Synthesising Optimal Timing Delays for Timed I/O Automata

Marco Diciolla, Chang Hwan Peter Kim, Marta Kwiatkowska and Alexandru Mereacre
Department of Computer Science, University of Oxford, Oxford, UK
{marco.diciolla, peter.kim, marta.kwiatkowska, mereacre}@cs.ox.ac.uk

ABSTRACT

In many real-time embedded systems, the choice of values for the timing delays can crucially affect the safety or quantitative characteristics of their execution. We propose a parameter synthesis algorithm that finds optimal timing delays guaranteeing that the system satisfies a given quantitative property. As a modelling framework we consider networks of Timed Input/Output Automata (TIOA) with priorities and parametric guards. To express system properties we extend Metric Temporal Logic (MTL) with counting formulas. We implement the algorithm using constraint solving and Monte Carlo sampling, and demonstrate the feasibility of our approach on a simplified model of a pacemaker. We are able to synthesise timing delays that ensure with high probability that energy usage is minimised, while maintaining the basic safety property of the pacemaker.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Parametric Synthesis, Cardiac Pacemakers, Counting

1. INTRODUCTION

Model-based design of safety-critical real-time embedded systems, such as implantable medical devices and automotive airbag controllers, increasingly often relies on automated verification in order to establish that certain key requirements hold for the system model. In many cases, the choice of the timing delays can crucially affect the safety of the system or its quantitative characteristics such as energy consumption. Parametric timed automata [3], where parameters instead of constants can be used to specify such delays, have been proposed to bypass the need to perform the verification multiple times, for different constant delays. Instead, the parameter synthesis problem [10] asks whether there exists a parameter valuation which guarantees the satisfaction of the requirement. The parameter synthesis problem has been studied in different forms for timed automata models (see Related Work below), e.g. for reachability and branching-time logic specifications, but suffers from undecidability when parameters are real-valued. Most recent work has focused on identifying subclasses of models or parameter domains where the problem is tractable [18].

In this paper, we target embedded software modelled with parametric delays and develop algorithms to automatically synthesise optimal, robust values to guarantee the satisfaction of a range of quantitative properties. To this end, we extend the networks of timed I/O automata (TIOA) that communicate by matching input and output actions with parametric guards and priorities (the latter to determinise the system). As a property specification notation, we propose Counting Metric Temporal Logic (CMTL), an extension of the Metric Temporal Logic (MTL) with counting, which can express, e.g., constraints on the number of events occurring in an interval of time and the associated energy consumption. To address the potential undecidability, we work with finite path lengths and discretise the parameter space. Our main contribution is a solution to the *optimal parameter synthesis* problem for TIOA models with respect to a given CMTL formula and a quantitative objective function. We implement the algorithms in Python using constraint solving and Monte Carlo sampling.

We then demonstrate the usefulness of the techniques on an implantable cardiac pacemaker case study, which has been modelled in [15] using timed automata, where we automatically synthesise values for certain critical timing delays for the pacemaker. Counting is necessary in order to express the fundamental safety property of a pacemaker, i.e. that it maintains a regular rhythm of 60-120 heart beats per minute. Such a property cannot be expressed in MTL since in [22, 13] the authors show that MTL is unable to express counting. We derive a composition of the pacemaker model with the heart model and synthesise the time that the pacemaker waits before delivering a pace ($\mathsf{TLRI} - \mathsf{TAVI}$ according to Boston Scientific specification [1]). This value is critical to ensure the pacemaker safety (i.e. waiting too long can cause patient discomfort or even death), while at the same time it also affects the energy efficiency of the pacemaker (i.e. pacing too often will consume more energy). We were additionally able to confirm that the parameter values that our synthesis algorithm yields are in line with those recommended by pacemaker manufacturers.

Contributions.

ESWEEK'14 October 12 - 17 2014, New Delhi, India
Copyright 2014 ACM 978-1-4503-3052-7/14/10 ...$15.00
http://dx.doi.org/10.1145/2656045.2656073 .

The contributions of this paper can be summarised as follows:

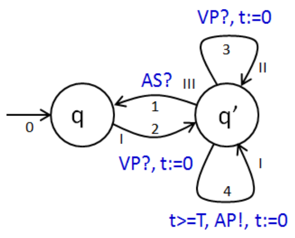
- We generalise the timed I/O automata model of [19] with priorities and parametric guards.
- We propose CMTL, an extension of the linear-time logic MTL with counting.
- We formulate a parameter synthesis algorithm which finds all parameter valuations such that, when instantiated, the network of TIOAs satisfies a given CMTL property. Instead of enumerating all possible parameter valuations, we generate symbolic constraints that satisfy the property, and then find an optimal, robust parameter valuation with respect to an objective function.
- We demonstrate the usefulness of the methods on a pacemaker case study.

Related work.
In [14], the authors study the decidability problem for parametric timed automata. They consider the special case of L/U automata for which they show that the emptiness problem is decidable. [4, 5] describe an approach to derive the constraints on parameters such that the behaviours of the timed automata are time-abstract equivalent, starting from a reference valuation rather than a logic formula. In [10], undecidability for parametric reachability problem is proved. The parameter synthesis problem for branching-time logic TCTL is studied in [8], where parameters are given both in the model and the formula. The authors show the decidability for a fragment of TCTL where equality is not allowed. In [20], the authors apply the bounded model checking procedure to solve the synthesis problem for the existential fragment of CTL without the next operator. In [7], the authors show PSPACE-completeness of the emptiness problem in parametric L/U timed automata for properties on infinite runs, while [18] consider the same class of automata for TCTL properties, also showing PSPACE-completeness. A parametric extension of timed I/O automata is given in [23], where it is shown how to construct an implementation of the specification that is robust under a given timed perturbation. In [17] the authors propose a method to synthesise optimal values of timing parameters for probabilistic timed automata given a reachability property.

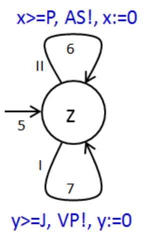
In this paper we present algorithms for parameter synthesis from specifications given in a generalisation of the linear-time logic MTL, rather than a branching-time logic or a reference valuation. Our results are akin to bounded model checking, since MTL formulas impose time bounds, and share similarities with the work of [4]. We also consider a generalisation of timed automata networks with priority, which are more expressive than L/U timed automata.

2. PROBLEM FORMULATION

Consider the TIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ in Fig. 1. The automata $\mathcal{A}_1$ and $\mathcal{A}_2$ form a network and they communicate with each other by means of actions $Act = \{\mathsf{VP}, \mathsf{AP}, \mathsf{AS}\}$. We distinguish input (marked with ?) and output actions (marked with !). For instance, when automaton $\mathcal{A}_2$ takes a transition



(a) $\mathcal{A}_1$



(b) $\mathcal{A}_2$

Figure 1: Example network $\mathcal{N}$ with two components.

and outputs the action VP!, the automaton $\mathcal{A}_1$ synchronises by taking the corresponding transition with the input action VP?. We use numbers $1, \ldots, 7$ to label the transitions and Roman numbers for the priorities with the lowest number denoting the highest priority. In the initial state (q, z), both automata start three clocks t, x and y. There are two ways to take a transition. First, when an input action is enabled. Second, when the clock satisfies a given condition (guard). For example, automaton $\mathcal{A}_2$ has two transitions labelled with the conditions $x \geq P$ and $y \geq J$, where P and J are parameters of the automaton. As soon as the clock y satisfies the guard $y \geq J$, the automaton takes the corresponding transition and outputs the action VP!, resetting to zero the value of the clock y ($y := 0$). When multiple transitions are enabled in a location, then the one with the highest priority will be taken.

Consider the finite path $\rho = (q, z)[2, 7](q', z)[4, -](q', z)$ of the network $\mathcal{N}$ from the initial state (q, z). Each parenthesised tuple represents a global state of the network, namely, states of each component automaton. Each bracketed tuple shows the transitions taken from the respective states of the tuple, with hyphen meaning that no transition was taken. In the automaton $\mathcal{A}_2$, y is initially 0 and, after J time units have passed, the guard $y \geq J$ becomes true and the corresponding transition (7) is triggered at this point, outputting the action VP and resetting the clock t to 0. The automaton $\mathcal{A}_1$ then synchronises with $\mathcal{A}_2$ via the matching input, VP, which moves the automaton to q' through transition 2. Then $\mathcal{A}_1$ takes transition 4 and $\mathcal{A}_2$ does not transition. Note that, for the automata to transition this way, the parameters must be constrained such that transition 4 triggers *before* transitions 1 and 3. When we instantiate all the parameters T, P and J in the network $\mathcal{N}$, we obtain a single timed path $\rho = (q, z) \xrightarrow{t_0} (q', z) \xrightarrow{t_1} (q, z) \xrightarrow{t_2} (q, z) \cdots$, $t_i \in \mathbb{R}_{\geq 0}, i \geqslant 0$, that describes the evolution of the network composed of $\mathcal{A}_1$ and $\mathcal{A}_2$. This is equivalent to saying that $\mathcal{N}$ is *deterministic*.

