Department of Computer Science

# Synthesising robust and optimal parameters for cardiac pacemakers using symbolic and evolutionary computation techniques

**Marta Kwiatkowska, Alexandru Mereacre, Nicola Paoletti, and Andrea Patane'**

CS-RR-15-09


Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

**Abstract.** We consider the problem of automatically finding safe and robust values of timing parameters of cardiac pacemaker models so that a quantitative objective, such as the pacemaker energy consumption or its cardiac output (a heamodynamic indicator of the human heart), is optimised in a finite path. The models are given as parametric networks of timed I/O automata with data, which extend timed I/O automata with priorities, real variables and real-valued functions, and specifications as Counting Metric Temporal Logic (CMTL) formulas. We formulate the parameter synthesis as a bilevel optimisation problem, where the quantitative objective (the outer problem) is optimised in the solution space obtained from optimising an inner problem that yields the maximal robustness for any parameter of the model. We develop an SMT-based method for solving the inner problem through a discrete encoding, and combine it with evolutionary algorithms and simulations to solve the outer optimisation task. We apply our approach to the composition of a (non-linear) multi-component heart model with the parametric dual chamber pacemaker model in order to find the values of multiple timing parameters of the pacemaker for different heart diseases.

**Keywords:** parameter synthesis, timed I/O automata, cardiac pacemaker, satisfiability modulo theories, evolutionary strategies

## 1 Introduction

*Motivation.* The growing demand for wearable health monitoring devices, from fitness apps running on smart watches to implantable devices such as cardiac pacemakers and glucose monitoring, calls for design methodologies that can ensure their safety, effectiveness and energy efficiency. Model-based verification [9,21,31] has proved useful in establishing key correctness properties of cardiac pacemakers [19], but the approach has limitations, in that it is not clear how to redesign the model if it fails to satisfy a given property. Instead, the parameter synthesis problem aims to automatically find optimal values of parameters to guarantee that a given property is satisfied. Similarly to verification, this problem has prohibitive complexity and may suffer from undecidability, typically tackled through discretisation of the parameter space.

In [15], we presented a parameter synthesis method for timed I/O automata (TIOA) that optimises the choice of timing delays for a given objective function to guarantee that a property, expressed in Counting MTL, a generalisation of Metric Temporal Logic, is satisfied. The method is based on exploring finite discrete paths and the corresponding timing constraints. We have applied the techniques to cardiac pacemakers, deriving robust values for safety and energy efficiency, but could only guarantee partial coverage via sampling, as fully exhaustive exploration was not practical.

*Contribution.* In this paper, we tackle, for the first time, the problem of ensuring effectiveness for pacemakers defined in terms of cardiac output (a heamodynamic indicator of the human heart), as well as safety. To this end, we extend the models and logic of [15] with real-valued data variables, and provide a novel method for synthesising timing delays that are simultaneously safe *and* robust, whilst guaranteeing that a given quantitative objective is optimised. This is formulated as a bi-level optimisation problem, which we solve through a combination of symbolic, SMT-based, analysis of finite

paths based on discrete encoding (for the inner problem), with evolutionary computation techniques (for the outer problem). We consider a novel multi-component heart model given as a network of TIOA with data [5] and extend it in order to compute the cardiac output. We apply the developed techniques to the synthesis of multiple pacemaker parameters for different heart conditions, in order to optimise, at the outer level, either energy consumption or cardiac output, on top of the solution space that yields a safe heart rhythm (formulated as a CMTL property) with maximum robustness.

*Related work.* The undecidability of the parametric reachability problem is proved in [16]. The majority of work for timed systems concerns synthesis from logic formulas, e.g. [8], with the exception of [1,2] who consider a reference valuation. In [7,23], the authors show PSPACE-completeness of the emptiness problem and TCTL, respectively. Robustness under a given timed perturbation is considered in [39] and parameter synthesis for reachability for probabilistic timed automata in [22]. SMT-based verification of timed and hybrid systems has received a lot of attention recently, see e.g. [10]. In [26], the authors present an SMT-based timed system extension to the IC3 algorithm. [25] and [27] respectively develop real-time bounded model checking (BMC) approaches for LTL and CTL. [20] presents an SMT technique to generate inductive invariants for hybrid systems. Sturm et al [38] applies real quantifier elimination tools to synthesise continuous and switched dynamical systems. The dReal solver [18] uses a relaxed notion of satisfiability in order to provide decision procedures for non-linear hybrid systems.

In this paper, we extend the model and logic of [15], and replace the path exploration with a fully symbolic BMC-based algorithm. We adopt the pacemaker model from [21] but consider a different heart model [5], which we enhance with the blood pressure component.

## 2 Background

### 2.1 Timed I/O automata with priorities and data

We extend the timed I/O automata model with priorities of [15] with data variables. Let $\mathcal{X}$ be a set of *non-negative* real-valued variables, called *clocks*. Let $\mathcal{D}$ be a set of real-valued variables, called *data*. A variable valuation is a function $\eta = \eta_{|\mathcal{X}} \cup \eta_{|\mathcal{D}}$ where $\eta_{|\mathcal{X}} : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and $\eta_{|\mathcal{D}} : \mathcal{D} \to \mathbb{R}$. We denote the set of variables with $V = \mathcal{X} \cup \mathcal{D}$. Let $\Gamma$ be a set of real-valued *parameters*. A parameter valuation is a function $\gamma : \Gamma \to \mathbb{R}$ mapping each parameter $p$ to a value in its domain $\mathrm{dom}(p) \subseteq \mathbb{R}$.

Let $\mathcal{Y}$ be a set and $\mathcal{V}(\mathcal{Y})$ denote the set of all valuations over $\mathcal{Y}$. We consider guard constraints of the form $\bigwedge_i v_i \bowtie_i f_i$, where $v_i \in \mathcal{X}$ is a clock, $\bowtie_i \in \{<, \leqslant, >, \geqslant\}$ and $f_i : \mathcal{V}(\mathcal{D}) \times \mathcal{V}(\Gamma) \to \mathbb{R}$ is a real-valued function over data variable and parameter valuations. A variable valuation $\eta$ and a parameter valuation $\gamma$ satisfy the above constraint iff $\bigwedge_i \eta(v_i) \bowtie_i f_i(\eta_{|\mathcal{D}}, \gamma)$ holds. We denote with $\mathcal{B}(V)$ the set of guard constraints over $V$. The reset of a set of variables $V' \subseteq V$ is an arbitrary function $r : V' \times \mathcal{V}(V) \times \mathcal{V}(\Gamma) \to \mathbb{R}$. Given valuations $\eta$ and $\gamma$, $\eta$ is updated by reset $r$ to the valuation $\eta[r] = \{v \mapsto r(v, \eta, \gamma) \mid v \in V'\} \cup \{v \mapsto \eta(v) \mid v \notin V'\}$ that applies the reset $r$ to the variables in $V'$ and leaves unchanged the others. We denote with $\mathcal{R}$ the

set of reset functions. The valuation $\eta$ after time $\delta \in \mathbb{R}_{\geq 0}$ has elapsed is denoted with $\eta{+}\delta$ and is such that $\eta{+}\delta(v) = \eta(v) + \delta$ if $v \in \mathcal{X}$ and $\eta{+}\delta(v) = \eta(v)$ otherwise. This implies that all clocks proceed at the same speed and data variables are not affected by the passage of time.

**Definition 1 (Deterministic Timed I/O Automaton with Priority and Data).** *A deterministic timed I/O automaton (TIOA) with priority and data* $\mathcal{A} = (\mathcal{X}, \Gamma, \mathcal{D}, Q, q_0, \Sigma_{\text{in}}, \Sigma_{\text{out}}, \rightarrow)$ *consists of:*

- *A finite set of clocks* $\mathcal{X}$*, data variables* $\mathcal{D}$ *and parameters* $\Gamma$*.*
- *A finite set of locations* $Q$*, with the initial location* $q_0 \in Q$*.*
- *A finite set of input actions* $\Sigma_{\text{in}}$ *and a finite set of output actions* $\Sigma_{\text{out}}$*.*
- *A finite set of edges* $\rightarrow \subseteq Q \times (\Sigma_{\text{in}} \cup \Sigma_{\text{out}}) \times \mathbb{N} \times \mathcal{B}(V) \times \mathcal{R} \times Q$*. Each edge* $e = (q, a, pr, g, r, q')$ *is described by a source location* $q$*, an action* $a$*, a priority* $pr$*, a guard* $g$*, a reset* $r$ *and a target* $q'$*.*

We require that priorities define a total ordering of the edges out of any location, and that output actions have higher priority than input actions. The TIOAs as defined above are able to synchronise on matching input and output actions, thus forming *networks of communicating automata*. We say that an output edge is *enabled* when the associated guard holds. On the other hand, an input edge is enabled when both its guard holds and it can synchronise with a matching output action fired by another component of the network. A component of a network of TIOAs is enabled if, from its current location, there is at least one outgoing edge enabled. Also, we assume that output edges are *urgent*, meaning that they are taken as soon as they become enabled. As shown in [15], priority and urgency imply that *the TIOA is deterministic*.

**Definition 2 (Network of TIOAs).** *A network of TIOAs with* $m$ *components is a tuple* $\mathcal{N} = (\{\mathcal{A}^1, \ldots, \mathcal{A}^m\}, \mathcal{X}, \Gamma, \mathcal{D}, \Sigma_{\text{in}}, \Sigma_{\text{out}})$ *of TIOAs, where*

- *for* $j = 1, \ldots, m$*,* $\mathcal{A}^j = (\mathcal{X}, \Gamma, \mathcal{D}, Q^j, q_0^j, \Sigma_{\text{in}}, \Sigma_{\text{out}}, \rightarrow^j)$ *is a TIOA,*
- $\mathcal{X}, \Gamma, \mathcal{D}, \Sigma_{\text{in}}, \Sigma_{\text{out}}$ *are the common sets of clocks, parameters, data variables, input and output actions, respectively,*

*We define the set of network modes by* $\boldsymbol{Q} = Q^1 \times \cdots \times Q^m$*, with initial mode* $\boldsymbol{q_0} = (q_0^1, \ldots, q_0^m)$ *and the initial variable valuation* $\eta_0$*. A state of the network is a pair* $(\boldsymbol{q}, \eta)$ *where* $q \in \boldsymbol{Q}$ *and* $\eta \in \mathcal{V}(V)$*.*

A *parametric network of TIOAs* is a network where the parameter valuation is unknown, and is denoted by $\mathcal{N}(\cdot)$. $\mathcal{N}(\gamma)$ denotes the network obtained by instantiating valuation $\gamma$. We describe the formal semantics of a network of TIOAs in terms of timed paths. In the following, we use the predicate $\mathsf{enabled}(\mathcal{N}, j, \boldsymbol{q}, \eta, \gamma)$ (see Section 5.1 for its formal encoding) to indicate whether the $j$-th component of network $\mathcal{N}$ is enabled from the network mode $\boldsymbol{q}$ under variable valuation $\eta$ and parameter valuation $\gamma$.

