l_3\}$). The feasible control inputs are $\Sigma(l_0) = \{a\}$, $\Sigma(l_1) = \{a, b\}$, and $\Sigma(l_2) = \{a\}$ until the target is reached; thus, no input is considered for l_3. There are two possible control strategies $\mathcal{C}_1$ and $\mathcal{C}_2$ that only differ at l_1, let $\mathcal{C}_1(l_1) = a$ and $\mathcal{C}_2(l_1) = b$. It is not possible to reach l_3 under strategy $\mathcal{C}_1$ since it eliminates the transition to l_3. On the other hand, the only path under strategy $\mathcal{C}_1$ is $l_0 e_1 l_1 e_4 l_3$. Along this path, the total time spent in l_0 and l_1 is upper bounded by 4 via invariants, and it is lower bounded by*

$p_1 \in [7,8]$ *via the guard on* e_4. *Thus the parameter synthesis problem is infeasible under this strategy. However, the classical parameter synthesis problem is feasible as it does not require assigning an input to each location. In particular,* l_3 *is reachable along the path* $l_0e_1l_1e_2l_2e_3l_1e_4l_3$ *for the parameter valuation* $p_1 = 7$ *and* $p_2 = 3$. *Note that path* $l_0e_1l_1e_2l_2e_3l_1e_4l_3$ *cannot be taken under any strategy.*

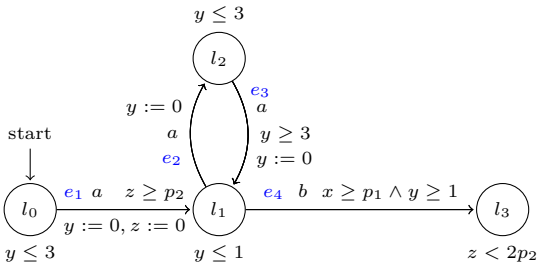

Figure 1. Timed automaton from Ex. 1. Transition labels are shown in blue. The control input, reset and guards are shown next to transitions. For example $e_1 = (l_0, a, \{y, z\}, z \geq p_2, l_1)$.

The proposed synthesis method to find a control strategy $\mathcal{C}$ and parameter valuation v for reaching L_T is summarized in Algorithm 1. The algorithm explores the possible paths and constructs an exploration tree in a depth first search manner. The nodes of the exploration tree correspond to the timed automaton locations, and child nodes are added with respect to the timed automaton transitions (see line 1 and line 19). First, the initial location of $\mathcal{A}$ is set as the root of the tree (see line 1). Thus, a path from root to a node of this exploration tree corresponds to a path of $\mathcal{A}$. The child nodes are added to the exploration tree as the possible paths are explored (stored in *node.children*), and control assignments are stored along the paths (the same control is used when a location appears more than once). Furthermore, the algorithm performs feasibility analysis for each new node and only adds nodes that are feasible, thus automaton paths obtained from the exploration tree are always realizable (see Defn. 2). Consequently, if a target node is reached, no further analysis is required and the algorithm returns the stored control assignments and parameter valuations obtained via feasibility analysis (see line 15). Next, the details of the steps of the algorithm and the feasibility computation are explained.

As in the classical depth first search implementation, the nodes to be explored are stored in a stack. In the proposed synthesis algorithm, in addition to storing such nodes, the stack is also used for marking the validity ranges of the previously selected control actions. In particular, a stack entry is in the following form $[node, explore]$ and there are 3 cases for $explore$: 1) it is $\perp$, 2) it contains a previously stored control for $node.l$, or 3) it contains the control choices to be explored from $node.l$. In the first case, it marks the validity range for the control choice stored in $\mathcal{C}(node.l)$. In the second case, along the path from *root* to *node*, location $node.l$ is previously visited and a control value is already set for it, thus exploration for the possible control inputs (lines 7-11) is not performed. Finally, in the last case, $explore \subseteq \Sigma(node.l)$ and a control input from $explore$ is set for $node.l$ (line 8), then, first, the rest of the control choices for the node are pushed back (line 9) for later exploration, second $[node.l, \perp]$ is pushed to mark the validity of the control choice. In particular, extracting this entry back from the stack (line 6) means that all possible paths from $node.l$ constrained to the inputs assigned from *root* to *node* (stored in $\mathcal{C}$) are already considered, and L_T is not reachable. Thus, the control assignment for $node.l$ is removed and the exploration continues with another node stored in the stack (line 4).

In the inner loop, each location l' that can be reached from $node.l$ under the control input $\mathcal{C}(node.l)$

Algorithm 1 Controller synthesis-reachability

Require: A TA $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$, parameter set P, parameter intervals I_p for each $\in P$, target set $L_T \subset L$, a bound on the number of cycles along a path *limit*.
Ensure: Control strategy $\mathcal{C}$ and parameter valuation v such that L_T is reachable on $\mathcal{C}(\mathcal{A}(v))$.