In this paper, we are interested in finding the values of parameters T, P and J such that the network $\mathcal{N}$ satisfies a given property. We consider properties expressed in counting metric temporal logic, which can count the number of actions in a given interval of time. For instance, the formula $\varphi = \#_5^7\mathsf{VP} \geqslant 1$ states that the number of VP actions in the interval of time $[5, 7]$ is greater than 1. There may be more than one set, possibly many sets, of parameter values that satisfy the set of linear inequalities. In practice, only the parameter values that are robust or that maximise a given objective function are likely to be of interest. To allow such interesting values to be found, we partition the set of parameters into controllable and uncontrollable. Then the objective function is used to choose the best value for a controllable parameter such that it maximizes a cost function over the uncontrollable parameter values.

We define the *optimal parameter synthesis problem* with respect to an objective function. We do not restrict to a single type of objective function, and instead admit a family of them, each of which will correspond to ensuring a particular quantitative property.

Optimal parameter synthesis problem
Input:
A parametric network of *Timed I/O Automata* (TIOAs) $\mathcal{N}$, a set of parameters $\Gamma = \Gamma_u \cup \Gamma_c$ composed of controllable (Γ_c) and uncontrollable (Γ_u) parameters, a *Counting Metric Temporal Logic* (CMTL) formula φ and a path length n.

Problem:
Find the optimal parameter values for Γ_c for any values of parameters Γ_u with respect to an objective function $\mathcal{O}$ such that φ is satisfied on paths of $\mathcal{N}$ of length n, if such values exist.

3. PARAMETRIC TIMED I/O AUTOMATA

In this section we introduce the modelling framework used in the paper. We adopt the timed I/O automata (TIOA) model defined in [19], which we augment with parametric guards and priority on the transitions in order to impose determinism. We remark that non-determinism is often viewed as an undesirable feature, since it could lead to dangerous behaviours of the system. For such a reason, we tailor our model to the specific domain in which we operate and exclude non-determinism by means of prioritised transitions.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of *nonnegative* real-valued variables, called *clocks*. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}_{\geq 0}$ assigning to each variable x a nonnegative real value $\eta(x)$. Let $\mathbf{0}$ denote the valuation that assigns 0 to all clocks. For a subset $X \subseteq \mathcal{X}$, the reset of X, denoted $\eta[X := 0]$, is the valuation η' such that $\forall x \in X.\ \eta'(x) := 0$ and $\forall x \notin X,\ \eta'(x) := \eta(x)$. For $\delta \in \mathbb{R}_{>0}$ and $\mathcal{X}$-valuation η, $\eta+\delta$ is the $\mathcal{X}$-valuation η'' such that $\forall x \in \mathcal{X}.\ \eta''(x) := \eta(x)+\delta$, which implies that all clocks proceed at the same speed.

Definition 1. A deterministic timed I/O automaton (TIOA) with priority $\mathcal{A} = (\mathcal{X}, \Gamma, Q, q_0, \Sigma_{\text{in}}, \Sigma_{\text{out}}, \rightarrow, \gamma)$ consists of:

- A finite set of clocks $\mathcal{X}$.
- A finite set of integer-valued parameters $\Gamma = \Gamma_c \cup \Gamma_u$, where Γ_c and Γ_u are respectively the set of controllable and uncontrollable parameters.
- A finite set of locations Q, with the initial location $q_0 \in Q$.
- A finite set of input actions Σ_{in} and a finite set of output actions Σ_{out}.
- A transition relation $\rightarrow \subseteq Q \times (\Sigma_{\text{in}} \cup \Sigma_{\text{out}}) \times \mathcal{B}(\mathcal{X}, \Gamma) \times 2^{\mathcal{X}} \times Q$, where $\mathcal{B}(\mathcal{X}, \Gamma)$ denotes the set transition guards over $\mathcal{X}$ and Γ. For any $q, q' \in Q$, $X \subseteq \mathcal{X}$, if $a \in \Sigma_{\text{out}}$ then $e = (q, a, g, X, q') \in \rightarrow$ has $g \neq \mathbf{true}$. Also, for any $q \in Q$ and any two outgoing transitions of q with guards $g_1, g_2 \neq \mathbf{true}$, it holds that $g_1 \cap g_2 = \varnothing$.
- A priority function $\gamma : Q \times (\Sigma_{\text{in}} \cup \Sigma_{\text{out}}) \to \mathbb{N}$ that assigns a priority to an action in a given location. For any $q \in Q$, $a_{\text{in}} \in \Sigma_{\text{in}}$, $a_{\text{out}} \in \Sigma_{\text{out}}$ and $a_1, a_2 \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$ we require $\gamma(q, a_{\text{in}}) < \gamma(q, a_{\text{out}})$ and $\gamma(q, a_1) \neq \gamma(q, a_2)$.

Let $e = (q, a, g, X, q')$ be a transition of TIOA $\mathcal{A}$ and η a clock valuation. We say that a transition with an action a is *enabled* if either $a \in \Sigma_{\text{in}}$ or $a \in \Sigma_{\text{out}}$ and $\eta \models g$. Here $\eta \models g$ means that the clock valuation η satisfies the clock constraints in g. Observe that every transition of the TIOA $\mathcal{A}$ that has an output action is *urgent*, i.e., it is taken as soon as the guard becomes true. The TIOA in the above definition can still exhibit Zeno behaviour, but one can use the sufficient criteria in ([6], Lem. 9.24) to check for Zenoness. From the above definition it is clear that the TIOA is deterministic, i.e., at every point in time only one transition can be taken. We impose the determinism by means of the priority function, as well as the fact that a transition labelled with an output action is urgent. For instance, if two transitions are enabled the one with the highest priority will be taken.

The TIOAs as defined above are able to synchronise on matching input and output actions, thus forming networks $\mathcal{N}$ of communicating automata. Informally, the network $\mathcal{N}$ evolves as follows. Each component $\mathcal{A}_i$ of $\mathcal{N}$ can either (i) have an output transition with maximum priority enabled, in which case the component fires the output transition and moves to the next location accordingly, or (ii) if no output transition is enabled then it either synchronises with an output transition fired by another component, which must have a matching input transition, or lets time pass. We define the *finite timed path* of a network $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1, \ldots, m\}\}$ of TIOAs $\mathcal{A}^{(i)}$ with the set of locations $Q^{(i)}$, $i \in \{1, \ldots, m\}$ as the sequence $\rho = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ such that $\vec{q_j} \in Q^{(1)} \times \cdots \times Q^{(m)}$ and $t_j \geqslant 0$ for all $j \in \{0, \ldots, n\}$. Thus, each path of the network $\mathcal{N}$ consists of m paths corresponding to the TIOAs $\mathcal{A}^{(i)}$ ($\vec{q_j}$ in ρ is a vector). Each transition from every component of $\vec{q_j}$ takes the same amount of time.

4. COUNTING MTL

In this section we define the Counting Metric Temporal Logic (CMTL). CMTL extends MTL with basic counting formulas (BCF), with which one can count the number of actions (events) in a given interval of time. We use the *pointwise semantics* for both MTL and BCF. We refer the reader to a survey of the differences between MTL and counting variants of MTL in [22, 13].

Given a finite timed path ρ, we define the set $Act_j = \Sigma_{\text{out},j} \cup \Sigma_{\text{in},j}$ of actions at step j corresponding to the set of transitions that are taken at step j. We write $\rho[j] := Act_j$ for ($j \leqslant n$) and $\rho\langle j \rangle := t_j$. Moreover, for $t \in \mathbb{R}_{\geqslant 0}$, $\rho@t := o$, where o is the smallest index such that $\sum_{k=0}^{o} \rho\langle k \rangle > t$. We define

$$\rho[\![j]\!] := \vec{q_j} \xrightarrow{t_j} \vec{q_{j+1}} \xrightarrow{t_{j+1}} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$$

to be the suffix of ρ starting at step j.