**Definition 3 (Path of a TIOA Network).** *Let* $\mathcal{N}$ *be a network of TIOAs and* $n \in \mathbb{N}^+$*. Let* $\rho = (\boldsymbol{q_0}, \eta_0) \xrightarrow{t_0} (\boldsymbol{q_1}, \eta_1) \xrightarrow{t_1} \cdots \xrightarrow{t_{n-2}} (\boldsymbol{q_{n-1}}, \eta_{n-1})$ *be a timed sequence of length* $n$ *where, for* $i = 1, \ldots, n-1$*,* $t_{i-1} \geq 0$*,* $\boldsymbol{q_i} \in \boldsymbol{Q}$ *and* $\eta_i$ *is a variable valuation. Then,* $\rho$ *is the* timed path *of network* $\mathcal{N}$ *if for any position* $i = 0, \ldots, n-2$*:*

*I) there exists at least one component enabled:* $\exists j.\ \mathsf{enabled}(\mathcal{N}, j, q_i^j, \eta_i + t_i, \gamma)$*; let* $E_{i,t_i}$ *be the set of such components;*

*II) each component* $j \in E_{i,t_i}$ *fires the edge* $e_i^j = (q_i^j, a_i^j, pr_i^j, g_i^j, r_i^j, q_{i+1}^j) \in \rightarrow^j$ *that is enabled and with maximum priority among the enabled edges;*

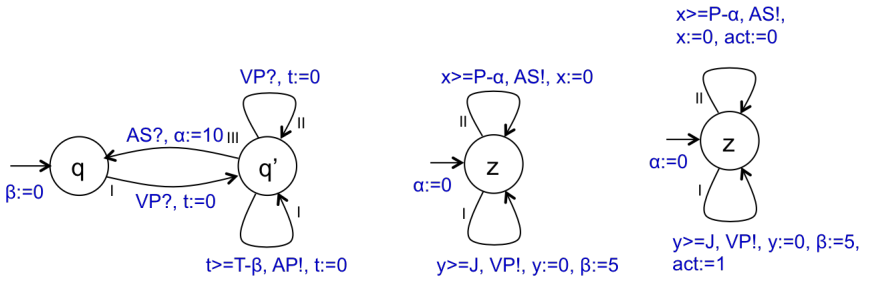
*III) the variable valuation is updated according to the elapsed time and the resets of enabled components*[1]*:* $\eta_{i+1} := \eta_i + t_i[\bigcup_{j \in E_{i,t_i}} r_i^j]$*; and*

*IV)* $t_i$ *is the least time for which there are enabled components:* $\forall t' < t_i.\ E_{i,t'} = \varnothing$.

*For* $k, m \in \mathbb{N}$, $\rho[k] = (\boldsymbol{q_k}, \eta_k)$ *is the* $k$*-th state of the path,* $\rho^{[k,m]}$ *is the subpath of length* $m - k + 1$ *starting at position* $k$*,* $\rho\langle k \rangle = t_k$ *is the* $k$*-th delay and* $\rho\langle k, m \rangle = \sum_{k'=k}^{m} t_{k'}$ *is the total time spent in the subpath* $\rho^{[k,m]}$*. For* $t \in \mathbb{R}_{\geqslant 0}$*, we denote with* $\rho@t$ *the smallest index* $o$ *such that* $\sum_{k=0}^{o} \rho\langle k \rangle > t$*. If no such index exists, then* $\rho@t = n - 1$*. When the network is parametric, i.e. of the form* $\mathcal{N}(\cdot)$*, the corresponding parametric path is denoted with* $\rho(\cdot)$.

In the following, we denote with $\Pi$ the set of finite timed paths. Given $t \in \mathbb{R}^{\geq 0}$, we also define the *path up to time* $t$ as the path $\rho$ with length $|\rho| = \rho@t$, that is, such that: (a) $\rho\langle 0, |\rho| - 1 \rangle \leq t$; and (b) let $\rho'$ be the 1-step extension of $\rho$, then $\rho'\langle 0, |\rho'| - 1 \rangle > t$. We say that a parametric path $\rho(\cdot)$ is up to time $t$ if, for each $\gamma \in \mathcal{V}(\Gamma)$, $\rho(\gamma)$ is up to $t$.

*Example 1.* Consider the TIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ in Fig. 1. The automata $\mathcal{A}_1$ and $\mathcal{A}_2$ form




(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$ (c) $\mathcal{A}_2$ with additional variables

Fig. 1: Example network $\mathcal{N}$ with two components, $\mathcal{A}_1$ and $\mathcal{A}_2$.

a network. They communicate with each other by means of actions $\{\mathsf{VP}, \mathsf{AP}, \mathsf{AS}\} \in \Sigma_{\text{in}} \cup \Sigma_{\text{out}}$. We distinguish input (marked with ?) and output actions (marked with !). For instance, when automaton $\mathcal{A}_2$ takes a transition and outputs the action VP!, the automaton $\mathcal{A}_1$ synchronises by taking the corresponding transition with the input action

[1] In order to have consistent resets, we assume that different components cannot update the same variable with different values during the same transition.

VP?. We use Roman numbers to denote priorities, with the lowest number denoting the highest priority. The network $\mathcal{N}$ has three clocks $t$, $x$ and $y$, and two variables $\alpha$, and $\beta$. The initial mode of the network is $(q, z)$ and the initial values for the $\alpha$ and $\beta$ variables are zero. Each edge of the automaton is labelled with an action, a guard over the set of clocks and a reset over the set of clocks and variables. For instance, one of the edges from $q'$ to $q'$ is labelled with the guard $t \geq T - \beta$, action AP and clock reset $t := 0$. The network $\mathcal{N}$ has also three parameters $T$, $P$ and $J$.

There are two ways to take an edge. First, when an input action is enabled. Second, when the clock satisfies a given guard. For example, automaton $\mathcal{A}_2$ has two transitions labelled with the conditions $x \geq P - \alpha$ and $y \geq J$. 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 $y$ and assigning the value of five to the variable $\beta$. When multiple transitions are enabled in a location, then the one with the highest priority will be taken. Consider the finite path below, where transitions are labelled with enabled output actions:

$$\begin{gathered}((q,z),(\alpha=0,\beta=0,t=0,x=0,y=0))\\ \downarrow J,\mathsf{VP}\\ ((q',z),(\alpha=0,\beta=5,t=0,x=J,y=0))\\ \downarrow T{-}5,\mathsf{AP}\\ ((q',z),(\alpha=0,\beta=5,t=0,x=J{+}T{-}5,y=T{-}5))\\ \downarrow P{-}J{-}T{+}5,\mathsf{AS}\\ ((q,z),(\alpha=10,\beta=5,t=P{-}J{-}T{+}5,x=0,y=P{-}J)).\end{gathered}$$

Each element of the tuple represents the state of the network and the values of the variables. The network starts in the initial state $(q, z)$ with the values of the variables $(\alpha = 0, \beta = 0, t = 0, x = 0, y = 0)$. In the automaton $\mathcal{A}_2$, after $J$ time units have passed, the guard $y \geq J$ becomes true and the corresponding transition is triggered at this point, outputting the action VP and resetting the clock $t$ to 0 and the variable $\beta$ to 5. The automaton $\mathcal{A}_1$ then synchronises with $\mathcal{A}_2$ via the matching input, VP, which moves the automaton to $q'$. Then $\mathcal{A}_1$ takes transition labelled with $T - \beta$ and $\mathcal{A}_2$ does no transition. Then the automaton takes the transitions labelled with $P - \alpha$ outputting the action AS and the state of the network becomes $(q, z)$. Note that, in order to take the transition labelled with action VP in $\mathcal{N}$, the parameters $P$ and $J$ have to satisfy the urgency constraint $P - \alpha < J$. Similar relations can be derived for the remaining transitions of the path.

### 2.2 Counting MTL

We work with the *Counting Metric Temporal Logic* (CMTL), which is an extension of MTL with the counting operator (#) [33,15], now interpreted over TIOAs with data.
Let $\mathcal{E}(V)$ be the set of constraints $\bigwedge_i c_i \bowtie_i g_i$ over variables in $V = \mathcal{X} \cup \mathcal{D}$, where $c_i \in \mathbb{R}$, $\bowtie_i \in \{<, \leq, =, \geq, >\}$ and $g_i : \mathcal{V}(V) \to \mathbb{R}$ is a real-valued function over $V$. To this end, we replace CMTL atomic propositions by predicates from $\mathcal{E}(V)$. For instance, given two variables $x, y \in V$, a constraint from $\mathcal{E}(V)$ is $x = 1 \wedge y \geq 10$. The syntax of

CMTL is defined by

$$\varphi ::= e \mid \sum_{j\in J} c_j \#_{\ell_j}^{u_j} e_j \bowtie b \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathcal{U}^{[\ell,u]} \varphi,$$

where $J$ is a finite set of indices, $\bowtie \in \{>, \geqslant, <, \leqslant\}$, $b \in \mathbb{Z}$, $c_j \in \mathbb{Z}$, $\ell \in \mathbb{R}_{\geqslant 0}$, $\ell_j \in \mathbb{R}_{\geqslant 0}$, $u \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$, $u_j \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$ are time points such that $\ell \leqslant u$ and $\ell_j \leqslant u_j$, and $e, e_j \in \mathcal{E}(V)$ for all $j \in J$. The counting term $\#_{\ell_j}^{u_j} e_j$ counts how many times, in the interval of time $[\ell_j, u_j]$, $e_j$ holds true. Such terms can be combined to form a so-called basic counting formula $\sum_{j\in J} c_j \#_{\ell_j}^{u_j} e_j \bowtie b$, i.e. a linear constraint (with integer coefficients) over counting terms. The semantics of CMTL is defined over timed paths as follows.

**Definition 4.** *Let $\rho = (\boldsymbol{q_0}, \eta_0) \xrightarrow{t_0} (\boldsymbol{q_1}, \eta_1) \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} (\boldsymbol{q_n}, \eta_n)$ be the finite timed path of the network $\mathcal{N}(\gamma)$ of TIOAs with parameter valuation $\gamma$ and $i \in \mathbb{N}$ be an index. We say that $\mathcal{N}$* satisfies *$\varphi$ at $i$, denoted $(\rho, i) \models^{\mathcal{N}} \varphi$, iff*

$$\begin{aligned}
&(\rho,i) \models^{\mathcal{N}} e && \text{iff } \eta_i \models e\\
&(\rho,i) \models^{\mathcal{N}} \sum_{j\in J} c_j \#_{\ell_j}^{u_j} e_j \bowtie b && \text{iff } \left(\sum_{j\in J} c_j \sum_{k=\rho^{[i,|\rho|]}@\ell_j}^{\rho^{[i,|\rho|]}@u_j-1} \mathbf{1}\left(\eta_k \models e_j\right)\right) \bowtie b\\
&(\rho,i) \models^{\mathcal{N}} \varphi_1 \wedge \varphi_2 && \text{iff } (\rho,i) \models^{\mathcal{N}} \varphi_1 \wedge (\rho,i) \models^{\mathcal{N}} \varphi_2\\
&(\rho,i) \models^{\mathcal{N}} \neg\varphi_1 && \text{iff } (\rho,i) \not\models^{\mathcal{N}} \varphi_1\\
&(\rho,i) \models^{\mathcal{N}} \varphi_1\, \mathcal{U}^{[\ell,u]} \varphi_2 && \text{iff } \exists i'.\ i \leqslant i' \text{ s.t. } \sum_{k=i}^{i'} \rho\langle k\rangle \in [\ell,u] \wedge (\rho,i') \models^{\mathcal{N}} \varphi_2 \wedge\\
& && \quad \forall i''.\ i \leqslant i'' < i' \wedge (\rho,i'') \models^{\mathcal{N}} \varphi_1,
\end{aligned}$$

*where $\varphi_1$, $\varphi_2$ are CMTL formulas, $i', i'' \in \mathbb{N}$, $e \in \mathcal{E}(V)$, $\ell_j \in \mathbb{R}_{\geqslant 0}$ and $\mathbf{1}\left(\eta_k \models e_j\right)$ is the characteristic function that returns $1$ if $\eta_k \models e_j$ and $0$ otherwise.*

We define $\Diamond^{[\ell,u]}\varphi := \mathbf{true}\, \mathcal{U}^{[\ell,u]}\varphi$ and $\Box^{[\ell,u]}\varphi := \neg\Diamond^{[\ell,u]}\neg\varphi$. Details on the decidability and complexity of the logic can be found in [32,33].