1: $root = Node(l \leftarrow l_0, children \leftarrow \emptyset)$.
2: $\mathcal{C}(l) = \perp$ for each $l \in L$, $Stack = \emptyset$.
3: $Stack.push([root, \Sigma(l_0)])$
4: **while** $Stack$ is not empty **do**
5: $node, explore = Stack.pop$
6: **if** $explore = \perp$ **then** $\mathcal{C}(node.l) = \perp$, Continue to line 4. ▷ A backtracking point, delete the corresponding input from $\mathcal{C}$ and continue with stack.
7: **if** $IsExploreSet(explore)$ **then**
8: $\mathcal{C}(node.l) = explore.pop()$ ▷ Assign a control input for l.
9: **if** $explore \neq \emptyset$ **then**Stack.push([node, explore]) ▷ Push back for the remaining control choices.
10: $Stack.push([Node(l \leftarrow node.l), \perp])$ ▷ Push a backtracking point for the control choice.
11: **end if**
12: **for each** $(l', e') \in \{(l, e) \mid e = (node.l, \mathcal{C}(node.l), \lambda, \phi, l) \in \Delta\}$ **do**
13: $v = IsFeasible(root - to - l')$ ▷ Find parameters that makes the path to l' realizable.
14: **if** $v = \perp$ **then** Continue to line 12. ▷ If no parameters exists, continue with the next location.
15: **if** $l' \in L_T$ **then Return** $\mathcal{C}$, v ▷ A solution is found.
16: **if** $CycleCount(root - to - l') > limit$ **then** Continue to line 12. ▷ Do not continue exploring.
17: **if** $\mathcal{C}(l') = \perp$ **then** $explore' = \Sigma(l')$ **else** $explore' = \mathcal{C}(node.l)$
18: $node' = Node(l \leftarrow l', children \leftarrow \emptyset)$ ▷ Create a new node.
19: $node.children.push(node', e')$ ▷ Add the node and e' for path generation.
20: $Stack.push([node', explore'])$
21: **end for**
22: **end while**
23: **return** No solution

is explored (line 12-21). First, a feasibility check is performed on the timed automaton path induced by the exploration tree path from *root* to l'. This check returns a parameter valuation v such that the path is realizable on $\mathcal{A}(v)$ if such a valuation exists, otherwise it returns $\perp$. The details of this method is given in the next subsection. If the path cannot be realized by any parameter valuation, exploration along l' is stopped (line 14). If the path is feasible, then it is checked whether l' is a target region. If this is the case, it is concluded that there is a realizable path from l_0 to l' in $\mathcal{C}(\mathcal{A}(v))$, and the strategy $\mathcal{C}$ and valuation v are returned (line 15). Otherwise, a cycle check is performed. If the number of cycles including l' is greater than a predefined value, the exploration along l' is stopped (line 16). If the cycle limit is not reached, a new node for l' is constructed (lines 18 and 19), and added to tree and stack for further exploration (line 20). If l' is already visited along the path from *root* to $node'$, the same control input is set to $\mathcal{C}(l')$, thus the control choices will not be considered for l' along the branches from $node'$ (line 7) and the control input for location l' along the path from root to this node will be consistent. However, if a control input is not assigned for l', all possible choices are pushed to stack for further exploration (line 17).

Complexity. The complexity of Algorithm 1 is characterized by the number of locations $|L|$, branching factor b of the underlying graph structure, i.e., $b = \max_{l \in L} |\{e \mid e = (l, \alpha, \delta, \phi, l') \in \Delta\}|$, and the cycle bound *limit*. In particular, the size of the resulting exploration tree is upper bounded by $b^{|L| \cdot limit}$, where $|L| \cdot limit$ is an upper bound on the tree depth. The number of feasibility analysis, i.e. MILP solutions (2), is also upper

bounded by $b^{|L|\cdot limit}$. However, thanks to the pruning of the infeasible directions, in practice, the number of the solved MILP problems is significantly less. In addition, the MILP size (the number of decision variables and the constraints) is linear with the length of the corresponding automaton path. Thus, while the number of decision variables is upper bounded by $b^{|L|\cdot limit} + |P|$, this bound is only reached along the longest path.