Definition 2. Let $\rho = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ be the finite timed path of the network $\mathcal{N}$ of TIOAs. The *counting function* $\#_\ell^u a$ for an action $a \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$ and time points $\ell \in \mathbb{R}_{\geqslant 0}$, $u \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$, such that $\ell < u$, is defined as $\#_\ell^u a = \sum_{k=(\rho@\ell)}^{(\rho@u)-1} (a \in \rho[k])$.

A *basic counting formula* (BCF) $\mathbb{B}$ is of the form

$$\mathbb{B} = \sum_{j \in J} c_j \#_{\ell_j}^{u_j} a_j, \quad \text{where } J \text{ is a finite index set,} \tag{1}$$

$c_j \in \mathbb{Z}_{>0}$, $\ell_j, u_j \in \mathbb{R}_{\geqslant 0}$ (with the usual constraint that $\ell_j < u_j$ for all j) and $a_j \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$.

We now define our logic CMTL as an extension of MTL, where we replace atomic propositions with BCF formulas.

Definition 3. The syntax of the *Counting Metric Temporal Logic* (CMTL) φ is defined inductively by

$$\varphi ::= \mathbb{B} \bowtie b \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathcal{U}^{[\ell,u]} \varphi,$$

where $\bowtie \in \{>, \geqslant, <, \leqslant\}$, $b \in \mathbb{Z}$, and $\ell \in \mathbb{R}_{\geqslant 0}$, $u \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$ are time points such that $\ell \leqslant u$.

The satisfaction relation for CMTL is defined over timed paths of the network $\mathcal{N}$ of TIOAs.

Definition 4. Let $\rho = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ be the finite timed path of the network $\mathcal{N}$ of TIOAs and $i \in \mathbb{N}$ be an index. We say that $\mathcal{N}$ *satisfies* φ at i, denoted $(\rho, i) \models^{\mathcal{N}} \varphi$, iff

$$(\rho, i) \models^{\mathcal{N}} \mathbb{B} \bowtie b \quad \text{iff } \sum_{j \in J} c_j \sum_{k=\rho[\![i]\!]@\ell_j}^{\rho[\![i]\!]@u_j - 1} (a_j \in \rho[k]) \bowtie b$$

$$(\rho, i) \models^{\mathcal{N}} \varphi_1 \wedge \varphi_2 \quad \text{iff } (\rho, i) \models^{\mathcal{N}} \varphi_1 \wedge (\rho, i) \models^{\mathcal{N}} \varphi_2$$

$$(\rho, i) \models^{\mathcal{N}} \neg\varphi_1 \quad \text{iff } (\rho, i) \not\models^{\mathcal{N}} \varphi_1$$

$$(\rho, i) \models^{\mathcal{N}} \varphi_1\, \mathcal{U}^{[\ell,u]} \varphi_2 \quad \text{iff } \exists i'.\, i \leqslant i' \text{ s.t. } \sum_{k=i}^{i'} \rho\langle k \rangle \in [\ell, u] \wedge (\rho, i') \models^{\mathcal{N}} \varphi_2 \wedge \forall i''.\, i \leqslant i'' < i' \wedge (\rho, i'') \models^{\mathcal{N}} \varphi_1,$$

where φ_1, φ_2 are CMTL formulas, and $i', i'' \in \mathbb{N}$.

We define $\Diamond^{[\ell,u]}\varphi := \mathbf{true}\, \mathcal{U}^{[\ell,u]}\varphi$ and $\Box^{[\ell,u]}\varphi := \neg\Diamond^{[\ell,u]}\neg\varphi$.

5. PARAMETER SYNTHESIS

In this section we describe the algorithm to find the values for the controllable parameters such that the instantiated network of TIOAs $\mathcal{N}$ satisfies a given CMTL property φ.

A naive solution to the problem is to enumerate all possible values $\bar{\Gamma}$ for the parameters Γ, under the assumption of a bounded integer parameter space, and for each value generate the *unique* path ρ in the network of TIOAs $\mathcal{N}$. Observe that $\bar{\Gamma}$ is finite. Next, we check the satisfaction of the property φ on path ρ, which results in a set of parameter values $\bar{\Gamma}' \subseteq \bar{\Gamma}$ such that each value in $\bar{\Gamma}'$ induces an instantiated network that satisfies the property φ. The best choice of the parameter value is the one that maximises the objective function. Note that, if m is the number of possible values for a parameter, then the size of $\bar{\Gamma}'$ is $m^{|\Gamma|}$. Given the exponential size of $\bar{\Gamma}'$, the problem of finding the best parameter values becomes infeasible in practice.

We instead propose an approach based on parameter sampling and constraints generation. First, we sample a sufficiently large number of values from the set $\bar{\Gamma}$. Second, for each sampled parameter value we generate the discrete path ρ in the instantiated network of TIOAs $\mathcal{N}$. Given the *untimed* path ρ, i.e., the discrete path obtained by eliding the time values, we generate a formula $\mathcal{S}$ on the parameter set Γ. Therefore, the formula $\mathcal{S}$ will correspond to all parameter values that generate the same untimed path ρ. From here on we use the notation $\vartheta \in \mathcal{S}$ to say that the parameter value ϑ, once plugged into the parameters of $\mathcal{S}$, makes the formula $\mathcal{S}$ true. We say that $\vartheta \notin \mathcal{S}$ otherwise. The advantage of the constraint generation approach is that, with fewer samples, we can cover more values from $\bar{\Gamma}$.

In a nutshell, the synthesis problem can be solved by first generating a formula $\mathcal{S}$ from property φ and path ρ of $\mathcal{N}$, described in Section 5.1, and then finding an optimal solution for $\mathcal{S}$ with respect to the given objective function $\mathcal{O}$, described in Section 5.2.

5.1 Constraint generation

We first describe the intuition for how to compute the formula $\mathcal{S}$ that guarantees the satisfaction of the property along the path ρ, and next present Algorithm 1 that generates $\mathcal{S}$.

The formula $\mathcal{S}$ is computed with the following simple steps:

1. Sample the domains of the model parameters in order to generate a discrete path.

2. For each sampled parameter value do:

   - If the value does not make the formula $\mathcal{S}$ true
     - Generate the untimed path ρ.
     - Generate the set of inequalities which satisfy φ in ρ (Algorithm 2 and 5).

Algorithm 1 generates the formula $\mathcal{S}$ with the help of two main subroutines, Algorithm 2 and Algorithm 5. We now describe Algorithm 1 and its subroutines step by step.

Algorithm 1 Constraint generation for $\mathcal{N}$ with m-components, CMTL formula φ and path length n

```
Require: Network $\mathcal{N}$, formula $\varphi$ and path length $n$
Ensure: Formula $\mathcal{S}$
 1: Function Sat($\mathcal{N}, \varphi, n$)
 2: $\bar{\Gamma}$ := Sample($\Gamma$)
 3: for $\vartheta \in \bar{\Gamma}$ do
 4:     if $\vartheta \notin \mathcal{S}$ then
 5:         $\rho$ := Gen_path($\mathcal{N}, n, \vartheta$)
 6:         $(\mathcal{S}_\rho, \mathcal{T})$ := Path_Constr_Gen($\mathcal{N}, \rho$)
 7:         $\mathcal{S}_\varphi$ := Constr_Gen($\rho, 0, \varphi, \mathcal{T}$)
 8:         $\mathcal{S} := \mathcal{S} \bigvee (\mathcal{S}_\rho \bigwedge \mathcal{S}_\varphi)$
 9:     end if
10: end for
11: return $\mathcal{S}$
```

The first step (line 2) of Algorithm 1 samples the domain of Γ obtaining the set of parameter values $\bar{\Gamma}$. The algorithm then iterates over each point of $\bar{\Gamma}$ and at every iteration checks whether the value under consideration, say ϑ, satisfies the formula $\mathcal{S}$ or not. This operation is indicated in Algorithm 1 with $\vartheta \notin \mathcal{S}$. The intuition behind this step is that multiple model parameters will satisfy the same formula $\mathcal{S}$. Thus, instead of generating the constraints for the discrete path ρ given by parameter value ϑ, we check whether $\vartheta \in \mathcal{S}$. If this is the case, we then skip this value and therefore save computation time. The second step of the algorithm generates a discrete path ρ from the parameter value ϑ. The algorithm is based on the semantics of the network of TIOAs (see technical report [9]). The function Gen_path returns the untimed discrete path ρ. Afterwards, Algorithm 1 generates constraints over the parameter set Γ from the discrete path ρ of length n. This is accomplished with Algorithm 2, which is composed of two function calls described in the next paragraph. The algorithm returns the formula $\mathcal{S}_\rho$ and the matrix of time constraints $\mathcal{T}$. The matrix $\mathcal{T}$ contains the time constraints t_j over the parameter set Γ corresponding to every discrete transition that is taken at step $j \in \{0, \dots, |\sigma| - 1\}$ of ρ. The formula $\mathcal{S}_\rho$ contains the relationship between time constraints $\mathcal{T}$, as well as the clock valuations η and guards g. For instance, if there is a transition labelled with a guard $x \leqslant \gamma$, where $x \in \mathcal{X}$ and $\gamma \in \Gamma$, then $\mathcal{S}_\rho$ will contain the constraint $\eta(x) \leqslant \gamma$. In this case $\eta(x)$ is an expression over the parameter set Γ.