*Example 2.* Let $\mathcal{A}_2$ from Fig. 1c be the modified version of the TIOAs $\mathcal{A}_2$ from Fig. 1, where we add a new variable $act$. The variable $act$ identifies the presence of the action VP or AS through expression $act = 1$ or $act = 0$, respectively. We also modify automaton $\mathcal{A}_1$ by adding the update $act := 2$ to the edge labelled with the action AP!. We set the initial valuation to $act := -1$. We consider the following CMTL formula which states that, starting from any time in the interval $[0, 100]$, the number of performed VP actions in the interval of time $[0, 7]$ has to be no lower than 1 and at most 4:

$$\Box^{[0,100]} \left(\#_0^7(act = 1) \geq 1 \wedge \#_0^7(act = 1) \leq 4\right) \tag{1}$$

## 3 Robust Optimal Synthesis Problem

We introduce a parameter synthesis problem for networks of TIOAs that asks for the parameter valuation that, first, maximises parameter robustness and, second, minimises some cost function, e.g. energy consumption. This problem is motivated by the fact that, in the design of medical devices, safety is of paramount importance and it is desirable to maintain patient's physiological properties in a robust way w.r.t. perturbations of parameter values. We express such properties in CMTL, which we use to formulate the requirement of a safe heart rhythm (see Sect. 4). We also assume a cost function $f : \Pi \to \mathbb{R}$ that maps timed paths to reals.

Thus, we aim at finding a valuation $\gamma$ with maximum *robustness radius*, i.e. a quantity $\epsilon \in \mathbb{R}^+$ such that a CMTL formula $\phi$ is guaranteed to hold for any perturbation of $\gamma$ bounded by $\epsilon$. Then, we synthesise parameters that yield the minimum cost on top of the solution space with maximum $\epsilon$. This problem can be effectively formulated as a *bi-level optimisation* problem (see e.g. [12]), where robustness maximisation and cost optimisation represent the so-called *inner and outer problems*, respectively.

Let $\epsilon \in \mathbb{R}^+$ and $\gamma \in \mathcal{V}(\Gamma)$. The $\epsilon$-bounded perturbations of $\gamma$ are denoted by the set $B_\epsilon(\gamma) = \{\gamma' \mid \forall p \in \Gamma.\ |\gamma'(p) - \gamma(p)| \leq \epsilon\}$. Given property $\phi$ and path $\rho(\cdot)$ of network $\mathcal{N}(\cdot)$, we say that a parameter valuation $\gamma$ is *$\epsilon$-robust* w.r.t. $\phi$ if it holds that $\forall \gamma' \in B_\epsilon(\gamma).\ \rho(\gamma') \models^{\mathcal{N}(\gamma')} \phi$. Note that, for arbitrary $\epsilon$, there can exist perturbed valuations outside the domain of parameters, i.e. $B_\epsilon(\gamma) \not\subseteq \mathcal{V}(\Gamma)$. In the following, we only admit the case when $B_\epsilon(\gamma) \subseteq \mathcal{V}(\Gamma)$.

*Problem 1 (Robust Optimal Synthesis).*
Let $\mathcal{N}(\cdot)$ be a parametric network of TIOAs, $t \in \mathbb{R}^{\geq 0}$, $k \in \mathbb{N}^+$, $\phi$ be a CMTL property and $f$ be a cost function. Let $\rho(\cdot)$ be the parametric path of $\mathcal{N}(\cdot)$ of length $k$ and $\rho'(\cdot)$ be the parametric path up to time $t$. The *robust optimal synthesis problem* is finding a parameter valuation $\gamma$ that solves the following bi-level optimisation problem:

$$\min_{\gamma_o \in \mathcal{V}(\Gamma)} f(\rho'(\gamma_o)) \quad \text{subject to } \gamma_o \in \arg\max_{\gamma_i \in \mathcal{V}(\Gamma)} \epsilon$$
$$\text{subject to } B_\epsilon(\gamma_i) \subseteq \mathcal{V}(\Gamma) \text{ and } \forall \gamma' \in B_\epsilon(\gamma_i).\ \rho(\gamma') \models^{\mathcal{N}(\gamma')} \phi.$$

Note that in the above problem the path lengths for the inner and outer problem are arbitrary and in general not interrelated. In practice, as explained in Sect. 5.3, $f$ is evaluated through simulation and thus we can support longer lengths for $\rho'$.

*Running example.* We formulate an instance of Problem 1 by taking the CMTL property in Example 2 and the modified network defined therein. In the inner problem, we take the parametric path $\rho$ of length $k = 15$. In the outer problem, we consider the path $\rho'$ up to time $t = 100$ and aim to minimise the number of AS actions performed along $\rho'$, leading to the objective $f(\rho') = \#_0^{100}\,(act = 0)$.

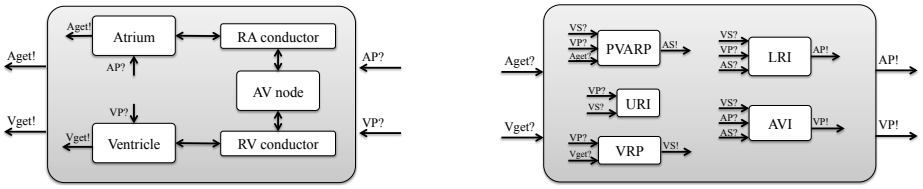
## 4 Heart and Pacemaker Models

In this section we describe the heart and the pacemaker models, and provide the properties and functions for the synthesis problem. The pacemaker has the role of maintaining

the synchronisation between the atrium and the ventricle. In particular, we consider a basic DDD pacemaker specification [37], that is, pacing and sensing both the atrium and the ventricle, and provide a TIOA network adapted from [21].

The heart model is used to reproduce the propagation of the cardiac action potential, and is a TIOA translation of the model by Lian et al [30] (see [5] for details). In [5], the authors provide a probabilistic model where parameters are drawn from probability distributions derived from patients data. Here, parameters are set to a given fixed value, thus resulting in a fully deterministic model. The composed heart-pacemaker model consists of 11 TIOA components, with 11 clock variables, 7 data variables and 18 action labels.

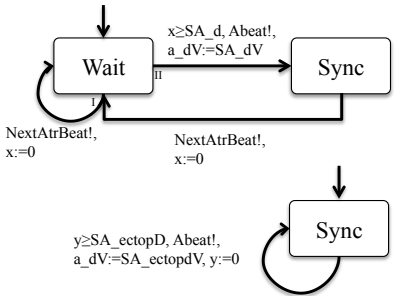
*Heart model.* The high-level structure of the model is depicted in Fig. 2a. It comprises five main conduction nodes and two main conduction paths: from the atrium to the ventricle (*antegrade* conduction) or vice-versa (*retrograde*). The *Atrium* component is responsible for modelling the sinoatrial (SA) node, i.e. the natural pacemaker of the heart. This has a predefined firing rate, given by parameter $\mathrm{SA\_d}$. It can also produce ectopic beats with rate $\mathrm{SA\_ectopD}$. In Fig. 3 we depict the sub-network that models the atrium. In Fig. 3a, we illustrate the automata for the SA node and the ectopic beat generator, respectively. When their clocks ($x$ and $y$) satisfy the corresponding guards, the output action Abeat is produced, which indicates the generation of an atrial impulse. $a\_dV$ is a variable storing the action potential in the atrium, which might vary depending on whether the beat is regular or ectopic. The TIOA in Fig. 3b models the current state of the atrium. In the *Refractory* mode the atrium component starts a timer, modelled by clock $z$. After the atrial refractory period has elapsed ($z \geq$ Atr_refrD), the atrium changes its mode to *Excitable*. In this mode, it can receive three types of actions: an SA node signal, Abeat; a pacing signal from the pacemaker, AP; or a retrograde signal from the ventricle, AtrRetroReached. Finally, the atrium generates the output action Aget to notify the pacemaker of the atrial impulse, and returns to the *Refractory* mode. The $aPeriod$ variable is used to store the duration of the last atrial cycle.



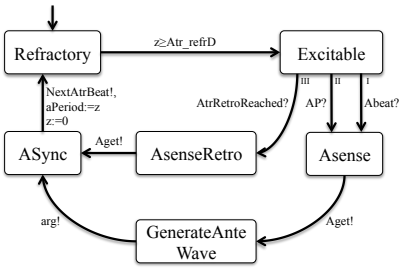

(a) Heart model (b) Pacemaker model

Fig. 2: Heart and pacemaker models.

The *Ventricle* component is similar to the *Atrium* component, i.e., it has an intrinsic beat generator and an ectopic beat generator. In addition, it has a variable $vPeriod$ to store the ventricular period. The *RA conductor* and *RV conductor* are used to model the propagation delay of the action potential from the atrium to the ventricle and back (antegrade or retrograde conduction).




(a) SA node (top) and ectopic beat generation (bottom) components




(b) Atrium state component

Fig. 3: The *Atrium* component.

The *AV node* component is responsible for delaying the entrance of the action potential from the atrium into the ventricle. The AV conduction delay (AVD) is given by an exponential function $\text{AVD} := \text{AVD}_{\min} + \alpha \exp(\frac{-T_{\text{rec}}}{\tau_c})$, where $T_{\text{rec}}$ is the AV recovery time, $\text{AVD}_{\min}$ is the shortest AVD when $T_{\text{rec}} \to \infty$, $\alpha$ is the longest extension of AVD when $T_{\text{rec}} = 0$ and $\tau_c$ is the conduction time constant. More details of the AV node constants and parameters are provided in [5]. We remark that some guards of the heart model components contain non-linear functions.

*Pacemaker model.* We briefly describe the components of the basic DDD pacemaker model shown in Fig. 2b (see [9] for details): *AVI* maintains the synchronisation between the atrium and the ventricle, *LRI* sets a lower bound for the heart rate, *URI* sets an upper bound for the heart rate, *PVARP* detects intrinsic atrial events, and *VRP* detects intrinsic ventricle events. The pacemaker communicates with the heart model by means of four actions: AP (atrial pace), VP (ventricle pace), Aget (atrial sense) and Vget (ventricle sense). Every component has associated a timing parameter, which we discuss in Sect. 6. By changing these parameters one can control, for instance, the pacing rate in the atrium or ventricle, or the signal propagation delay from the atrium to the ventricle.

*Cardiac output.* Cardiac output ($CO$, $\text{cm}^3 \cdot \text{s}^{-1}$) is an important heamodynamic indicator that describes the volume of blood pumped by a ventricle over time and is used in clinical practice to monitor patients with heart conditions.

We compute $CO$ following the modified Windkessel method in [17] for modelling the cardiovascular system, where the aortic flow is modelled as a square wave, which is more realistic than the standard Windkessel model (see e.g. [3]) where it is approximated as a series of impulses.