3.2.1. Feasibility check

We present a mixed integer linear programming based method for feasibility check. For a given parametric timed automaton $\mathcal{A}$, its path $\pi = l_0 e_1 l_2 e_2 \dots e_n l_n$ with $e_i = (l_{i-1}, \alpha_i, \lambda_i, \phi_i, l_i)$ for each $i \geq 1$, we find a valuation v such that π is realizable on $\mathcal{A}(v)$ (if such a valuation exists). Here, we define a mixed integer linear program such that its feasible solution defines a parameter valuation $v^\star$ and delay sequence $\mathbf{d}^\star = d_0^\star, d_1^\star, \dots, d_{n-1}^\star$ such that π and $\mathbf{d}^\star$ induce a run $\rho \in \mathcal{A}(v^\star)$. Essentially, the delay variables $d_0, d_1, \dots, d_{n-1}$, and parameters $p \in P$ are the decision variables of the MILP. For a given path, and a constraint along the path (either on a transition or an invariant), the clock can be represented in terms of the delay variables as it measures the time passed since its last reset. As time can only pass on a location, when leaving a location, a clock equals to the sum of the delay variables that correspond to the locations since the clock's last reset. In order to formalize this notion, we define the following mapping:

$$\Gamma(x, \pi, i) = d_k + d_{k+1} + \ldots + d_{i-1} \text{ where } k = \max(\{m \mid x \in \lambda_m, m < i\} \cup \{0\}), \tag{1}$$

where k is the index of the transition where x is last reset before e_i along π, and it is 0 if it is not reset. $\Gamma(0, \pi, i)$ is defined as 0 for notational convenience. The clock x equals to $\Gamma(x, \pi, i)$ on the i-th transition e_i along π.

Recall that a clock constraint is conjunction of clock inequalities $x \sim c$, where c is either a parameter $p \in P$ or a constant from $\mathbb{N}$ and $\sim \in \{<, \leq, >, \geq\}$. An inequality $x \sim c$ is mapped to the new delay variables with respect to its position. If it is on the guard ϕ_i of transition e_i, it is mapped to $\Gamma(x, \pi, i) \sim c$. If it is on the invariant $Inv(l_i)$ of location l_i, it should be satisfied when arriving to (i.e. for lower bounds) and leaving from (i.e. for upper bounds) the location. Thus it is mapped to $\Gamma(x, \pi, i+1) \sim c$ for leaving, and mapped to $\Gamma(x, \pi, i).\mathbf{I}(x \notin \lambda_i) \sim c$ for arriving, where $\mathbf{I}$ is a binary function mapping $true$ to 1 and $false$ to 0. Finally, the MILP for the path π is defined as:

$$\text{find } v_p \in \mathbb{N} \text{ for each } p \in P \text{ and } d_i \in \mathbb{R} \text{ for each } i = 0, \dots, n-1 \tag{2}$$

$$\text{subject to} \tag{3}$$

$$\Gamma(x, \pi, i) \sim c \qquad \text{for each } i = 1, \dots, n-1, \text{ and for each } x \sim c \text{ from } \phi_i \tag{4}$$

$$\Gamma(x, \pi, i).\mathbf{I}(x \notin \lambda_i) \sim c \qquad \text{for each } i = 1, \dots, n, \text{ and for each } x \sim c \text{ from } Inv(l_i) \tag{5}$$

$$\Gamma(x, \pi, i+1) \sim c \qquad \text{for each } i = 0, \dots, n-1, \text{ and } x \sim c \in \text{ from } Inv(l_i) \tag{6}$$

$$l_p \leq v_p \leq u_p \qquad \text{for each } p \in P, \text{ where } I_p = [l_p, u_p] \tag{7}$$

$$d_i \geq 0 \qquad \text{for each } i = 0, \dots, n-1 \tag{8}$$

Proposition 1 *Let $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ be a parametric timed automaton with parameter set P, and parameter range I_p for each $p \in P$ and π be a path of $\mathcal{A}$. Then the MILP as defined in (2) is feasible if and only if there exists a parameter valuation v such that π is realizable on $\mathcal{A}(v)$.*

Proof Let π be $l_0e_1l_1e_2\dots e_nl_n$ with $e_i = (l_{i-1}, \alpha_i, \lambda_i, \phi_i, l_i)$. (If) Assume that MILP (2) is feasible and let $v_p^\star$ for each $p \in P$ and delay sequence $\mathbf{d}^\star = d_0^\star, d_1^\star, \dots, d_{n-1}^\star$ be a solution, and $T(\mathcal{A}(v^\star)) = (S, s_0, \Sigma', \rightarrow)$ be defined as in Defn. 2. Define clock value sequence ν_0, ν_1, ..., ν_{n-1} with respect to the delay sequence and transitions $e_1, \dots, e_n$ iteratively as $\nu_0 = \mathbf{0}$ and :

$$\nu_i = (\nu_{i-1} + d_{i-1}^\star)[\lambda_i] \qquad \text{for } i = 1, \dots, n.$$