The first function call of Algorithm 2 at line 5 (Algorithm 3) iterates over the set of transition with maximal priority $\mathcal{I}_j$ that are taken at step j and generates the symbolic time constraints $\mathcal{T}[j, i]$. It also generates the formula $\mathcal{S}_o$ relating the clock valuations and the guard bounds. For instance, given the transition $e^{(i)} := (q_j^{(i)}, a, g^{(i)}, X^{(i)}, q_{j+1}^{(i)})$, where a is an output action, $\mathcal{S}_o$ will contain the expression $\{\eta^{(i)}(x) \leqslant g^{(i)}.x(2)\}$ if $x \leq g^{(i)}.x(2)$ is a constraint in guard $g^{(i)}$. Here $g^{(i)}.x(1)$ denotes the sign for the clock x in guard $g^{(i)}$ and $g^{(i)}.x(2)$ denotes the bound of x. At the end of the cycle (line 14), the algorithm creates a new clock valuation η_{next} from the symbolic time constraint $\mathcal{T}[j, i]$. The for cycle at lines 7-11 in Algorithm 2 resets all the clocks that are associated with transitions that are taken at step j. The set of transitions that are taken and labeled with an output or input action is given by $\mathcal{I}_j \cup \mathcal{I}_j^s$. The last function call of Algorithm 2 at line 13 (Algorithm 4) generates the formula $\mathcal{S}_n$ for the remaining transitions $\mathcal{I}_j^c$ that are not taken. Finally, line 15 and 16 of Algorithm 2 adds to $\mathcal{S}$ the relationships between all time constrains, namely, $\mathcal{T}[j, i] = \mathcal{T}[j, k]$ for every taken transition that has an output action, and $\mathcal{T}[j, k'] > \mathcal{T}[j, i]$ for the remaining transitions that are not taken, where $i, k \in \mathcal{I}_j$ and $k' \in \mathcal{I}_j^c$.

Example 1. In Table 1 we show a sample execution of Algorithm 2 for the discrete path $\rho = (q, z)[2, 7](q', z)[4, -](q', z)$ (recall that parenthesised tuples represent states and bracketed tuples represent transitions) of the TIOA from Fig. 1, where j denotes the index of the path. At the beginning of the path, all three clocks, t, x, and y, are set to zero initially (shown by the clock valuations in column 0). $\mathcal{A}_2$ outputs VP and $\mathcal{A}_1$ synchronizes with it after J time units (time constraints in column 0). Transition 7 is the first transition taken by $\mathcal{A}_2$, and therefore it must occur before other possible transitions, namely, transition 6, meaning that $y \geq J$ must become true before $x \geq P$. Since both clocks started at 0, $J < P$ (S in column 0). Then $\mathcal{A}_1$ takes transition 4, which is fired when $t = T$. Since $t = J$ before the transition (clock valuation in column 1), the time taken to fire the transition is $T - J$ (time constraint in column 1). The time taken to fire transition 4, $T - J$, must be less than the time taken to fire transition 1, $P - J$ (since $x = J$ at the beginning of $j = 1$ and transition 1 synchronizes with transition 6 when $x = P$), and transition 3, $J - 0$ (analogously to transition 1). This is shown in the S row in column 1.

Algorithm 2 Constraints generation for the path ρ

```
Require: Network $\mathcal{N}$ and discrete path $\rho$
Ensure: Formula $\mathcal{S}$ and matrix $\mathcal{T}$ over $\Gamma$
 1: Function Path_Constr_Gen($\mathcal{N}, \rho$)
 2: $\eta := \mathbf{0}$, $\mathcal{S} := \varnothing$
 3: for $j := 0$ to $|\rho| - 1$ do
 4:     $\mathcal{I}_j$ - index of taken transitions that have an output action
 5:     $(\mathcal{S}_o, \mathcal{T}, \eta)$:=Sync_Constr($\mathcal{N}, j, \rho, \mathcal{I}_j, \mathcal{T}, \eta$) (Alg. 3).
 6:     $\mathcal{I}_j^s$ - index of taken transitions that have an input action
 7:     for $x \in \mathcal{X}$ do
 8:         if $x \in X^{(i)}$ for some $e^{(i)} := (q_j^{(i)}, a, g^{(i)}, X^{(i)}, q_{j+1}^{(i)})$ and $i \in \mathcal{I}_j \cup \mathcal{I}_j^s$ then
 9:             $\eta(x) := 0$ - reset all the clocks that are associated with a taken transition
10:         end if
11:     end for
12:     $\mathcal{I}_j^c$ - index of transitions that are not taken
13:     $\mathcal{T}$:=NSync_Constr($\mathcal{N}, j, \rho, \mathcal{I}_j^c, \mathcal{T}$) (Alg. 4).
14:     $\mathcal{S} := \mathcal{S} \wedge \mathcal{S}_o$ - guard constraints
15:     $\mathcal{S} := \mathcal{S} \wedge \left\{ \bigwedge_{i,k \in \mathcal{I}_j} \mathcal{T}[j, i] = \mathcal{T}[j, k] \right\}$
16:     $\mathcal{S} := \mathcal{S} \wedge \left\{ \bigwedge_{i \in \mathcal{I}_j, k \in \mathcal{I}_j^c} \mathcal{T}[j, i] < \mathcal{T}[j, k] \right\}$
17: end for
18: return $(\mathcal{S}, \mathcal{T})$
```

Algorithm 3 Constraints generation for the path ρ (components that synchronise)

```
Require: Network $\mathcal{N}$, path index $j$, discrete path $\rho$, set of transition indices $\mathcal{I}$, sequence of time constraints $\mathcal{T}$ and clock valuation $\eta$
Ensure: Formula $\mathcal{S}$, matrix $\mathcal{T}$ and clock valuation $\eta_{\text{next}}$
 1: Function Sync_Constr($\mathcal{N}, j, \rho, \mathcal{I}, \mathcal{T}, \eta$)
 2: for $i \in \mathcal{I}$ do
 3:     $\mathcal{T}[j, i] := 0$, $\mathcal{S} := \varnothing$
 4:     $e^{(i)} := (q_j^{(i)}, a, g^{(i)}, X^{(i)}, q_{j+1}^{(i)})$ - is the transition with maximal priority from location $q_j^{(i)}$, $a$ is the corresponding action, $g^{(i)}$ is the guard and $X^{(i)}$ is a set of clocks
 5:     for $x \in g^{(i)}$ do
 6:         if $g^{(i)}.x(1) = " \geqslant "$ or $g^{(i)}.x(1) = " > "$ then
 7:             $\mathcal{T}[j, i] := \max\{\mathcal{T}[j, i], g^{(i)}.x(2) - \eta(x)\}$
 8:         else if $g^{(i)}.x(1) = " \leqslant "$ then
 9:             $\mathcal{S} := \mathcal{S} \wedge \{\eta^{(i)}(x) \leqslant g^{(i)}.x(2)\}$
10:         else
11:             $\mathcal{S} := \mathcal{S} \wedge \{\eta^{(i)}(x) < g^{(i)}.x(2)\}$
12:         end if
13:     end for
14:     $\eta_{\text{next}} := \eta + \mathcal{T}[j, i]$
15: end for
16: return $(\mathcal{S}, \mathcal{T}, \eta_{\text{next}})$
```