The ventricular period alternates in two parts: the systole ($T_S$) and the diastole ($T_D$). During systole the ventricles first contract and

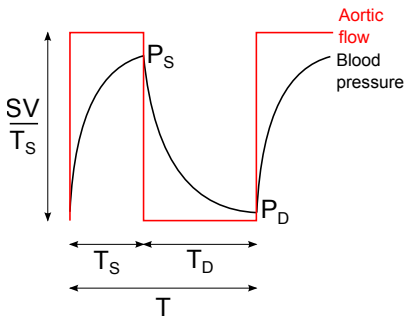



Fig. 4: Blood pressure (black) computed considering a square wave aortic flow (red).

reach a maximum pressure giving rise to a heart beat. Then, they drive the blood flow to the pulmonary and aortic valves. During diastole, ventricular pressure drops to its minimum and blood starts flowing from the atria to the ventricles until it is ejected in the next systole. Therefore, each wave in the the aortic flow signal has amplitude $\frac{SV}{T_S}$ and period $T_S$ (see Fig. 4), where $SV$ ($\text{cm}^3$) is the stroke volume, i.e. the difference between the volume at diastole and that at systole.

Following [17], the maximum arterial pressure at systole $P_S$ (mmHg) is computed as $P_S = P_D \cdot \exp\left(\frac{-T_S}{R \cdot C}\right) + R \cdot \frac{\text{SV}}{T_S}\left(1 - \exp\left(\frac{-T_S}{R \cdot C}\right)\right)$, where $R$ ($\text{mmHg}\cdot\text{s}\cdot\text{cm}^{-3}$) and $C$ ($\text{cm}^3\cdot\text{mmHg}^{-1}$) are the aortic resistance and compliance parameters, respectively. The first term of the equation describes the decay of the diastolic pressure at the previous cycle with rate $\frac{1}{R \cdot C}$ during $T_S$, while the second term accounts for the pressure given by the aortic flow. The equation for the minimum pressure at diastole, $P_D$ (mmHg), is $P_D = P_S \cdot \exp\left(\frac{-T_D}{R \cdot C}\right)$, which describes the decay of the pressure at systole during $T_D$. Finally, $CO = C \cdot (P_S - P_D)/T$ depends on the difference between $P_S$ and $P_D$ over the heart period $T = T_S + T_D$.

At each ventricular event (actions $\mathsf{Vget}$ or $\mathsf{VP}$), we compute the CO of the previous cycle, assuming $T_S = 0.25 \cdot T$ and $T_D = 0.75 \cdot T$, where $T$ is the time elapsed from the previous ventricular event. We consider the parameters of a healthy patient, namely, $C = 1.3$, $R = 0.9$ and $\text{SV} = 90$ [24]. The initial diastolic pressure is set to 80.

*Properties.* We consider two variants of the robust optimal synthesis problems (Probl. 1) where, in the outer problem, we minimise the energy consumption of the pacemaker or optimise the cardiac output, respectively. In the first variant, we take a simplified model assuming that only atrial and ventricular pacing contribute to energy consumption, with weights 2 and 3 respectively. Thus, the cost function is given by $f(\rho') = 2 \cdot \#_0^{60000}(act = AP) + 3 \cdot \#_0^{60000}(act = VP)$, where $act$ is a variable storing the last performed action, and $AP, VP \in \mathbb{R}$ identify an $\mathsf{AP}$ or a $\mathsf{VP}$ action, respectively. By abuse of notation, the operator $\#$ denotes a function of $\rho'$, i.e., it counts how often an expression from $\mathcal{E}(V)$ holds true in the path $\rho'$. In the second variant, we seek to find parameters that make the computed cardiac output as close as possible to a given reference value $\overline{CO}$ (set to 80 $\text{cm}^3\cdot\text{s}^{-1}$). Let $Vbeat(\rho')$ be the set of states along path $\rho'$ where a ventricular beat happens. Then, $f(\rho') = \left(\sum_{(\boldsymbol{q},\eta)\in Vbeat(\rho')} |\eta(CO) - \overline{CO}|\right) / \left(|Vbeat(\rho')|\right)$ is the cost function, i.e. the mean difference between the valuations of $CO$ in $Vbeat(\rho')$ and the reference $\overline{CO}$.

For both variants, in the inner problem we find parameters that guarantee a safe heart rhythm with maximum robustness. We express this requirement by imposing that the ventricular period (the time distance between two consecutive ventricular beats) is always within the interval $[500, 1000]$ ms, i.e. between 60 and 120 BPM. The corresponding CMTL property is $\phi = \Box^{[0,T]}(vPeriod \in [500, 1000])$, where the time bound $T = \rho\langle 0, |\rho| - 1\rangle$ is chosen to cover the whole length of path $\rho$.

## 5 Parameter Synthesis Algorithms

We now present methods to solve the bi-level optimisation problem introduced in Sect. 3. We restrict ourselves to *safety properties*, i.e. formulas of the form $\Box^{[\ell,u]}\phi$, and to

reachability properties, of the form $\Diamond^{[\ell,u]}\phi$, where $\phi$ is a CMTL formula without temporal operators. First, we describe the algorithm for the inner problem, namely maximising the robustness radius w.r.t. a given property. Second, we devise a method for optimising the outer objective by exploiting the solution of the inner problem. Both methods are based on encoding the TIOA model and the synthesis problem as a Satisfiability Modulo Theories (SMT) problem, which we describe below. In our implementation, we use the Z3 theorem prover [13].

### 5.1 SMT encoding

We provide a discrete encoding of the problem in the theory of bit-vectors (SMT $\mathrm{UF\_BV}$). Clocks, parameters and variables are expressed as bit-vectors and therefore have finite domains. Through this discrete SMT encoding, the verification problem for TIOAs with data and CMTL formulas is NEXPTIME [28]. Below we describe techniques and constraints that characterizes the encoding.

*Abstraction of non-integer values and non-linear functions.* We now introduce the notion of TIOAs extended with *non-deterministic variables*, which are necessary to provide a sound encoding of the problem. Intuitively, such variables can be updated in a non-deterministic way to a number of possible values.

Let $\bar{V}$ be the set of non-deterministic variables; let $\bar{v} \in \bar{V}$ and $\eta$ and $\gamma$ be valuations of variables and parameters, respectively. Then, the reset $r$ of $\bar{v}$ induces a set of admissible values $r(\bar{v}, \eta, \gamma)$ and a set of admissible updated valuations $\eta[r] = \{\eta' \mid \eta'(\bar{v}) \in r(\bar{v}, \eta, \gamma)$ and $\eta'(v') = \eta(v')$ for $v' \neq \bar{v}\}$. This implies that a network $\mathcal{N}$ of thus extended TIOAs can have multiple admissible paths. We denote with $\Pi(\mathcal{N})$ the set of such paths. Clearly, fixing a valuation for the non-deterministic variables at each state of the path induces a deterministic path according to Def. 3.

Non-deterministic variables are used to provide an interval-based abstraction for non-integer values and non-linear functions as follows. Consider a generic update $y := f(x)$ with $f : \mathbb{D} \to \mathbb{R}$, where $y \in V$ and $\mathbb{D}$ is the discrete and finite domain of $f$. For each $x \in \mathbb{D}$, we pre-compute the discrete bounds of $f$ on $x$, $[f(x)^{\perp}, f(x)^{\top}]$ as

$$f(x)^{\perp} \leq \left\lfloor \min_{x' \in [x, x+1)} f(x') \right\rfloor \text{ and } f(x)^{\top} \geq \left\lceil \max_{x' \in [x, x+1)} f(x') \right\rceil$$

Then, we encode the update $y := f(x)$ as $y' \in [f(x)^{\perp}, f(x)^{\top}]$ using a non-deterministic variable $y' \in \bar{V}$. In this way, we provide a *conservative over-approximation* of the original system, since the above interval-based abstraction induces additional behaviours but preserves the original ones. As discussed later, this potentially leads to spurious counter-examples, i.e. valuations that violate the given property in the abstracted system but not in the original one. In this work, we did not implement a refinement step for excluding spurious counter-examples [11], but we experimentally evaluated their number (see Sect. 6). In general, the quality of the abstraction is affected by the dynamics of the functions involved, e.g. the presence of large variations in small intervals.

*Transition relation.* We illustrate the constraints that characterise the transition relation:

$$T(s, s') \iff \exists t, \boldsymbol{e}.\, T'(s, t, \boldsymbol{e}, s') \wedge \mathsf{priority}(s, \boldsymbol{e}) \wedge \mathsf{urgency}(s, t, \boldsymbol{e})$$

where $s = (\boldsymbol{q}, \eta)$ and $s' = (\boldsymbol{q}', \eta')$ are states of the network; $t$ is the time spent in $s$; and $\boldsymbol{e} = (e^1, \ldots, e^m)$ is the vector of the edges fired by the components of the network. Predicate $T'(s, t, \boldsymbol{e}, s')$ describes the transition relation when no priority and urgency constraints hold, and is defined as:

$$T'(s,t,\boldsymbol{e},s') = \left( \bigwedge_{j=1}^{m} \mathsf{enabled}(s,t,\boldsymbol{e},j) \wedge \mathsf{Trg}^j(e^j) = q'^j \wedge \mathsf{localReset}(s,t,\boldsymbol{e},\eta',j) \right) \wedge \bigwedge_{x \in V_{glbl}} \mathsf{globalReset}(\eta,t,\boldsymbol{e},\eta',x)$$

where $\mathsf{Trg}^j(e^j)$ gives the target location of edge $e^j$ in component $j$. In order to simplify the reset constraints, we treat separately clock and data variables that are local, i.e. updated in one single component $j$, $V^j$, from those that can be reset by multiple components $V_{glbl}$.

Predicate $\mathsf{enabled}(s, t, \boldsymbol{e}, j)$ tells whether, from state $s$, the edge taken by component $j$ is enabled:

$$\mathsf{enabled}(s,t,\boldsymbol{e},j) = \mathsf{Grd}^j(e^j, \eta + t) \ \wedge \ \mathsf{Src}^j(e^j) = q^j \wedge \left( \mathsf{Act}^j(e^j) \in \Sigma_{in} \implies \bigvee_{j' \neq j} \mathsf{Act}^j(e^j) = \overline{\mathsf{Act}^{j'}(e^{j'})} \right)$$

where $\mathsf{Grd}^j(e^j, \eta + t)$ evaluates the guard of $e^j$ in the variable valuation after time $t$ elapses; $\mathsf{Src}^j(e^j)$ gives the source location of $e^j$; $\mathsf{Act}^j(e^j)$ is the action; and with $\overline{\mathsf{Act}^{j'}(e^{j'})}$ we indicate the co-action of $\mathsf{Act}^{j'}(e^{j'})$, i.e. mapping an input to the corresponding output action (and vice-versa).

The following predicate handles the resets of local variables:

$$\mathsf{localReset}(s,t,\boldsymbol{e},\eta',j) = \bigwedge_{x \in V^j} \mathsf{ite}\left(\mathsf{Res}^j(e^j, \eta + t, x, \bot),\ \eta'(x) = \eta + t(x),\ \mathsf{Res}^j(e^j, \eta + t, x, \eta'(x))\right)$$

where ite is the if-then-else function; $\mathsf{Res}^j(e^j, \eta + t, x, \eta'(x))$ is a predicate imposing on $\eta'(x)$ the admissible resets of variable $x$ at edge $e^j$ and with variable valuation $\eta + t$. Note that this allows us to express the non-deterministic updates discussed above. Indeed, for fixed $e^j, \eta, t$ and $x$, $\mathsf{Res}^j(e^j, \eta + t, x, \eta'(x))$ can hold for multiple values of $\eta'(x)$, thus implementing a choice of admissible resets. When instead $x$ is not updated by $e^j$, $\mathsf{Res}^j(e^j, \eta + t, x, \bot)$ is true and $\mathsf{Res}^j(e^j, \eta + t, x, r)$ is false for any valuation $r$.