Observe that the ν_i definition is consistent with $\Gamma(\cdot, \pi, i)$ (1) along the path π and

$$a: \ \nu_i(x) = \Gamma(x, \pi, i)\mathbf{I}(x \notin \lambda_i) \quad \text{and} \quad b: \ \nu_i(x) + d_i^\star = \Gamma(x, \pi, i+1) \tag{9}$$

for each clock $x \in C$. For illustration of non-parametric $(x \sim c)$ and parametric $(y \sim p)$ inequalities let $Inv(l_i) = x \sim c \wedge y \sim p \wedge \phi'$ for an arbitrary clock constraint ϕ'. Then $\nu_i(x) \sim c \wedge \nu_i(y) \sim v_p^\star$ holds via (5) and (9)-a. As the same argument holds for each inequality from ϕ', we reach that $\nu_i \models Inv(l_i)$, thus $(l_i, \nu_i) \in S$ for each i by Defn. 2. By applying the same argument on the inequalities over (6) and (9)-b, we reach that $\nu_i + d_i^\star \models Inv(l_i)$, thus $(l_i, \nu_i) \xrightarrow{d_i^\star} (l_i, \nu_i + d_i^\star)$ (delay transition). Furthermore, from (4) and (9)-b, we have that $\nu_i + d_i^\star \models \phi_{i+1}$, thus $(l_i, \nu_i + d_i^\star) \xrightarrow{\alpha_i} (l_{i+1}, \nu_{i+1})$. Observing that $s_0 = (l_0, \mathbf{0}) \in S$, and the above derivation applies to each $i = 1, \dots, n$, we conclude that $\rho = (l_0, \nu_0) \xrightarrow{a_1}_{d_0^\star} (l_1, \nu_1) \xrightarrow{a_2}_{d_1^\star} \dots (l_n, \nu_n) \in [[\mathcal{A}(v^\star)]]$. (Only if) Assume that MILP (2) is infeasible, but there exists a parameter valuation v' such that π is realizable on $\mathcal{A}(v')$ via a delay sequence $d_0', \dots, d_{n-1}'$. Then (4),(5), and (6) holds for $d_0', \dots, d_{n-1}'$ and v' along the path π via Defn. 2. Thus, $d_0', \dots, d_{n-1}'$ and v' is a feasible solution of MILP, thus we reached a contradiction. □

In line 13 of Algorithm 1, the feasibility check is performed on the timed automaton path induced by the exploration tree path from *root* to *node*, and the location l'. The tree path and the timed automaton path are defined in (10) and (11), respectively.

$$node_0, \dots, node_{n-1} \quad \text{where } node_0 = root, node_{n-1} = node \text{ from line 5, and } (node_i, e_i) \in node_{i-1}.children \text{ for } i = 1, \dots, n-1, \tag{10}$$

$$\pi = l_0e_1l_2e_2\dots e_nl_n \quad \text{where } l_0 = root.l, l_i = node_i.l, \text{ and } e_i \text{ as in (10) for } i = 1, \dots, n-1, \text{ finally } e_n = e', l_n = e' \text{ from line 12.} \tag{11}$$

The path π is uniquely defined due to the tree structure. The feasibility of this path is checked via (2). Note that the iterative path construction via depth first search ensures that MILP for $\pi' = l_0e_1l_2e_2\dots l_{n-1}$ is previously constructed and it is feasible. For π, the constraints regarding d_{n-1}, $Inv(I_{n-1})$, e_n and $Inv(l_n)$ are added to this one.

In [24], a linear programming based method was used to generate an optimal delay sequence for a weighted timed automaton. Here, the optimization problem is in MILP form (2) since both integer valued parameters and delay variables are synthesized. An MILP based encoding was used in [25] for non-parametric timed automata under reachability specifications, where the integer variables were used to encode possible automaton paths.

3.2.2. Analysis of the synthesized controller

In this section, we first show that if Algorithm 1 generates a control strategy $\mathcal{C}$ and parameter valuation v, then L_T is reachable on $\mathcal{C}(\mathcal{A}(v))$, thus the result is correct. Then, we analyze the completeness, i.e., does the

algorithm find a solution when one exists, regarding the pruning of the exploration with respect to the detected cycles (line 16) and identify the cases in which the solution is complete. Finally, we present an extension to Algorithm 1 to guarantee completeness for any timed automata.

Proposition 2 *Let $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ be a parametric timed automaton with parameter set P, and parameter range I_p for each $p \in P$, and $L_T \subset L$ be a set of its states. If Algorithm 1 generates control strategy $\mathcal{C}$ and parameter valuation v when run on $\mathcal{A}, I_p$ for each $p \in P$ and L_T, then L_T is reachable on $\mathcal{C}(\mathcal{A}(v))$.*

Proof Let $\mathcal{C}$ and v be the control strategy and parameter valuation pair returned by Algorithm 1 in line 15. Let $\pi = l_0 e_1 l_2 e_2 \dots e_n l_n$ be the corresponding path of $\mathcal{A}$ as defined in (11). Note that by line 15, $l_n \in L_T$. By Prop. 1, π is realizable on $\mathcal{A}(v)$, thus L_T is reachable on $\mathcal{A}(v)$. The backtracking mechanism in lines 6 and 10 and the depth first exploration ensures that $(node_i, \perp)$ is in the stack for each $node_i$ defined in (10). Thus, $\mathcal{C}(l_i) \in \Sigma$ is well-defined. Furthermore, e_i is defined with respect to $\mathcal{C}(l_i)$ in line 12. Consequently, π is a path of $\mathcal{C}(\mathcal{A})(v)$, which concludes the proof. □

Proposition 2 shows that the result obtained from Algorithm 1 is correct. Next, we prove that the algorithm is complete if it does not perform a pruning in line 16. In other words, when the answer for Problem 3.1-a) is yes, the algorithm might not be able to find a control strategy and parameter valuation pair due to the pruning of the exploration performed in line 16. However, pruning is necessary for a termination guarantee as illustrated in Example 2.