Algorithm 4 Constraints generation for the path ρ (components that don't synchronise)

```
Require: Network $\mathcal{N}$, path index $j$, discrete path $\rho$, set of
    component indices $\mathcal{I}$ and matrix of time constraints $\mathcal{T}$
Ensure: Matrix $\mathcal{T}$ over $\Gamma$
 1: Function NSync_Constr($\mathcal{N}, j, \rho, \mathcal{I}, \mathcal{T}$)
 2: for $k \in \mathcal{I}$ do
 3:     $\mathcal{T}[j,k] := 0$
 4:     $e^{(k)} := (q_j^{(k)}, a, g^{(k)}, X^{(k)}, q_{j+1}^{(k)})$
 5:     for $x \in g^{(k)}$ do
 6:         if $g^{(k)}.x(1) = " \geqslant "$ or $g^{(k)}.x(1) = " > "$ then
 7:             $\mathcal{T}[j,k] := \max\{\mathcal{T}[j,k], g^{(k)}.x(2) - \eta(x)\}$
 8:         end if
 9:     end for
10: end for
11: return $\mathcal{T}$
```

j	0	1
	$\mathcal{I}_0=\{7\}, \mathcal{I}_0^s=\{2\}, \mathcal{I}_0^c=\{6\}$	$\mathcal{I}_1=\{4\}, \mathcal{I}_1^s=\varnothing, \mathcal{I}_1^c=\{6,7\}$
$\mathcal{A}_1$	$\eta_0(t)=0$	$\mathcal{T}[1,4]=T-J$ $\eta_1(t)=J$
$\mathcal{A}_2$	$\mathcal{T}[0,7]=J, \mathcal{T}[0,6]=P$ $\eta_0(x)=0$ $\eta_0(y)=0$	$\mathcal{T}[1,6]=P-J, \mathcal{T}[1,7]=J$ $\eta_1(x)=J$ $\eta_1(y)=0$
$\mathcal{S}$	$J<P$	$T-J<P-J\wedge$ $T-J<J \wedge J<P$

Table 1: Example constraints for Algorithm 2

The function Constr_Gen from Algorithm 5 generates the formula $\mathcal{S}_\varphi$ for the CMTL formula φ. It uses the function Sum_Gen to generate constraints for a basic counting formula BCF $\mathbb{B}$ (see Definition 1). The function Sum_Gen creates two sets, L and U, for the lower and upper bounds, respectively, appearing in $\mathbb{B}$. The sequence $\bar{w}$ contains the ordered set of elements from $L \cup U$ and the function f maps an element of $L \cup U$ to an element of the sequence $\bar{w}$. The main phase of Sum_Gen involves generating *all possible orderings* of the transitions occurring in $\bar{\rho}$, where $\bar{\rho}$ is the untimed suffix of length i of ρ, with respect to the elements of $\bar{w}$. This is achieved with the outer disjunction over the set $\{0, \ldots, |\bar{\rho}| - 1\}$. For every possible ordering, the algorithm checks whether the formula $\sum_{j \in J} c_j \sum_{\iota = y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \bar{\rho}[\iota] \bowtie b$ holds.

Example 2. In this example we show the execution of Algorithm 5 (function Sum_Gen) for the path in Example 1 and formula $\varphi = \#_5^7\mathsf{VP} \geqslant 1$. The first column of the table shows all possible ordering of variables y_1 and y_2. Note that, for a path of length 2, $y_i \in \{0,1\}$, $i \in \{1,2\}$. The second column of the table shows how the time constraints are generated, while the third column shows the formula that checks whether there is at least one VP action present in the interval of time 5 to 7. Here $t_0 = \mathcal{T}[0,7]$ and $t_1 = \mathcal{T}[1,4]$ (see Example 1).

The remaining steps of Algorithm 5 generate constraints for a CMTL formula. The algorithm proceeds by induction over the structure of the formula and generates the formula

Ordering	$\bigwedge_{z \in \{0,\ldots,|\bar{w}|\}} \bar{\rho}@\bar{w}(z) = y_z$	φ
$(y_1 = 0 \wedge y_2 = 0)$	$(t_0 > 5)$	**false**
$(y_1 = 0 \wedge y_2 = 1)$	$(t_0 > 5) \wedge$ $(t_0 + t_1 > 7 \wedge t_0 < 7)$	**true**
$(y_1 = 1 \wedge y_2 = 1)$	$(t_0 + t_1 > 5 \wedge t_0 < 5) \wedge$ $(t_0 + t_1 > 7 \wedge t_0 < 7)$	**false**

Table 2: Example constraints for BCF $\varphi = \#_5^7\mathsf{VP} \geqslant 1$.

$\mathcal{S}$ over Γ. Finally, line 8 of Algorithm 1 takes the disjunction between the current expression for the formula $\mathcal{S}$ and the conjunction between $\mathcal{S}_\rho$ for the path ρ and $\mathcal{S}_\varphi$ for the CMTL formula φ. If every path in $\mathcal{N}$ does not satisfy the formula φ then $\mathcal{S} :=$ **false**. The formula $\mathcal{S}$ is used to compute the value of an objective function described in the next section.

In this paper we state three main theorems. Theorem 1 deals with the correctness of the generated formula $\mathcal{S}$. Theorem 2 shows that any CMTL formula is preserved, even if the domain of the parameter Γ is the set of rational numbers. Finally, Theorem 3 shows how the number of parameter values covered by the contraints in Algorithm 1 increases with the number of samples.

Theorem 1 *Let $\rho = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ be the timed path of the network $\mathcal{N}$ of TIOAs and $i \in \mathbb{N}$ an index ($i \leqslant n$). For every CMTL formula φ it holds*

$$(\rho, i) \models^{\mathcal{N}} \varphi \quad \text{iff} \quad \mathsf{Constr\_Gen}(\rho, i, \varphi, \mathcal{T}) = \mathbf{true},$$

where $\mathcal{T}[j, \cdot] := t_j$ for all $j < n$.

Let Γ^n be the set of all natural numbers and Γ^r be the set of all rational numbers representing the parameter set Γ. More formally, for every $\vartheta \in \Gamma^r$ we have $\lfloor \vartheta \rfloor \in \Gamma^n$ or $\lceil \vartheta \rceil \in \Gamma^n$. Here we assume that the domain of Γ is bounded.

Theorem 2 *Let $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1, \ldots, m\}\}$ be a network of TIOAs $\mathcal{A}^{(i)}$ and $n \in \mathbb{N}$. We have that*

$$\bigvee_{\vartheta \in \Gamma^n} (\mathcal{S} \wedge \mathsf{Constr\_Gen}(\rho, 0, \varphi, \mathcal{T})) = \bigvee_{\vartheta' \in \Gamma^r} (\mathcal{S}' \wedge \mathsf{Constr\_Gen}(\rho', 0, \varphi, \mathcal{T}'))$$

where $\rho := \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta)$, $\rho' := \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta')$, $(\mathcal{S}$, $\mathcal{T}$ $) := \mathsf{Path\_Constr\_Gen}(\mathcal{N}, \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta))$ and $(\mathcal{S}', \mathcal{T}') :=$ Path_ Constr_Gen $(\mathcal{N}, \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta'))$ for all $\vartheta \in \Gamma^n$, and $\vartheta' \in \Gamma^r$. Here we say that two constraints are equal if they share the same solution set.

Let $|\Gamma^n|$ be the size of Γ^n and $\#_{\mathcal{S}} = \frac{1}{|\Gamma^n|} \sum_{i=1}^{|\Gamma^n|} \mathbf{1}(\vartheta_i \in \mathcal{S})$, where $\mathbf{1}(\vartheta_i \in \mathcal{S})$ is the characteristic function, be the fraction of the total number of parameter valuations that satisfy the formula $\mathcal{S}$. Let $\#_{\mathcal{S}_k}^k = \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}(\vartheta_i \in \mathcal{S}_k)$ be the estimator of $\#_{\mathcal{S}}$ based on a sample of size $k < |\Gamma^n|$. Here $\mathcal{S}_k$ denotes the constraints corresponding to k discrete paths. Note that $\lim_{k \to |\Gamma^n|} \#_{\mathcal{S}_k}^k = \#_{\mathcal{S}}$.