On the other hand, the update of a global variable $x$ is encoded by:

$$\mathsf{globalReset}(\eta, t, \boldsymbol{e}, \eta', x) = \mathsf{ite}(\bigvee_{j=1}^{m} \neg\mathsf{Res}^j(e^j, \eta + t, x, \bot), \tag{2}$$

$$\bigwedge_{j=1}^{m} \neg\mathsf{Res}^j(e^j, \eta + t, x, \bot) \implies \mathsf{Res}^j(e^j, \eta + t, x, \eta'(x)), \tag{3}$$

$$\eta'(x) = \eta + t(x)) \tag{4}$$

As explained in Def. 3, we need to assume that different components cannot update the same variable with different values during the same transition. Indeed assume that there exist two components $j_1$ and $j_2$ that update (deterministically) the same variable $x$ along the transition $\boldsymbol{e}$ with two values $r_1$ and $r_2$, respectively, such that $r_1 \neq r_2$. Then, the constraint at Eq. 3 would reduce to:

$$\mathsf{Res}^{j_1}(e^{j_1}, \eta + t, x, \eta'(x)) \wedge \mathsf{Res}^{j_2}(e^{j_2}, \eta + t, x, \eta'(x))$$
$$=$$
$$\eta'(x) = r_1 \wedge \eta'(x) = r_2$$

which is clearly unsatisfiable.

Finally, urgency and priority constraints are described by the following:

$$\mathsf{urgency}(s, t, \boldsymbol{e}) =$$

$$\bigvee_{0 \leq t' < t} \bigwedge_{j=1}^{m} \bigwedge_{e^j \in E^j} \mathsf{Act}^j(e^j) \in \Sigma_{out} \implies \neg(\mathsf{Grd}^j(e^j, \eta + t) \wedge \mathsf{Src}^j(e^j) = q^j)$$

$$\mathsf{priority}(s, \boldsymbol{e}) = \bigwedge_{j=1}^{m} \bigwedge_{e' \in \longrightarrow^j} \mathsf{enabled}(s, t, \boldsymbol{e'}, j) \implies \mathsf{Pri}^j(e^j) \leq \mathsf{Pri}^j(e')$$

where $\boldsymbol{e'} = (e^1, \ldots, e^{j-1}, e', e^{j+1}, \ldots, e^m)$ (i.e. equals to $\boldsymbol{e}$, but the $j$-th edge), and $\mathsf{Pri}^j(e^j)$ is the priority of edge $e^j$.

Note that in this encoding functions $\mathsf{Src}^j, \mathsf{Trg}^j, \mathsf{Act}^j, \mathsf{Res}^j, \mathsf{Grd}^j, \mathsf{Pri}^j$ are all *symbolic*, i.e. uninterpreted functions plus constraints for imposing their interpretation according to the model. Importantly, this enables *model synthesis* (not discussed here), when we drop the constraints for their interpretation.

*Quantifier elimination from urgency constraints.* We exploit the fact that, unlike classical hybrid automata, TIOAs do not allow continuous flows. This implies that the valuation of data variables can only change after a reset. Therefore, for any guard $g = \bigwedge_i v_i \bowtie_i f_i$, where $v_i$ is a clock, and $f_i$ is a function over data variables and parameters, the value of $f_i$ stays constant for all the time spent in the current state. In turn, this implies that, for each clause $c_i = (v_i \bowtie_i f_i)$ of the guard, there exists at most

one time value $\bar{t_i}$ such that $c_i$ holds for all $t' \leq \bar{t_i}$, and does not for $t' > \bar{t_i}$, or vice versa.

The idea is based on finding the minimum time $\bar{t}$ s.t. $g$ becomes true. This is calculated as $\bar{t} = \mathsf{ite}(t_> \leq t_<, t_>, \infty)$ where $t_>$ gives the minimum time s.t. all the clauses where $\bowtie_i \in \{>, \geq\}$ (the *delays*) are true, and $t_<$ gives the maximum time s.t. clauses where $\bowtie_i \in \{<, \leq\}$ (the *time-outs*) are true. Clearly, if $t_>$ is greater than $t_<$, the guard is always false from the current state and $\bar{t} = \infty$.

Let $\eta$ be the current variable valuation, $\eta_{|\mathcal{D}}$ the valuation restricted to data variables, and $\gamma$ the parameter valuation. Then,

$$t_> = \max\{\max_{v_i \geq f_i \in \mathsf{clauses}(g)}(y_i - \eta(v_i)), \max_{v_i > f_i \in \mathsf{clauses}(g)}(y_i - \eta(v_i) + 2^{-h})\}$$
$$t_< = \min\{\min_{v_i \leq f_i \in \mathsf{clauses}(g)}(y_i - \eta(v_i)), \min_{v_i < f_i \in \mathsf{clauses}(g)}(y_i - \eta(v_i) - 2^{-h})\}$$

where $y_i = f_i(\eta_{|\mathcal{D}}, \gamma)$ is the value of the function $f_i$; and $2^{-h}$ is the smallest positive real that we can express with precision $h$ in our discrete SMT encoding (see Sect. 5). Finally, by urgency, we can conclude that the time spent in the current state will be the least $\bar{t}$ among all the output edges[2], which allows dropping the above urgency constraint that contains quantifiers.

### 5.2 The inner problem

The main algorithm for solving the inner problem is given in Alg. 1, which extends the SMT-based method for *bounded model checking (BMC)* [4] in order to synthesise the space of parameters that yields maximum robustness. Below, we first explain the synthesis method for safety properties, and then show how this is adapted to reachability properties.

Given a safety property $\varphi$, the algorithm returns the maximum robustness radius $\epsilon$, a parameter valuation $\bar{\gamma}$ that is $\epsilon$-robust w.r.t. $\varphi$, and an *under-approximation* Unsafe of the true unsafe parameter valuations. We encode an SMT problem where the Unsafe region is built by searching for counter-examples (CEs) to safety, which amounts to finding valuations s.t. $\neg\varphi$ holds at some point in the path, up to a fixed length $n$. Enumerating all possible counter-examples up to $n$, especially when $n$ is large, is clearly infeasible. Here, we implement several solutions to overcome this problem.

First, we exploit *incremental solving*, so that CEs are computed step-wise, for increasing path lengths, exploiting the fact that SMT solvers can use the clauses learned in the previous steps to improve the solution time of the current step. Second, we include an algorithm for *counter-example generalization* (procedure GeneralizeCE, Alg. 2), that, given a CE, attempts to derive an unsafe region that contains the CE. Third, we *restrict the search space for counter-examples* to the extent necessary to prove the actual maximum robustness radius $\epsilon$, thus avoiding the computation of irrelevant CEs.

*Counter-example generation.* In Alg. 1, we first initialize $\epsilon$, Unsafe and $\bar{\gamma}$ (lines 2-4). The $Init$ predicate (line 5) is used to constrain the initial automata locations and

[2] Since we use digital clocks, we take the ceiling of the least $\bar{t}$

variable valuation. Command Assert adds in the SMT solver a formula that must hold true. At a generic step $k$ of the path, we first assert the safety property up to the current total time $\rho\langle 0, k\rangle$ if this is lower than the time bound $u$ (line 7). In this case, the assertion is named with a literal $p_k$, meaning that the satisfaction value of $\Box^{[\ell,\min(u,\rho\langle 0,k\rangle)]}\phi$ is the same as $p_k$. During the counter-example generation cycle (lines 8-17), MaxRadius procedure is called to update the maximum $\epsilon$ and $\epsilon$-robust valuation $\bar{\gamma}$ according to the current Unsafe region. This information is used to temporarily restrict the search space for CEs to the region $B_\epsilon(\bar{\gamma})$ (line 11). Solve $\neg p_k$ checks if the negated safety is satisfied under the current assertions. If so, the solver returns a model, in our case a counter-example $\gamma_{CE}$, which we generalize to $\gamma'_{CE}$ by calling GeneralizeCE. $\gamma'_{CE}$ is then excluded from the search space (line 15) and added to Unsafe (line 16). The Pop command removes from the solver all the constraints asserted after the last Push (in this case, only $B_\epsilon(\bar{\gamma})$). If instead no CEs can be found in $B_\epsilon(\bar{\gamma})$, we can conclude that, up to step $k$, $\epsilon$ is the actual max radius and $\bar{\gamma}$ is $\epsilon$-robust. Thus, we can exit the CE generation loop, assert the transition constraints (line 21) and increase the step to $k+1$. When $\epsilon < 1$, the algorithm terminates since it implies that no robust parameters exist (lines 18-19). In the algorithm, $T$ indicates the transition predicate between states of the path, i.e. $T(s, s') = \exists t.\ s \xrightarrow{t} s'$. We remark that, by bounding and discretising the parameter space, we can ensure that the cycle at lines 8-17 always terminates. Note that the bound $n$ on the path length is given in input to the algorithm. Other stopping criteria could be considered based on, for instance, the size of Unsafe or the worst-case time bound.

*Spurious counter-examples.* Due to the abstraction induced by the non-deterministic variables, a CE $\gamma_{CE}$ can be spurious, i.e. it does not violate the property in the original system. Let $\boldsymbol{\eta} = \bar{\eta}_1, \ldots, \bar{\eta}_k$ be a sequence of valuations over $\bar{V}$, $\gamma \in \mathcal{V}(\Gamma)$, and $\rho(\gamma, \boldsymbol{\eta})$ be the path of $\mathcal{N}(\gamma)$ where the non-deterministic variables at $i$-th state are set to $\bar{\eta}_i$. Let $\boldsymbol{\eta}^*$ be the sequence of valuations describing the evolution of the original system. For a safety property $\varphi$, any CE $\gamma_{CE}$ generated by Alg. 1 is such that $\exists \boldsymbol{\eta}.\ \rho(\gamma_{CE}, \boldsymbol{\eta}) \in \Pi(\mathcal{N}(\gamma_{CE})) \wedge \rho(\gamma_{CE}, \boldsymbol{\eta}) \not\models^{\mathcal{N}(\gamma_{CE})} \varphi$, i.e. $\gamma_{CE}$ violates $\varphi$ for some valuations $\boldsymbol{\eta}$ of $\bar{V}$. The first term of the conjunction expresses that $\boldsymbol{\eta}$ is admissible, that is, $\rho(\gamma_{CE}, \boldsymbol{\eta})$ is a path of $\mathcal{N}(\gamma_{CE})$. Then, $\gamma_{CE}$ is spurious if $\rho(\gamma_{CE}, \boldsymbol{\eta}^*) \models^{\mathcal{N}(\gamma_{CE})} \varphi$.