Example 2 *Consider the timed automaton from Figure 1. The path $l_0 e_1 l_1 (e_2 l_2 e_3 l_1 e_2 l_2 e_3 l_1)^n$ is feasible (line 13) for any $n \geq 0$. Thus, it is necessary to detect such cases to avoid an infinite computation loop. On the other hand, consider a variation of the TA on which the input on e_4 is also a. As discussed in Example 1, while $l_0 e_1 l_1 e_4 l_3$ is not realizable for any parameter valuation, $l_0 e_1 l_1 e_2 l_2 e_3 l_1 e_4 l_3$ is realizable. Thus, traversing such a cycle can enable a transition, thus avoiding all of the cycles is not viable.*

Proposition 3 *Let $\mathcal{A} = (L, l_0, \Sigma, C, \Delta, Inv)$ be a parametric timed automaton with parameter set P, and parameter range I_p for each $p \in P$ and $L_T \subset L$ be a set of its states. If Algorithm 1 does not generate a result without eliminating any node in line 16, then no strategy and valuation pair exists for the reachability problem.*

Proof Assume that there exists control strategy $\mathcal{C}$ and proper valuation v with $v_p \in I_p$ for each $p \in P$, such that a path $\pi = l_0 e_1 l_2 e_2 \dots e_n l_n$ of $\mathcal{C}(\mathcal{A}(v))$ is realizable and $l_n \in L_T$. As Algorithm 1 searches paths exhaustively in a depth first manner, the exploration along path π is stopped before reaching l_n. Let $node$ be the furthest one reached and retrieved from stack in line 8 along π and $\mathcal{C}$ and let $node.l = l_i$. In the inner loop, each location l' that can be reached from $node.l_i$ under the control input $\mathcal{C}(node.l_i)$ is explored (line 12-21). As π is a path of $\mathcal{C}(\mathcal{A})$, (l_{i+1}, e_{i+1}) is in the set defined in line 12. As l_i is assumed to be the furthest one, an exploration tree node is not defined for l_{i+1}, thus the execution is ended in line (a) 14,(b) 15, or(c) 16. Case (a) contradicts with the assumption that π is realizable on $\mathcal{C}(\mathcal{A}(v))$, since if it is realizable, then any prefix of this path is also realizable via (2) and Proposition 1. Both case (b) and (c) contradicts with the proposition statement. Thus, the initial assumption on the existence of such a path is wrong. □

We deduce from Proposition 3 that the algorithm is complete for acyclic timed automata. Furthermore, completeness can be guaranteed with a proper cycle limit. For example, if there is a deadline t to reach a target location (checked on the transitions ends in L_T), and then t can be used as the limit in line 16.

Guaranteeing completeness: The completeness for any parametric timed automaton with bounded integer valued parameters can be guaranteed with an additional computation step. The barrier for the completeness is the pruning step in line 16, where a prefix of a realizable path can be pruned due to the cycle limit. To eliminate this barrier, the idea is to mark the nodes in line 16 instead of only pruning them, and perform further analysis if a solution is not found. In particular, if the algorithm reaches line 23 (no solution found), for each marked node in line 16 1) find a set of parameter valuations for the path from root to the marked node and define the search space for controller, and 2) run a greedy analysis for each of these. Essentially, the partial path to *node* reduces the search space of the reachability problem. The first step is the computation of the reduced space and the second step is the search for a solution in this space via enumeration. In the first step, first, the set of valuations is computed by constructing the MILP (2) for the path from *root* to *node* and projecting the feasible solution space on to parameter variables. This computation results in a set of valuations $\mathcal{V} \subseteq I_{p^1} \times \ldots \times I_{p^m}$, $P = \{p^1, \ldots, p^m\}$ such that the MILP (2) is not feasible for a $v \in I_{p^1} \times \ldots \times I_{p^m} \setminus \mathcal{V}$ (a feasible solution is not eliminated). Thus, it reduces the parameter search space to $\mathcal{V}$. Next, the search space for the control strategy is defined with respect to the strategy $\mathcal{C}$ computed from *root* to *node*. The search is only performed for locations with $\mathcal{C}(l) = \perp$, since if $\mathcal{C}(l) = \alpha \in \Sigma$, a control assignment is already done for l on *root* to *node*. Finally, in the second step reachability analysis via a verification tool such as UPPAAL is performed on the resulting timed automaton for each parameter and control strategy pair. Note that the search space is potentially reduced significantly compared to the initial problem. As each possible strategy and valuation pair from the marked node is analyzed, performing these steps for each marked node guarantees completeness.