Theorem 3 (Finite Population Sampling) *Given a sample size of $k < |\Gamma^n|$ we have that the standard error of $\#_{\mathcal{S}_k}^k$ is $\sigma_\epsilon = \frac{\sigma(\#_{\mathcal{S}_k}^k)}{\sqrt{k}} \sqrt{1 - \frac{k}{|\Gamma^n|}}$, where σ is standard deviation.*

Algorithm 5 Constraints generation for CMTL formulas

```
Require: Discrete path $\rho$, path index $i$, CMTL formula $\varphi$ and sequence of time constraints $\mathcal{T}$
Ensure: Formula $\mathcal{S}$
 1: Function $\mathsf{Constr\_Gen}(\rho, i, \varphi, \mathcal{T})$
 2: case($\varphi$) :
        $\varphi = \sum_{j \in J} c_j \#_{\ell_j}^{u_j} a_j \bowtie b$   :   $\mathcal{S} := \mathsf{Sum\_Gen}(\rho, i, \varphi)$
        $\varphi = \neg\varphi_1$   :   $\mathcal{S} := \neg\mathsf{Constr\_Gen}(\rho, i, \varphi_1, \mathcal{T})$
        $\varphi = \varphi_1 \wedge \varphi_2$   :   $\mathcal{S} := \mathsf{Constr\_Gen}(\rho, i, \varphi_1, \mathcal{T}) \wedge \mathsf{Constr\_Gen}(\rho, i, \varphi_2, \mathcal{T})$
        $\varphi = \varphi_1 \mathcal{U}^{[\ell,u]} \varphi_2$   :   $\mathcal{S} := (\bigvee_{i'=i}^{n} \mathsf{Constr\_Gen}(\rho, i', \varphi_2, \mathcal{T}) \wedge \ell \leq \sum_{k=i}^{i'} \bar{\mathcal{T}}[k] \leq u \wedge (\bigwedge_{i''=i}^{i'-1} \mathsf{Constr\_Gen}(\rho, i'', \varphi_1, \mathcal{T})))$
 3: return $\mathcal{S}$
 4: Function $\mathsf{Sum\_Gen}(\rho, i, \varphi, \mathcal{T})$
 5: $\bar{\rho} := \rho[\![i]\!]$, $L := \{l_j \mid j \in J\}$, $U := \{u_j \mid j \in J\}$ and $\bar{w} := \text{sort}(L \cup U)$
 6: $f$ maps an element of $L \cup U$ to an element of $\bar{w}$
 7: $\mathcal{S} := \bigvee_{\substack{y_k \in \{0,\ldots,|\bar{\rho}|-1\} \\ y_1 \leqslant \cdots \leqslant y_{|\bar{w}|}}} \left( \bigwedge_{z \in \{0,\ldots,|\bar{w}|\}} \bar{\rho}@\bar{w}(z) = y_z \right) \wedge \left( \sum_{j \in J} c_j \sum_{\iota = y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \bar{\rho}[\iota] \bowtie b \right)$
 8: return $\mathcal{S}$
 9: We define $(\bar{\rho}@\bar{w}(z) = y_z) := \left( \sum_{\iota=0}^{y_z} \bar{\mathcal{T}}[i+\iota] > \bar{w}(z) \wedge \sum_{\iota=0}^{y_z-1} \bar{\mathcal{T}}[i+\iota] < \bar{w}(z) \right)$ and
10: $\bar{\mathcal{T}}[j] := \mathcal{T}[j, \cdot]$ - to be the sequence of time constraints that correspond to taken transitions for all $j \leq |\rho| - 1$
```

Proofs of the theorems can be found in [9]. From the above theorem it follows that error σ_ϵ decreases with $\frac{\sqrt{1-\frac{k}{|\Gamma^n|}}}{\sqrt{k}}$ as the sample size increases.

5.2 Parameter optimisation

After generating the formula $\mathcal{S}$, we are ready to tackle the parameter synthesis problem, i.e., to find the *optimal* solution for the set of controllable parameters Γ_c with respect to an objective function $\mathcal{O}$. The optimal solution will be the one that maximises $\mathcal{O}$. We emphasise that there is no single optimal solution. The optimal solution should be the one that best fits the domain of the application. For this reason, we present two different choices of the objective functions that we believe are relevant for the pacemaker case study presented in the next section. The first consists in maximizing the value of an integral over the domain $\mathcal{V}(\Gamma_u)$, i.e.,

$$\text{opt}_v := \operatorname*{argsup}_{\vartheta_c \in \mathcal{V}(\Gamma_c)} \int_{\vartheta_u \in \mathcal{V}(\Gamma_u), (\vartheta_c, \vartheta_u) \in \mathcal{S}} \mathit{Distr}_{\Gamma_u}(d\vartheta_u).$$

Here $\mathcal{V}(\Gamma_c)$ and $\mathcal{V}(\Gamma_u)$ denote the set of all possible values for parameters Γ_c and Γ_u, respectively. The idea of the integral is to find a valuation for the controllable parameters that satisfies the formula $\mathcal{S}$ and that also maximises the probability mass associated with the uncontrollable parameter set. In the above objective function, we assume that the set of uncontrollable parameters, Γ_u, are distributed according to $\mathit{Distr}_{\Gamma_u}$. If $\mathit{Distr}_{\Gamma_u}$ is a discrete probability distribution, then the above objective function can be reduced to a linear programming problem, for which standard solution algorithms exist. If $\mathit{Distr}_{\Gamma_u}$ is continuous, then it is always possible to discretise $\mathcal{V}(\Gamma_u)$ or apply Monte Carlo simulation techniques. In the special case when $\mathit{Distr}_{\Gamma_u}$ is the uniform distribution, the above objective function becomes a volume integral parametric in $\mathcal{V}(\Gamma_c)$, for which efficient solutions also exist [21].

In some practical examples it does not suffice to find an optimal solution unless it is also *robust* (see [12, 11] for various definitions of robustness). Intuitively, we say that a set of model parameters is robust if a small variation at the values of the model parameters does not affect the validity of the formula under consideration. We explain the concept with an abstract example. For instance, consider the problem of finding optimal parameters for an embedded device. Running Algorithm 1, we find the optimal controllable parameters opt_v for which the device satisfies the safety property φ. Let opt'_v be a sub-optimal solution, i.e., $\text{opt}'_v < \text{opt}_v$. Now consider that a small change of ϵ in opt_v invalidates φ, whereas the same change in opt'_v does not affect the validity of φ. In this case it makes sense to chose opt'_v rather than opt_v because opt'_v is more "robust". In light of this example, we introduce a new optimal parameter synthesis problem (opt_r) that captures the concept of robustness:

$$B_\epsilon(\vartheta) = \{\vartheta' \in \mathcal{V}(\Gamma) \mid ||\vartheta' - \vartheta||_\infty \leqslant \epsilon\},$$

$$\text{opt}_r := \operatorname*{argsup}_{\vartheta_c \in \mathcal{V}(\Gamma_c)} \{\sup_{\epsilon}\{\epsilon \mid \vartheta_u \in \mathcal{V}(\Gamma_u), B_\epsilon((\vartheta_c, \vartheta_u)) \subseteq \mathcal{S}\}\}$$

where the norm $||\vartheta' - \vartheta||_\infty$ for $\vartheta, \vartheta' \in \mathcal{V}(\Gamma)$ and $\Gamma = \{v_1, \ldots, v_n\}$ is defined as $\max\{|\vartheta(v_1) - \vartheta'(v_1)|, \ldots, |\vartheta(v_{n'}) - \vartheta'(v_n)|\}$. Note that the above optimisation problem can be transformed into a linear programming problem.

6. IMPLEMENTATION

We have implemented all the algorithms in Python for the full fragment of CMTL, using Z3 theorem prover [2] for constraint solving. Just as the synthesis technique was described in terms of constraints (Section 5.1) and optimization (Section 5.2), we also describe implementation along the same lines.

Our implementation currently assumes one controlled parameter and one uncontrolled parameter. Each parameter has a lower and an upper bound, which is encoded as a constraint (in Z3). Even with bounds, the parameter space is too large, and therefore we sample points from which we synthesize the parameter values. Namely, for each sampled point (which consists of a controllable value and an uncontrollable value), a discrete path and the corresponding timed path, safety and energy constraints are generated. The disjunction of the constraints generated from each sampled point, conjuncted with the parameter bounds constraint, represents a subset of the parameter values within the parameter bounds. This subset encodes the synthesized parameter values that satisfy the specified safety and energy constraints.