*Counter-example generalisation.* The GeneralizeCE procedure is executed on top of the solver used in Alg. 1 and exploits the ability of SMT solvers to generate unsatisfiable cores, i.e., when a formula is unsatisfiable under the current assertions, produce a subset of its clauses whose conjunction is still unsatisfiable. The functioning of the algorithm is also illustrated in Fig. 5. Given a CE $\gamma_{CE}$, the idea is to derive a larger unsafe region $\gamma'_{CE}$ that contains $\gamma_{CE}$. This is achieved by asserting the safety property (line 3) and the valuation $\gamma_{CE}$ (line 4). In particular, we associate each assertion $p = \gamma_{CE}(p)$ for $p \in \Gamma$ (used to assert $\gamma_{CE}$) with a literal $g_p$. If formula $\bigwedge_{p\in\Gamma} g_p$ (line 5) is unsatisfiable, the solver returns an unsat core, i.e. a set UnsatCore $\subseteq \{g_p \mid p \in \Gamma\}$. If UnsatCore is a strict subset of the $g_p$ literals, we say that the generalization is successful since we obtain a larger region: $\gamma'_{CE} = \{\gamma \mid \gamma(p) = \gamma_{CE}(p) \text{ if } g_p \in \text{UnsatCore}\}$. Otherwise, $\gamma'_{CE} = \gamma_{CE}$. As an example, let $\gamma_{CE} = (p_1 = 3 \wedge p_2 = 5)$, and let $g_{p_1}$ and $g_{p_2}$ be the corresponding literals. If UnsatCore $= \{g_{p_2}\}$, then the generalization $\gamma'_{CE} = (p_2 = 5)$ strictly contains $\gamma_{CE}$. Importantly, $\bigwedge_{p\in\Gamma} g_p$ being unsatisfiable means that $\gamma_{CE}$ violates safety for any valuation of the non-deterministic variables and, therefore,

it is a CE also for the original system, i.e., it holds that

$$\forall \boldsymbol{\eta}.\ \rho(\gamma_{CE}, \boldsymbol{\eta}) \in \Pi(\mathcal{N}(\gamma_{CE})) \implies \rho(\gamma_{CE}, \boldsymbol{\eta}) \not\models^{\mathcal{N}(\gamma_{CE})} \varphi. \tag{5}$$

This applies also to its generalization $\gamma'_{CE}$.

In the implementation, we use a more advanced algorithm (Alg. 4) that can rule out even larger unsafe regions. Given a CE $\gamma_{CE}$, it works by guessing an intersection of half-spaces of the form $\gamma''_{CE} = \bigwedge_{p \in \Gamma} p \bowtie_p \gamma_{CE}$, where $\bowtie_p$ is chosen uniformly in $\{\geq, \leq\}$ (line 5). For each $p$, the literal $g_p$ is used to label the assertion $Gen(p)$ that describes the half-space associated to $p$. Now, $\bigwedge_{p \in \Gamma} g_p$ being unsatisfiable (line 8) means that safety is violated for any valuation in $\gamma''_{CE}$ (and, as above, for any valuation of non-deterministic variables). If this is the case, the generalisation is successful, since $\gamma''_{CE}$ strictly contains $\gamma_{CE}$ apart from isolated cases, e.g., when $\gamma_{CE}$ lies in one of the corner points of a rectangular parameter space and $\gamma''_{CE} = \gamma_{CE}$. Similarly to Alg. 1, the UNSAT core can be used to generalise $\gamma''_{CE}$ further, by dropping one or more half-space constraints (line 10). Using the above example, assume that $\gamma''_{CE} = (p_1 \geq 3 \wedge p_2 \geq 5)$ and that $\mathsf{UnsatCore} = \{g_{p_1}\}$. Then, $\gamma''_{CE}$ is generalised to $\gamma'_{CE} = p_1 \geq 3$. Note that, in general, the guess $\gamma''_{CE}$ has lower likelihood of being found unsatisfiable than the counter-example, $\gamma_{CE}$, itself, but this approach performed better in our experiments.

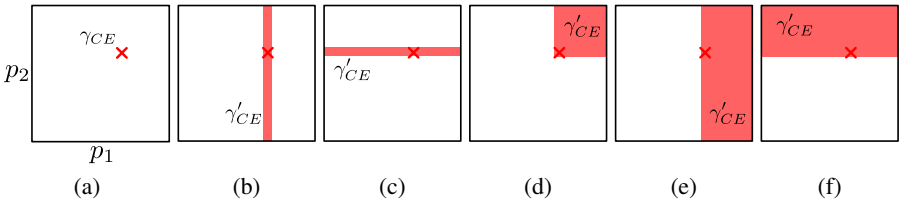



Fig. 5: Examples of counter-example generalization for a 2D parameter space. Given a CE $\gamma_{CE}$ (red cross), Alg. 2 can find a generalization (red area) $\gamma'_{CE} = (p_1 = \gamma_{CE}(p_1))$ (b) or $\gamma'_{CE} = (p_2 = \gamma_{CE}(p_2))$ (c). In Alg. 4, we guess a region $\gamma''_{CE} = (p_1 \geq \gamma_{CE}(p_1) \wedge p_2 \geq \gamma_{CE}(p_2))$ (d) and if unsatisfiable, we can generalise obtaining $\gamma'_{CE} = \gamma''_{CE}$, $\gamma'_{CE} = (p_1 \geq \gamma_{CE}(p_1))$ (e) or $\gamma'_{CE} = (p_2 \geq \gamma_{CE}(p_2))$ (f).

*Maximum robustness radius.* Procedure MaxRadius (Alg. 3) takes the current Unsafe region and the previous maximum radius $\epsilon$, and returns the updated maximum $\epsilon$ together with an $\epsilon$-robust valuation $\bar{\gamma}$, such that $B_\epsilon(\bar{\gamma}) \wedge \mathsf{Unsafe}$ is unsatisfiable. To this aim, we just need to find a valuation $\bar{\gamma}$ such that

$$B_\epsilon(\bar{\gamma}) \subseteq \mathcal{V}(\Gamma) \wedge \forall \gamma' \in B_\epsilon(\bar{\gamma}).\neg(\gamma' \wedge \mathsf{Unsafe}). \tag{6}$$

Procedure FindRobustParam (not shown) performs this check and returns such $\bar{\gamma}$ if any exists. In this case, we increment $\epsilon$ and repeat the procedure as long as an $\epsilon$-robust valuation is found. Otherwise, we decrement it and repeat the procedure until Eq. 6 is met. In our implementation, $\epsilon$ is discretized too. We remark that this procedure uses a separate SMT solver (SMT QBVF) so it can be efficiently parallelized. Since we consider parameters with bounded domains, $B_\epsilon(\bar{\gamma}) \subseteq \mathcal{V}(\Gamma)$ in Eq. 6 implies that $\epsilon$ is bounded too, and thus, that the algorithm terminates.

*Spurious robust valuations.* Since we do not exhaustively search for CEs, Unsafe is an under-approximation of the true unsafe set. For the same reason, there could be[3] *spurious $\epsilon$-robust valuations* $\gamma$ s.t. they meet Eq. 6, but CEs exist in $B_\epsilon(\gamma)$. This happens when Alg. 1 terminates without inspecting region $B_\epsilon(\gamma)$. The following proposition characterises when a valuation is in the solution space of the inner problem.

**Proposition 1 (Inner problem solution).** *Let $\gamma \in \mathcal{V}(\Gamma)$,* Unsafe *and $\epsilon$ be as returned by Alg. 1. Then, $\gamma$ is a solution of the inner problem in Probl. 1 iff it holds that:*

*i) $\gamma$ is $\epsilon$-robust w.r.t.* Unsafe*, i.e. it satisfies Eq. 6; and*
*ii) no counter-examples can be found in $B_\epsilon(\gamma)$.*

Note that ii) can be decided with one iteration of the CE generation loop in Alg. 1 within region $B_\epsilon(\gamma)$. Nevertheless, the algorithm guarantees that the returned $\epsilon$ is the maximum robust radius and that $\bar{\gamma}$ is a solution of the inner problem. Indeed, $\bar{\gamma}$ is computed by Alg. 3 and therefore meets Eq. 6. Further, no CEs exist in $B_\epsilon(\bar{\gamma})$ ($\bar{\gamma}$ is not spurious), otherwise the incremental synthesis algorithm could not exit the loop at lines 8-17 and would proceed by updating $\epsilon$ and $\bar{\gamma}$.

**Algorithm 1** Incremental Synthesis

**Require:** Parametric network $\mathcal{N}(\cdot)$, CMTL property $\square^{[\ell,u]}\phi$, path length $n \in \mathbb{N}^+$
**Ensure:** Maximum robust radius $\epsilon$, $\epsilon$-robust valuation $\bar{\gamma}$ and Unsafe region

1: **function** IncrementalSynth($\mathcal{N}(\cdot), \phi, n$)
2: $\quad\epsilon := 1$
3: $\quad$Unsafe $:= \bot$
4: $\quad\bar{\gamma} := \bot$
5: $\quad$Assert $Init(\rho[0])$
6: $\quad$**for** $k = 0, \ldots, n-1$ **do**
7: $\quad\quad$Assert $p_k : \square^{[\ell,\min(u,\rho\langle 0,k\rangle)]}\phi$
8: $\quad\quad$**repeat** ▷ CE generation cycle
9: $\quad\quad\quad(\epsilon, \bar{\gamma}) :=$ MaxRadius(Unsafe, $\epsilon$)
10: $\quad\quad\quad$Push
11: $\quad\quad\quad$Assert $B_\epsilon(\bar{\gamma})$
12: $\quad\quad\quad(SAT, \gamma_{CE}) :=$ Solve $\neg p_k$
13: $\quad\quad\quad$Pop
14: $\quad\quad\quad\gamma'_{CE} :=$ GeneralizeCE($\gamma_{CE}$)
15: $\quad\quad\quad$Assert $\neg\gamma'_{CE}$
16: $\quad\quad\quad$Unsafe $:=$ Unsafe $\vee\ \gamma'_{CE}$
17: $\quad\quad$**until** $SAT$
18: $\quad\quad$**if** $\epsilon < 1$ **then**
19: $\quad\quad\quad$**return** $(0, \bot, \top)$
20: $\quad\quad$**if** $k < n-1$ **then**
21: $\quad\quad\quad$Assert $T(\rho[k], \rho[k+1])$
22: $\quad$**return** $(\epsilon, \bar{\gamma},$ Unsafe$)$

[3] Not to be confused with the spurious counter-examples discussed before.