4. Case study

We show the results of Algorithm 1 on a timed automaton that models a scheduling problem. We first describe the scheduling problem, then introduce the timed automaton modeling it. In this scheduling scenario, there are two types of jobs namely J_1 and J_2 that should be processed in this order. J_1 can be processed by machine $M_{1,a}$ or machine $M_{1,b}$. Similarly, J_2 can be processed by $M_{2,a}$ or $M_{2,b}$. The initial setup time (idle time before using any machine or starting a job transfer process), non-idle time, the time spent during the job transfers between machines, and the machine use durations are bounded. In particular, machine $M_{1,a}$ cannot be used more than 10 time units. $M_{2,a}$ cannot complete J_2 in less than 6 time units. The non-idle time is upper bounded by 18. The total processing time is upper bounded by 20 (or 22 based on the schedule). $M_{2,b}$ cannot be started after the first 10 time units. In addition to these strict constraints there are some flexible constraints. $M_{2,a}$ cannot be used within the first $16, 17$ or 18 time units. The initial idle time and the job transfer task to $M_{1,a}$ cannot exceed $3, 4$ or 5. The idle time cannot be less than 4 or 5 if $M_{1,a}$ is directly utilized. If a job transfer task was employed before $M_{1,a}$, the sum of the idle time and the job transfer time should be more than 5, 6, 7 or 8. The time spent on J_2 should be less than the time spent on J_1. If $M_{2,a}$ is used, its upper bound can be in $[10, 14]$, however if $M_{2,b}$ is used, this bound can be in $[8, 16]$. The goal is to find values for the flexible constraints and a scheduling scenario that describes a sequence of events and the corresponding durations such that the scenario satisfies the constraints. To achieve this, we model this scheduling problem as a parametric timed automaton shown in Figure 2.

In TA shown in Figure 2, the initial location l_0 represents the initial setup (idle). l_1, l_4, l_2 and l_5 represent machines $M_{1,a}$, $M_{1,b}$, $M_{2,a}$, and $M_{2,b}$, respectively. l_7, l_8, and l_9 represent the job transfer tasks. Finally, l_3 and l_6 are the locations that can be reached once both jobs are complete, thus $L_T = \{l_3, l_6\}$. The

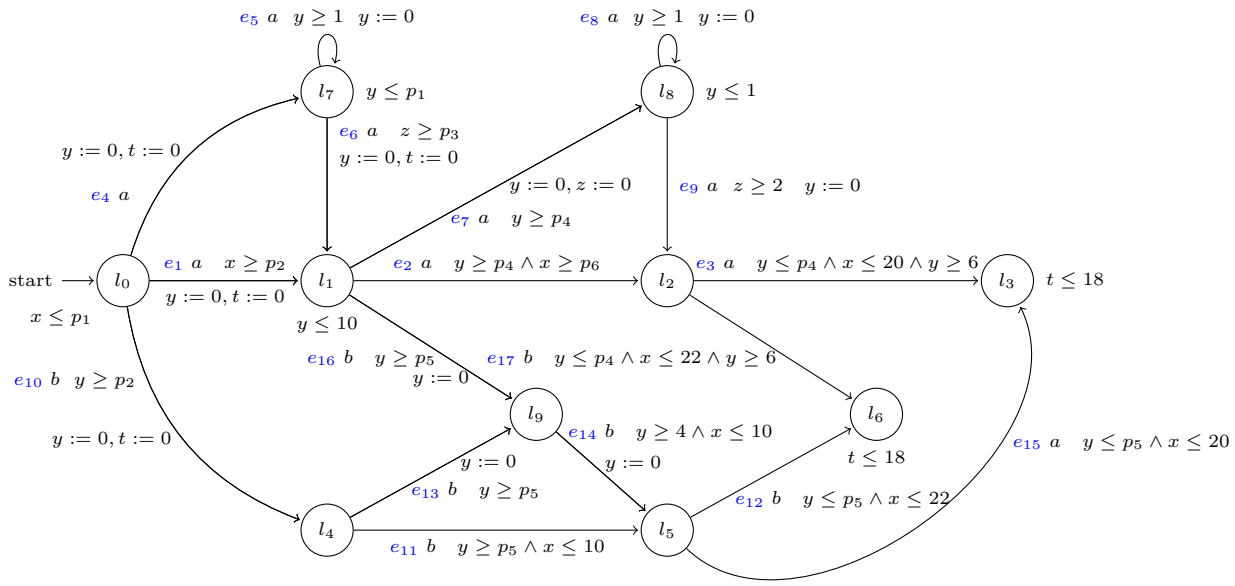

Figure 2. Timed automaton for the case study. Transition labels are shown in blue. The control input, reset and guards are shown next to transitions.