We determine the volume of a synthesized value by sampling the set of uncontrollable parameters and checking with the Z3 SMT solver if the sampled value of the uncontrollable parameter satisfies the generated constraints. The volume is the ratio between the total number of values that satisfy the constraints and the total number of sample points. We choose the controllable parameter value that gives the largest volume. To synthesise a robust value we first pick a seed point from the set of parameters values. Then we check to see if points that are ε distance away from the seed, in both controlled and uncontrolled directions, are also valid parameter values, with an increasing ε starting from the smallest possible value of 1 (since parameter values are integers). We use the Z3 SMT solver to check if all the parameters in the rectangle (defined by the sup norm) are valid. This process is shown in Fig. 2 as a search for the largest rectangle centered around the chosen point, y, that is within the triangle. The process continues until an invalid parameter value is found, i.e. the rectangle goes outside of the triangle, or until some upper bound for epsilon is reached. We choose the controllable parameter value that gives the largest ε.

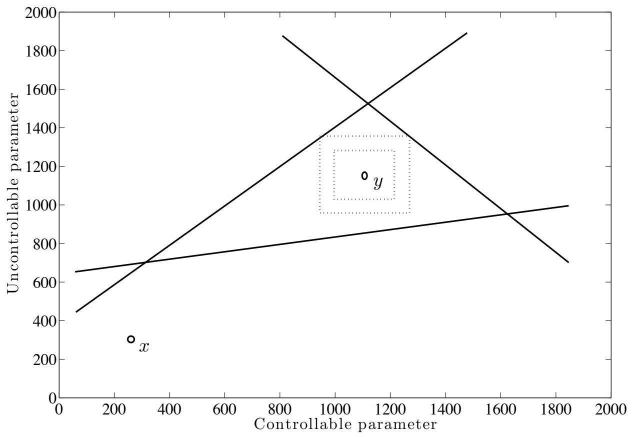


Figure 2: Robustness Example

7. CASE STUDY

In this section we present a pacemaker case study where we apply the techniques developed in the paper. The goal is to synthesise one of the parameters of the pacemaker in order to ensure its correct behaviour, while at the same time optimising the value of a given objective function.

The pacemaker is a medical device that is implanted under the skin of the chest and has the purpose of delivering electrical signals to the human heart in order to maintain a given heart rate. It delivers the electrical signals using two leads, one for the atrium and one for the ventricle. The pacemaker can pace the heart as well as read the signal (action potential) generated by the heart.

We solve the pacemaker synthesis problem by modeling the heart, the pacemaker and their composition using TIOAs.

The heart model is composed of three TIOA components (see Fig. 3): *atrium*, *conduction* and *ventricle*. The *atrium* component (Fig. 3a) is responsible for generating atrial beats. It waits for a signal (action potential) from the SA-node, which is the natural pacemaker of the heart, or from the pacemaker by means of action AP. The firing time of the SA-node is modelled by a transition labelled with the guard $t \geq \mathsf{PP}$, which defines the natural frequency of the heart. Note that the SA-node belongs to the *atrial* component. The *atrial* component generates atrial beats by means of action Aget. We also model a blocking period denoted by the paremeter AERP. The purpose of the period is to deny consecutive stimulation of the atrium from the pacemaker. That is, a stimulus from the pacemaker is blocked if the time difference between the previous stimulus and the current one is less than AERP.

The *conduction* component models the propagation delay of the atrial signal through the atrium and the AV-node. The delay is given by the parameter TAVD. When the action potential originating from the atrium reaches the ventricle, the *conduction* component notifies the *ventricle* component by means of action CD.

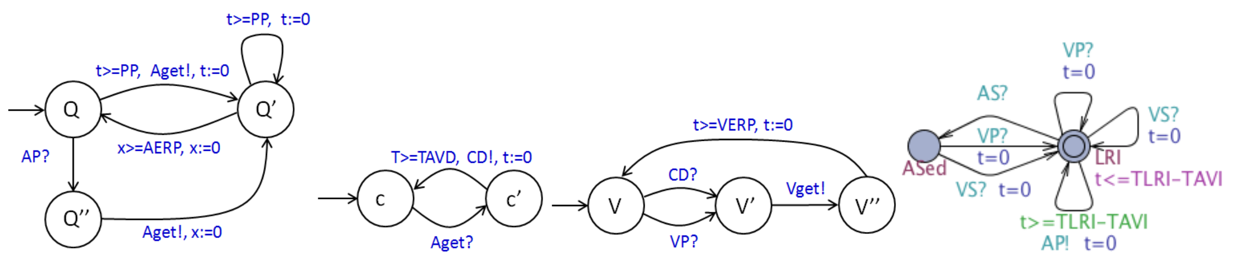
The *ventricle* component is responsible for generating ventricle beats. It can receive a signal VP from the pacemaker or from the conduction component CD. The *ventricle* component generates ventricle beats by means of action Vget. Here we also model a blocking period denoted by the parameter VERP.

We emphasise that the heart model in Fig. 3 can be tailored to individual patients. For instance, both PP and TAVD can be estimated from the patient electrocardiogram. The parameters AERP and VERP can be estimated at the time of the implantation of the pacemaker. We treat PP, TAVD, AERP and VERP as uncontrollable parameters.

For this case study, we consider the basic pacemaker model introduced in [16] which consists of five TIOA components: the lower rate interval (LRI) component (see Fig. 3d), the atrio-ventricular interval (AVI) component, the upper rate interval (URI) component, the post ventricular atrial refractory period (PVARP) component and the ventricular refractory period (VRP) component. In this case study, we focus only on the LRI component and omit descriptions of the components for reasons of space; see [16] for more detail.

The LRI component keeps the heart at a given minimal rate, which is denoted by the parameter $\mathsf{TLRI} - \mathsf{TAVI}$. Here the parameter TAVI denotes the atrial-ventricular delay, which has the same meaning as TAVD. The difference between TAVI and TAVD is that the former is a controllable parameter which can be modified in order adjust the pacing rate, whereas the latter is defined by the heart and varies from patient to patient. The LRI component stops pacing the atrium as soon as the input action AS is enabled. This occurs when the SA-node fires. Note that, for the sake of clarity, we do not depict the priorities in the pacemaker and heart components.

Now the goal is to synthesise the pacemaker parameter $\mathsf{TLRI} - \mathsf{TAVI}$ (we consider the difference as a single parame-



(a) Atrium component (b) Conduction component (c) Ventricle component (d) Pacemaker LRI component

Figure 3: Pacemaker and heart components.

ter), which is the amount of time that the pacemaker waits before delivering an atrial pace. $\mathsf{TLRI} - \mathsf{TAVI}$ is a controllable parameter in our model and its value is critical for the correct functioning of the pacemaker device. We check the correctness of the pacemaker against the following CMTL formulas:

1) $\Box^{[0,\tau]}(\#_0^\tau \mathtt{Vget} \geqslant B_1 \wedge \#_0^\tau \mathtt{Vget} \leqslant B_2)$ - safety property.
2) $1 \cdot \#_0^\tau \mathtt{AP} + 2 \cdot \#_0^\tau \mathtt{VP} \leqslant \mathsf{E}$ - energy property.

The first formula states that it is always the case that the heart beats (ventricle beat) at least B_1 and no more than B_2 times in the interval of time $[0, \tau]$. The second formula states that the pacemaker consumes no more than E units of energy in the interval of time $[0, \tau]$. For every atrial beat AP the pacemaker consumes 1 unit, and for every ventricle beat VP it consumes 2 units.

In the experimental results we pick PP to be the only uncontrollable parameter following a uniform distribution and all other parameters are constant. We also choose the path length $n := 15$ and the time bound τ above from the set $\{1000, 1500\}$. Here all the values of parameters are in milliseconds. We note that all the experiments run in less than an hour.