**Algorithm 2** Counter-example generalization

**Require:** Counter-example $\gamma_{CE}$
**Ensure:** Generalization $\gamma'_{CE}$ s.t. $\gamma_{CE} \implies \gamma'_{CE}$
1: **function** GeneralizeCE($\gamma_{CE}$)
2: Push
3: Assert $p_k$
4: **for all** $p \in \Gamma$ **do** Assert $g_p : p = \gamma_{CE}(p)$
5: $(SAT, \gamma) :=$ Solve $\bigwedge_{p \in \Gamma} g_p$
6: **if** $SAT$ **then** $\gamma'_{CE} := \gamma_{CE}$
7: **else** $\gamma'_{CE} := \bigwedge_{p.g_p \in \text{UnsatCore}} p = \gamma_{CE}(p)$
8: Pop
9: **return** $\gamma'_{CE}$

**Algorithm 3** Computation of maximum robust radius

**Require:** Unsafe region, starting radius $\epsilon$
**Ensure:** Maximum robust radius $\epsilon$ and valuation $\bar{\gamma}$ that is $\epsilon$-robust
1: **function** MaxRadius(Unsafe, $\epsilon$)
2: $\bar{\gamma} :=$ FindRobustParam(Unsafe, $\epsilon$, $\bot$)
3: **if** $\bar{\gamma} = \bot$ **then** $inc := -1$
4: **else** $inc := 1$
5: **repeat**
6: $\epsilon := \epsilon + inc$
7: $\bar{\gamma} :=$ FindRobustParam(Unsafe, $\epsilon$)
8: **until** $(inc < 0 \iff \bar{\gamma} = \bot) \wedge \epsilon > 0$
9: **if** $inc > 0$ **then** $\epsilon := \epsilon - inc$
10: **return** $(\epsilon, \bar{\gamma})$

**Algorithm 4** Counter-example generalization using half-spaces

**Require:** Counter-example $\gamma_{CE}$
**Ensure:** Generalization $\gamma'_{CE}$ s.t. $\gamma_{CE} \implies \gamma'_{CE}$
1: **function** GeneralizeCE($\gamma_{CE}, \bar{\gamma}$)
2: Push
3: Assert $p_k$
4: **for all** $p \in \Gamma$ **do**
5: $\bowtie = \mathcal{U}\{\leq, \geq\}$ ▷ Choose randomly a half-space to generalize
6: $Gen(p) := p \bowtie \gamma_{CE}(p)$
7: Assert $g_p : Gen(p)$
8: $(SAT, \gamma) :=$ Solve $\bigwedge_{p \in \Gamma} g_p$
9: **if** $SAT$ **then** $\gamma'_{CE} := \gamma_{CE}$
10: **else** $\gamma'_{CE} := \bigwedge_{p.g_p \in \text{UnsatCore}} Gen(p)$
11: Pop
12: **return** $\gamma'_{CE}$

*Encoding of counting formulas for incremental synthesis.* The safety property $p_k : \square^{[\ell, \min(u, \rho\langle 0,k\rangle)]}\phi$ at step $k$ (see Alg. 1, line 7) is encoded from previous assertions as

$p_k : p_{k-1} \wedge (\ell \leq \rho\langle 0, k\rangle \leq u \implies \phi)$. The encoding of $\phi$ when $\phi \in \mathcal{E}(V)$ is straightforward. Here we show the encoding of counting formulas.

We recall the semantics of a Basic Counting Formula (BCF) $\sum_{j\in J} c_j \#_{\ell_j}^{u_j} e_j \bowtie b$:

$$(\rho, i) \models^{\mathcal{N}} \sum_{j\in J} c_j \#_{\ell_j}^{u_j} e_j \bowtie b \quad \text{iff} \quad \left(\sum_{j\in J} c_j \sum_{k=\rho^{[i,|\rho|]}@\ell_j}^{\rho^{[i,|\rho|]}@u_j-1} \mathbf{1}\left(\eta_k \models e_j\right)\right) \bowtie b$$

where $\mathbf{1}\left(\eta_k \models e_j\right)$ is the characteristic function that return 1 if $\eta_k \models e_j$ and 0, otherwise. Note that the right-hand side of the above can be rewritten as

$$\left(\sum_{j\in J} c_j \sum_{k=i}^{|\rho|-1} \mathbf{1}\left((\rho\langle i, k\rangle - \rho\langle i\rangle) \in (\ell_j, u_j] \wedge \eta_k \models e_j\right)\right) \bowtie b$$

i.e. we "count" the satisfaction of $\eta_k \models e_j$ only if, starting from $i$, the time spent up to the $k$-th step is in within the bounds of the counting formula: $(\rho\langle i, k\rangle - \rho\langle i\rangle) \in (\ell_j, u_j]$.

In addition, we need to account for the fact that, in the incremental synthesis algorithm, the length of the path unrolled at a generic step is not enough to decide the counting formula, because it does not cover the upper time bound $u_j$. This implies that in the counter-example generation phase, the solver can find interpretations for the variables of the path positions not covered (and thus, not yet constrained by the transition relation), such that the negated property holds, thus giving wrong counter-examples. Let $t_{unroll}$ be the variable that tells the total time covered at a generic iteration $it$ of the algorithm, i.e. $t_{unroll} = \rho\langle 0, it\rangle$. Then, the following predicate characterises if the path length is enough to decide the formula at position $i$:

$$\mathsf{sufficientPath} \iff \bigwedge_{j\in J} t_{unroll} \geq \rho\langle i\rangle + u_j$$

In order to avoid the above problem, we just need to make the counting formula trivially true when the current path length is insufficient. This leads to:

$$\mathsf{sufficientPath} \implies \left(\sum_{j\in J} c_j \sum_{k=i}^{|\rho|-1} \mathbf{1}\left((\rho\langle i, k\rangle - \rho\langle i\rangle) \in (\ell_j, u_j] \wedge \eta_k \models e_j\right)\right) \bowtie b$$

*Incremental synthesis for CMTL reachability fragment.* In Alg. 1, we presented the incremental synthesis method for the safety fragment of CMTL, based on finding counter-examples to a safety property. Here we illustrate the incremental synthesis algorithm for reachability CMTL formulas, i.e. of the form $\Diamond^{[\ell,u]}\phi$, where $\phi$ is a CMTL formula without temporal operators. This is presented in Alg. 5. The structure is the same as its safety counterpart but, in this case, a counter-example (CE) to $\Diamond^{[\ell,u]}\phi$ is a parameter valuation s.t. $\phi$ *never* holds (or $\neg\phi$ always holds). Due to the incremental nature of the algorithm, a CE can be determined at a generic path position $k$ if either:

– the total time at $k$ exceeds the upper time bound of the formula, i.e. $\rho\langle 0, k\rangle > u$, and $\phi$ never holds in the time interval $[\ell, u]$. This is encoded in the algorithm by literal $UB_k$ (line 11).

– $k$ is the final path position ($k = n - 1$) and $\phi$ never holds up to $k$. In this case, a CE is reported even if the total time does not exceed the upper bound $u$ in the formula. Therefore, we ignore the fact that $\phi$ might be reached eventually at longer path lengths, because we focus on satisfying the reachability specification on bounded paths.
– $\rho\langle 0, k\rangle \leq u$ and $k < n - 1$ (i.e. $k$ is not the final position and does not exceed $u$), but there exist a $k, i$-loop for some $i$ along which $\phi$ never holds. For the sake of simplicity, we assume that the satisfaction value of $\phi$ never changes across successive loop iterations (e.g. when $\phi$ is a predicate from $\mathcal{E}(V)$). In this way, to check for a CE it suffices to check that, up to position $k$, $\phi$ is never reached and there exists a loop. In the algorithm, this loop checking is implemented through the formula at line 9. If $\phi$ is instead a counting formula, its satisfaction value could change across loop iterations and thus additional checks are needed (not shown here) to determine whether $\phi$ eventually holds along the loop unrolling.

Similarly to the safety algorithm, the property $p_k : \Diamond^{[\ell,\min(u,\rho\langle 0,k\rangle)]}\phi$ at step $k$ is encoded from previous assertions as $p_k : p_{k-1} \vee (\ell \leq \rho\langle 0, k\rangle \leq u \implies \phi)$.

**Algorithm 5** Incremental Synthesis for reachability

**Require:** Parametric network $\mathcal{N}(\cdot)$, CMTL property $\Diamond^{[\ell,u]}\phi$, path length $n \in \mathbb{N}^+$
**Ensure:** Maximum robust radius $\epsilon$, $\epsilon$-robust valuation $\bar{\gamma}$ and Unreach region

1: **function** IncrementalSynth($\mathcal{N}(\cdot), \phi, n$)
2: $\epsilon := 1$
3: Unreach $:= \bot$
4: $\bar{\gamma} := \bot$
5: Assert $Init(\rho[0])$
6: Assert $T(\rho[0], \rho[1])$
7: **for** $k = 1, \ldots, n-1$ **do**
8: Assert $p_k : \Diamond^{[\ell,\min(u,\rho\langle 0,k\rangle)]}\phi$
9: **if** $k < n - 1$ **then** Assert $loop_k : \bigvee_{k'=0}^{k-1} \rho[k'] = \rho[k]$ ▷ Loop checking literal
10: **else** Assert $loop_k : \top$
11: Assert $UB_k : \rho\langle 0, k\rangle > u$ ▷ Upper bound literal
12: **repeat** ▷ CE generation cycle
13: $(\epsilon, \bar{\gamma}) :=$ MaxRadius(Unreach, $\epsilon$)
14: Push
15: Assert $B_\epsilon(\bar{\gamma})$
16: $(SAT, \gamma_{CE}) :=$ Solve $\neg p_k \wedge (UB_k \vee loop_k)$
17: Pop
18: $\gamma'_{CE} :=$ GeneralizeCE($\gamma_{CE}$)
19: Assert $\neg\gamma'_{CE}$
20: Unreach $:=$ Unreach $\vee\ \gamma'_{CE}$
21: **until** $SAT$
22: **if** $\epsilon < 1$ **then**
23: **return** $(0, \bot, \top)$
24: **if** $k < n - 1$ **then**
25: Assert $T(\rho[k], \rho[k+1])$
26: **return** $(\epsilon, \bar{\gamma}$, Unreach$)$

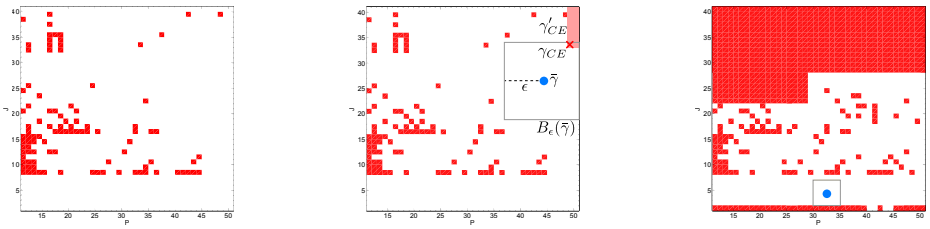
*Running example.* To simplify the presentation, we fix $T = 10$ and consider only parameters $J \in [1, 41]$ and $P \in [11, 51]$. Fig. 6 shows the incremental synthesis algorithm run on our example. The counter-example $J = 33$ and $P = 49$ indicated in plot (b) clearly violates the property (Eq. 1), since it gives the following path:

$$((q, z), (\alpha = 0, \beta = 0, t = 0, x = 0, y = 0, act = -1)) \xrightarrow{33} \\ ((q', z), (\alpha = 0, \beta = 5, t = 0, x = 33, y = 0, act = 1)) \dots$$

where, starting from position 0, no VP action is fired in the time interval $[0, 7]$. At the final step, we obtain $\epsilon = 2$ and $\bar{\gamma} = \{J \mapsto 4, P \mapsto 32\}$. Such parameters lead to the following path which can be shown to meet our CMTL property:

$$((q, z), (0, 0, 0, 0, 0, -1)) \xrightarrow{4} ((q', z), (0, 5, 0, 4, 0, 1)) \xrightarrow{4} ((q', z), (0, 5, 0, 8, 0, 1)) \xrightarrow{4} \\ ((q', z), (0, 5, 0, 12, 0, 1)) \xrightarrow{4} ((q', z), (0, 5, 0, 16, 0, 1)) \xrightarrow{4} ((q', z), (0, 5, 0, 20, 0, 1)) \xrightarrow{4} \\ ((q', z), (0, 5, 0, 24, 0, 1)) \xrightarrow{4} ((q', z), (0, 5, 0, 28, 0, 1)) \xrightarrow{4} ((q', z), (0, 5, 0, 32, 0, 1)) \xrightarrow{0} \\ ((q, z), (10, 5, 0, 0, 0, 0)) \xrightarrow{4} ((q', z), (10, 5, 0, 4, 0, 1)) \xrightarrow{4} ((q', z), (10, 5, 0, 8, 0, 1)) \xrightarrow{4} \\ ((q', z), (10, 5, 0, 12, 0, 1)) \xrightarrow{4} ((q', z), (10, 5, 0, 16, 0, 1)) \xrightarrow{4} ((q', z), (10, 5, 0, 20, 0, 1))$$

In the above, variable names are omitted and their ordering is as in the previous path. The validity of Eq. 1 can be shown for every valuation in $B_\epsilon(\bar{\gamma})$ in a similar way.