scheduling constraints are integrated to the timed automaton via clock constraints. Clock y is reset on each transition, thus it measures the time spent in the last location. For example, the constraint on the use of $M_{1,a}$ is encoded as $Inv(l_1) = y \leq 10$. The constraint that $M_{2,a}$ cannot complete J_2 in less than 6 time units is represented by the constraint $y \geq 6$ on the transitions leaving l_2. Clock x is not reset on any transition. Therefore, it represents the time passed since the beginning of the execution. It is used to describe the bounds on the total processing times (20 and 22). In addition, it is used to define the constraints relative to the initiation of the schedule. The constraint that $M_{2,b}$ cannot be started after the first 10 time units captured with the constraint $x \leq 10$ on the transitions that end in l_5. Clock t is reset only on transitions that leave l_0, thus it represents the non-idle time. The constraint on the non-idle time is enforced by $Inv(l_3) = Inv(l_6) = t \leq 18$. The flexible constraints are represented with parameters. The constraints on the duration of the idle time and the job transfer task to $M_{1,a}$ are captured with parameters $p_1 \in I_{p_1} = [3, 5]$, $p_2 \in I_{p_2} = [4, 5]$ and $p_3 \in I_{p_3} = [5, 8]$. The relative constraint on the time spend on J_1 and J_2 is captured with parameters $p_4 \in [10, 14]$ and $p_5 \in [8, 16]$ along the parametric constraints on the guards of the transitions that leave l_1, l_2, l_4 and l_5. The choice for the processing machines ($M_{1,a}$, $M_{1,b}$, $M_{2,a}$, $M_{2,b}$) are encoded as control inputs $\Sigma = \{a, b\}$.

The control strategy generated by Algorithm 1 is $\mathcal{C}(l_i) = a$ for $l_i \in \{l_0, l_1, l_7, l_8\}$ and $\mathcal{C}(l_i) = b$ for $l_i \in \{l_2, l_4, l_5, l_9\}$. The parameters are $p_1 = 5, p_2 = 4, p_3 = 5, p_4 = 10, p_5 = 8, p_6 = 16$. The path $\pi = l_0 e_4 l_7 e_6 l_1 e_2 l_2 e_{17} l_6 \in \mathcal{C}(\mathcal{A}(v))$ is realizable. The delay sequence $1, 5, 10, 6$ together with parameter valuations $p_1 = 5$, $p_3 = 5$, $p_4 = 10$ and $p_6 = 16$ is a solution of the MILP (2) defined for π. Thus, L_T is reachable on $\mathcal{C}(\mathcal{A}(v))$. Path π and the delay sequence specify the following schedule: stay idle for 1 time unit, then perform a job transfer to $M_{1,a}$ for 5 time units, use $M_{1,a}$ for 10 time units, and finally use $M_{2,a}$ for 5 time units.

In order to find all realizable paths (together with the corresponding control strategies and parameters) up to a given cycle count, instead of terminating the computation in line 15, the found strategy and parameter pair $\mathcal{C}, v$ is stored and the computation continued. For this case study, the modified version of the algorithm

generated 9 realizable paths and corresponding strategies. The paths are $\pi_n = l_0e_4l_7(e_5l_7)^ne_6l_1e_2l_2e_{17}l_6$ with $n = 0, 1, 2, 3, 4, 5, 6$ (7 paths) and $\pi_m = l_0e_1l_1e_7l_8(e_8l_8)^me_9l_2e_{17}l_6$ with $m = 1, 2$ (2 paths). Note that each π_n is generated by the same strategy, however the parameters differ. For each π_m, the strategy and the parameters are the same. Thus, the modified version of algorithm can be used to find all paths, and select a path together with the corresponding $\mathcal{C}, v$ according the an optimization criteria. The proposed methods are implemented as a python tool. The computation times for this example are 0.05 and 0.27 seconds on a laptop with $2.3Ghz$ quad core $i5$ processor for finding π and all realizable paths, respectively.

5. Conclusion

In this paper, we presented the controller synthesis for parametric timed automata problem to simultaneously tune parameters and restrict transitions. We proved that the negative decidability results apply to the most general form of this problem. Then, we focussed on a specific version of the problem: reachability with bounded integer parameters. We developed an algorithm to solve this problem and proved its correctness. In addition, we analyzed the algorithm in terms of completeness and identified the cases when the solution is complete. Furthermore, we presented an extension to guarantee completeness for any timed automaton. The results were shown on a nontrivial timed automaton modeling a scheduling scenario. The future research directions include considering other specifications such as safety and unavoidability.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 798482.

References

[1] Alur R, Dill D L. A theory of timed automata. Theoretical Computer Science 1994; 126 (2): 183–235.