For the safety property we synthesise the $\mathsf{TLRI} - \mathsf{TAVI}$ parameter. We set $\tau := 1000$ (milliseconds), $B_1 := 1$ and $B_2 := 2$, meaning that the pacemaker should maintain a heart rhythm between 60 and 120 beats per minute. We sample 160 parameter values for PP and $\mathsf{TLRI} - \mathsf{TAVI}$ and generate discrete paths of length 15. For all the paths and the formula we generate the set of constraints $\mathcal{S}$. The task is to synthesise a value for $\mathsf{TLRI} - \mathsf{TAVI}$ such that the validity of the safety formula is preserved for *any* value of PP. As discussed in Section 5.2, the optimal parameter valuation might not be robust. In this example, we have that a value for $\mathsf{TLRI} - \mathsf{TAVI}$ of around 1000 is optimal (we have used 200 sample points to compute the volume objective function). This is due to the fact that, when $\mathsf{TLRI} - \mathsf{TAVI}$ is in that range, the pacemaker model satisfies the safety formula φ for the largest set of parameter valuations of PP. However, setting $\mathsf{TLRI} - \mathsf{TAVI}$ to 1000 is not robust from an implementation point of view. In fact, if we have a small perturbation of $\mathsf{TLRI} - \mathsf{TAVI}$, say from 1000 to 1001, the safety formula φ is invalidated. A more robust choice is to pick values for $\mathsf{TLRI} - \mathsf{TAVI}$ around 850 (and this is the value returned by Algorithm 1 using the robust objective function). For the robust objective function we have used 500 sample points. Picking the value of $\mathsf{TLRI} - \mathsf{TAVI}$ around 850 reduces the number of PP behaviours that we cover. However, in this case, a small change of $\mathsf{TLRI} - \mathsf{TAVI}$ will not invalidate the safety formula φ. We remark that some major pacemaker manufacturers, such as Boston Scientific [1], suggest that these values be set between 750 and 900, which validates the result of our algorithms.

In addition to ensuring the correct number of beats, we can also guarantee that the pacemaker consumes no more than a given amount of energy in an interval of time. For the energy property we run three experiments with $\mathsf{E} = 40$. We pick two time bounds $\tau = 1000$ and $\tau = 1500$. In the first two experiments we compute the volume objective function for $\mathsf{TLRI} - \mathsf{TAVI}$ parameter (see Fig.4a). In Fig.4a we can see that the maximal volume increases with the value of $\mathsf{TLRI} - \mathsf{TAVI}$ until the time bound τ (blue plot for $\tau = 1000$ and red plot for $\tau = 1500$) and then it remains constant. Intuitively, for a pacemaker to consume the smallest amount of energy it has to pace as little as possible. Our experimental result confirms this intuition by synthesising the maximal value of the atrial pacing parameter $\mathsf{TLRI} - \mathsf{TAVI}$. Note that the maximal value of $\mathsf{TLRI} - \mathsf{TAVI}$ for $\tau = 1500$ does not make the pacemaker safe. The safe value for $\mathsf{TLRI} - \mathsf{TAVI}$ is around 850. In the second experiment we compute the robust objective function for $\mathsf{TLRI} - \mathsf{TAVI}$. In Fig.4b we see that the most robust value for $\mathsf{TLRI} - \mathsf{TAVI}$ is around 1000. A safe value for $\mathsf{TLRI} - \mathsf{TAVI}$ should be less than 1000. Therefore, to ensure the safety and the minimal energy consumption of the pacemaker one should check the conjunction of both the safety and the energy properties.

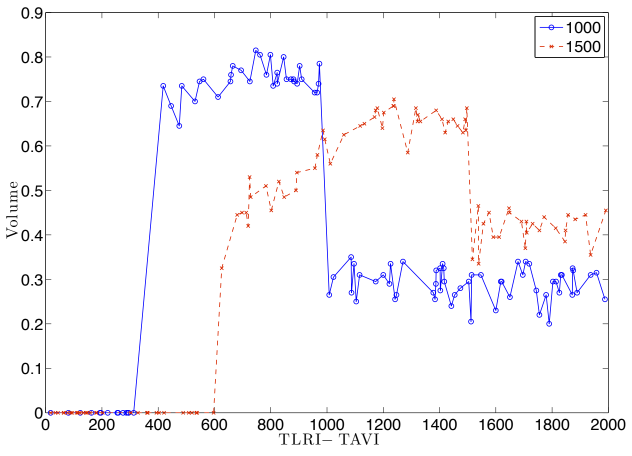
8. CONCLUSIONS

We have developed an algorithm to synthesise optimal timing delays for real-time embedded systems modelled as an extension of TIOA with priorities and parametric guards. Focusing on medical devices as an application domain, we propose CMTL, an extension of the Metric Temporal Logic with counting formulas, which can express fundamental safety properties for pacemakers, as well as quantitative requirements for energy consumption. As future work, we plan to improve the efficiency of the algorithms in order to tackle the high complexity of constraint generation algorithms.

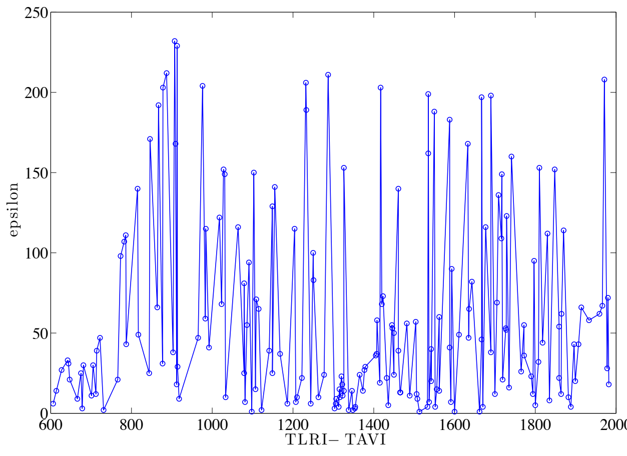
Acknowledgements. This work is supported by the ERC AdG VERIWARE, ERC PoC VERIPACE and the Institute for the Future of Computing, Oxford Martin School.

9. REFERENCES

[1] Pacemaker system specification. 2007.
[2] Z3 theorem prover. 2007.



(a) Maximal volume



(b) Robustness

Figure 4: The value of the objective function for controllable parameter TLRI − TAVI and uncontrollable paremeter PP.

[3] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[4] É. André, T. Chatain, L. Fribourg, and E. Encrenaz. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(05):819–836, 2009.

[5] É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP*, pages 76–90. Springer, 2010.

[6] C. Baier and J.-P. Katoen. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[7] L. Bozzelli and S. La Torre. Decision problems for lower/upper bound parametric timed automata. *FMSD*, 35(2):121–151, 2009.

[8] V. Bruyere and J.-F. Raskin. Real-time model-checking: Parameters everywhere. In *FST&TCS 2003*, pages 100–111. Springer, 2003.

[9] M. Diciolla, C. H. P. Kim, M. Kwiatkowska, and A. Mereacre. Synthesising optimal timing delays for timed I/O automata. Technical Report CS-RR-14-07, University of Oxford, July 2014.

[10] L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.

[11] G. E. Fainekos and G. J. Pappas. Robust sampling for MITL specifications. In *Formal Modeling and Analysis of Timed Systems*, pages 147–162. Springer, 2007.

[12] G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta. Robustness of model-based simulations. In *RTSS 2009*, pages 345–354. IEEE, 2009.

[13] Y. Hirshfeld and A. Rabinovich. Expressiveness of metric modalities for continuous time. In *Computer Science–Theory and Applications*, pages 211–220. Springer, 2006.

[14] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183–220, 2002.

[15] Z. Jiang, M. Pajic, and R. Mangharam. Cyber–physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

[16] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *TACAS*, pages 188–203. Springer, 2012.

[17] A. Jovanovic and M. Kwiatkowska. Parameter synthesis for probabilistic timed automata using stochastic game abstractions. In *RP*. Springer, 2014. To appear.

[18] A. Jovanović, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. In *TACAS*, pages 401–415. Springer, 2013.

[19] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The theory of timed I/O automata. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–137, 2010.

[20] M. Knapik and W. Penczek. Bounded model checking for parametric timed automata. In *Transactions on Petri Nets and Other Models of Concurrency V*, pages 141–159. Springer, 2012.

[21] J. B. Lasserre and E. S. Zeron. A Laplace transform algorithm for the volume of a convex polytope. *Journal of the ACM*, 48(6):1126–1140, 2001.

[22] A. Rabinovich. Complexity of metric temporal logics with counting and the Pnueli modalities. *Theoretical Computer Science*, 411(22):2331–2342, 2010.

[23] L.-M. Traonouez. A parametric counterexample refinement approach for robust timed specifications. *arXiv preprint arXiv:1207.4269*, 2012.