[2] Fehnker A. Scheduling a steel plant with timed automata. In: Proceedings Sixth International Conference on Real-Time Computing Systems and Applications; Hong Kong, China; 1999. pp. 280-286.

[3] David A, Illum J, Larsen K G, Skou A. Model-based framework for schedulability analysis using UPPAAL 4.1. In: Model-based design for embedded systems. CRC Press, 2009, pp. 117-144.

[4] Guan N, Gu Z, Deng Q, Gao S, Yu G. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In: Software Technologies for Embedded and Ubiquitous Systems; Santorini Island, Greece; 2007. pp. 263-272.

[5] Kwiatkowska M, Mereacre A, Paoletti N, Patanè A. Synthesising robust and optimal parameters for cardiac pacemakers using symbolic and evolutionary computation techniques. In: Abate, A., Šafránek, D. (editors). Hybrid Systems Biology. Lecture Notes in Computer Science, vol 9271. Cham, Switzerland: Springer, 2015, pp. 119-140.

[6] Jiang Z, Pajic M, Alur R, Mangharam R. Closed-loop verification of medical devices with model abstraction and refinement. International Journal on Software Tools for Technology Transfer 2014; 16 (2): 191-213. doi.org/10.1007/s10009-013-0289-7

[7] Wang F. Formal verification of timed systems: a survey and perspective. Proceedings of the IEEE 2004; 92 (8): 1283–1305. 0.1109/JPROC.2004.831197

[8] Behrmann G, David A, Larsen K G, Hakansson J, Petterson P et al. Uppaal 4.0. In: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems; Washington, DC, USA; 2006. pp. 125-126.

[9] André É, Fribourg L, Kühne U, Soulat R. Imitator 2.5: A tool for analyzing robustness in scheduling problems. In: Formal Methods; Paris, France; 2012. pp. 33-36.

[10] Henzinger T A, Preussig J, Wong-Toi H. Some lessons from the hytech experience. In: Proceedings of the 40th IEEE Conference on Decision and Control; Orlando, FL, USA; 2001. pp. 2887–2892.

[11] André É, Fribourg L, Mota J-M, Soulats R. Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking. In: Verification, Model Checking, and Abstract Interpretation; Cascais, Portugal; 2019. pp. 409-424.

[12] André E. What's decidable about parametric timed automata? International Journal on Software Tools for Technology Transfer 2019; 21 (2): 203–219.

[13] Bezděk P, Beneš N, Barnat J, Černá I. Ltl parameter synthesis of parametric timed automata. In: Software Engineering and Formal Methods; Vienna, Austria; 2016. pp. 172-187.

[14] Jovanovic A, Lime D, Roux, O H. Integer parameter synthesis for real-time systems. IEEE Transactions on Software Engineering 2015; 41 (5): 445-461.

[15] André É. A benchmark library for parametric timed model checking. In: Formal Techniques for Safety-Critical Systems; Shenzhen, China; 2019. pp. 75-83.

[16] Asarin E, Maler O, Pnueli A, Sifakis J. Controller synthesis for timed automata. In: 5th IFAC Conference on System Structure and Control; Nantes, France; 1998. pp. 447-452.

[17] Bouyer P, Fahrenberg U, Larsen KG, Markey N, Ouaknine J et al. Model Checking Real-Time Systems, pp. 1001–1046. Cham, Switzerland: Springer International Publishing, 2018.

[18] Alur R, La Torre S, Pappas G J. Optimal paths in weighted timed automata. Theoretical Computer Science 2004; 318 (3): 297-322.

[19] Étienne A, Knapik M, Penczek W, Petrucci L. Controlling actions and time in parametric timed automata. In: 16th International Conference on Application of Concurrency to System Design; Torun, Poland ; 2016. pp. 45-54.

[20] Kara M Y, Gol E A. Adaptive cruise control with timed automata. In: 21th IFAC World Congress; Berlin, Germany (online), 2020. pp. 1-6.

[21] Alur R. Timed automata. In: International Conference on Computer Aided Verification; Trento Italy; 1999. pp.8-22.

[22] Larsen K G, Yi W. Time abstracted bisimulation: Implicit specifications and decidability. In: International Conference on Mathematical Foundations of Programming Semantics; New Orleans, LA, USA; 1993. pp. 160-176.

[23] Baier C, Katoen J-P, Larsen K G. Principles of Model Checking. Cambridge, MA, USA: The MIT Press, 2008.

[24] Bouyer P, Brihaye T, Bruyère V, Raskin J-F. On the optimal reachability problem of weighted timed automata. Formal Methods in System Design 2007; 31 (2): 135-175.

[25] Ober I. Revisiting bounded reachability analysis of timed automata based on milp. In: Formal Methods for Industrial Critical Systems; Maynooth, Ireland; 2018. pp. 269-283.