(a) (b) (c)

Fig. 6: Counter-examples generation cycle for the running example. Plot (a) shows the Unsafe region (red points) during step $k = 1$. Procedure MaxRadius computes the maximum $\epsilon$ (here, 8) and the $\epsilon$-robust valuation $\bar{\gamma}$ ($J = 25, P = 43$, blue dot in plot b). The search for further CEs is restricted to $B_\epsilon(\bar{\gamma})$ (grey-bordered). Then, a CE $\gamma_{CE}$ is found ($J = 33, P = 49$, red cross). The GeneralizeCE procedure manages to find the larger unsafe region $\gamma'_{CE} = J \geq 33 \wedge P \geq 49$ (light red). Plot (c) shows the results at the final step ($k = 15$) with $\epsilon = 2$ and $\bar{\gamma} = \{J \mapsto 4, P \mapsto 32\}$.

### 5.3 The outer problem

We present two methods for solving the outer problem. The former is based on the enumeration of $\epsilon$-robust valuations, thus providing an exact solution to the outer problem, but is infeasible with high-dimensional parameter spaces. Note that enumeration is possible because we discretise the parameter space. The latter method provides an

approximate solution by exploiting evolutionary strategies (ES). In both methods, the outer objective is evaluated through simulation. Importantly, simulation can cover path lengths that are prohibitive for BMC, which allows us to consider objective functions over large time bounds.

*Exact solution.* The method consists of the following three steps:

1. Enumerate all valuations that meet Eq. 6. Let $\Gamma'$ be the set of such valuations.
2. Simulate all $\gamma \in \Gamma'$ and compute the outer objective $f(\gamma)$.
3. Following the ordering by the cost function $f(\gamma)$, return the first valuation that meets condition ii) of Prop. 1.

*Optimisation with Evolutionary Strategies.* ES are a class of stochastic optimisation methods which mimic the principles of Darwinian evolution in order to optimise a given objective. They work on a set of candidate solutions, the *population*, which at each iteration of the algorithm (*generation*) is subjected to various *natural operators*, until a pre-defined termination criterion is satisfied (e.g. max number of generations).

The procedure for solving the outer problem through ES is shown in Algorithm 6. We implement a *non-isotropic self-adaptive* $(\mu/\rho + \lambda)$-ES, i.e. $\mu$ parents are used to generate $\lambda$ offspring candidates through a $\rho$-parents recombination, and only the $\mu$ best solutions of the combined parents together with the offspring set are used in the next generation. In particular, we consider a 2-*parents dominant recombination*, which randomly takes two candidates from the parents set (line 5 of Algorithm 6) and generates a child by a parameter-wise random selection of the two parents' valuations (line 6). Since we deal with discrete parameters, we use the mutation operator in [34], which extends the principle of *maximum entropy* used in real ES problems to the integer case. We also include a *self-adaptation* mechanism [6] that changes the parameters of the mutation operator at each iteration.

Let $p \in \Gamma$ and $n = |\Gamma|$. We randomly generate an additive mutation $z_p$ from the distribution (line 8):

$$P\{Z = k\} = \frac{\bar{\sigma_p}}{2 - \bar{\sigma_p}} \left(1 - \bar{\sigma_p}\right)^{|k|}, \quad \bar{\sigma_p} = 1 - \frac{\sigma_p / n}{\left(1 + \left(\sigma_p / n\right)^2\right)^{1/2} + 1} \tag{7}$$

where $\sigma_p$ is a parameter controlled by the *self-adaptive* part of the algorithm, called *mean step*. This is mutated at each iteration according to the *log-normal* rule [6] (line 7):

$$\sigma_p = \exp\left(\tau_0 \mathcal{N}(0,1)\right) \cdot \sigma_p \cdot \exp\left(\tau \mathcal{N}(0,1)\right) \tag{8}$$

where $\mathcal{N}(0,1)$ are realizations of a standardized Gaussian process, and $\tau_0$ and $\tau$ are the *learning parameters*, which we set to $\tau_0 = \frac{1}{\sqrt{2n}}$ and $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$ as in [36]. The initial value of $\sigma_p$ is typically model-dependent and can be tuned to improve the performance of the algorithm. In order to avoid premature stagnation of the algorithm, we reset $\sigma_p$ to its initial value with a *small* probability [35].

In order to determine the best valuations at each generation, we define an order $\preceq$ that takes into account the outer objective and, following the *feasible-over-infeasible* principle [14], *penalizes valuations outside the solution space of the inner problem*. Let

$\gamma_i$ and $\gamma_j$ be two valuations, $f(\gamma_i)$ and $f(\gamma_j)$ be their objective function values. Then, $\gamma_i \preceq \gamma_j$ if either:

1. $\gamma_i$ meets condition i) of Prop. 1, and $\gamma_j$ does not; or
2. $\gamma_i$ meets i) and ii), and $\gamma_j$ meets only i); or
3. both $\gamma_i$ and $\gamma_j$ meet i) and ii), and $f(\gamma_i) \leq f(\gamma_j)$

**Algorithm 6** Evolutionary Strategy for solving the outer problem

**Require:** Ordering relation $\preceq$, maximum number of generations $gen_{\max}$
**Ensure:** Solution to the outer problem $\gamma_o$

1: **function** $(\mu/\rho + \lambda)$-ES($\preceq, gen_{\max}$)
2: $\quad\mathcal{P}^{(0)} := RandomInitialize(\mu)$ ▷ generates $\mu$ random valuations
3: $\quad$**for** $gen = 1, \ldots, gen_{\max}$ **do**
4: $\quad\quad$**for** $i = 1, \ldots, \lambda$ **do** ▷ $\gamma_i$ is the $i$-th valuation in the offspring set
5: $\quad\quad\quad(\gamma_{p_1}, \ldots, \gamma_{p_\rho}) \leftarrow RandomSelection\left(\mathcal{P}^{(gen)}\right)$ ▷ selects $\rho$ parents
6: $\quad\quad\quad\gamma_i := Recombination(\gamma_{p_1}, \ldots, \gamma_{p_\rho})$
7: $\quad\quad\quad\gamma_i := MutateStrategyParameters\,(\gamma_i)$ ▷ Eq. 8
8: $\quad\quad\quad\gamma_i := MutateObjectParameters\,(\gamma_i)$ ▷ Eq. 7
9: $\quad\quad\mathcal{P}^{(gen+1)} := \left(\mathcal{P}^{(gen)} \cup \{\gamma_i | i = 1, \ldots, \lambda\}, \preceq, \mu\right)$ ▷ best $\mu$ valuations w.r.t. $\preceq$.
10: $\quad\gamma_o := \sup(\mathcal{P}^{(gen+1)}, \preceq)$ ▷ $\gamma_o$ is the best valuation
11: $\quad$**return** $\gamma_o$

We say that a solution is feasible for the outer problem if it solves the inner problem as per Prop. 1. Note that, if the population at a generic iteration $i$, $\mathcal{P}^i$, contains feasible solutions, then, for any $j > i$, $\mathcal{P}^j$ will contain feasible solutions too. Indeed, by the order defined, if $\mathcal{P}^i$ has at least one feasible point, then the best solution in $\mathcal{P}^i$ is also feasible. Since, for any $k$, the best solutions of $\mathcal{P}^k$ are kept in $\mathcal{P}^{k+1}$ (see line 9 of the algorithm), we conclude that, for $j > i$, $\mathcal{P}^j$ will contain feasible solutions too.

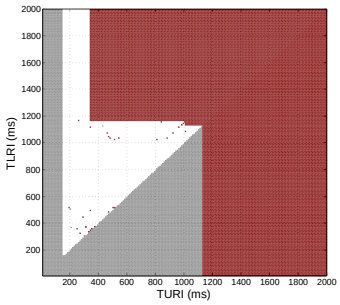
*Running example.* We obtain the exact solution $J = 4, P = 48$, which gives an outer objective of 2 (the number of AS actions fired within time 100). Due to the size of the problem, this required enumerating and simulating only 133 valuations at step 1 of the exact method. For the same reason, the ES algorithm is also able to achieve the optimal solution, being in this case $J = 4, P = 45$. In particular, this was obtained at the *first* iteration of the algorithm, run with $\lambda = 100$, $\mu = 50$ and $\rho = 2$.

## 6 Results

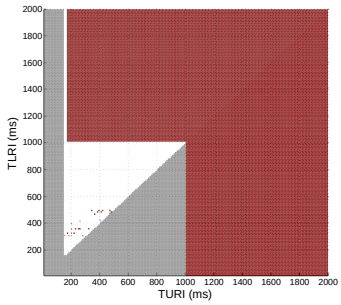
We apply our method to synthesise pacemaker parameters that ensure a safe heart rhythm and optimise either energy consumption or cardiac output (see Sect. 4 for the formulation of the problem and properties). We consider two parameters that are critical for the correct functioning of the pacemaker device. The first parameter, TLRI, regulates the frequency of atrial impulses: TLRI − TAVI is the amount of time that the pacemaker waits before delivering an atrial pace when no atrial or ventricular events are detected, where TAVI is the pacemaker atrioventricular delay (default value: 150 ms). The second parameter, TURI, sets an upper bound on the heart rate. In particular, it is the amount of time that the pacemaker waits before pacing the ventricle, after an

atrial stimulus has occurred and TAVI elapsed. We set the domain of both parameters to $[10, 2000]$ ms, and add constraints to exclude from the search pacemaker parameters that are not physiologically meaningful: we require $\mathrm{TLRI} \geq \mathrm{TURI}$, $\mathrm{TLRI} > \mathrm{TAVI}$ and $\mathrm{TURI} \geq \mathrm{TAVI}$. Note that the approach can be applied also to other pacemaker parameters: TAVI, TVRP (ventricular refractory period), TPVARP (post-ventricular atrial refractory period) and TPVABP (post-ventricular atrial blanking period).

Fig. 7 summarizes the synthesis results obtained with the following heart conditions: *bradycardia*, i.e. slow heart rate, reproduced through an increased SA node firing rate ($\mathrm{SA\_d} = 1500$ ms, i.e. 40 BPM), and the *AV conduction defect* obtained by increasing the AV delay ($\mathrm{AVD}_{\mathrm{min}} = 150$ ms, default: 50 ms). In the experiments we consider a path length of 20 for solving the inner problem and solve the outer problem with both exact and ES methods.




(a) Bradycardia. Inner problem solution time: 7354 s, $\epsilon = 250$ ms.



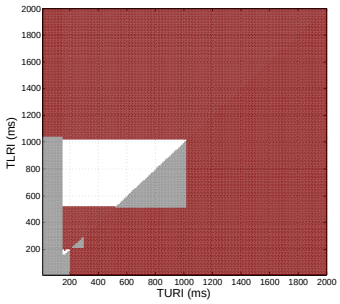

(b) AV defect. Inner problem solution time: 6601 s, $\epsilon = 240$ ms.

| **Outer objective:** | Energy | | | | Cardiac Output | | | |
|---|---|---|---|---|---|---|---|---|
| **Condition:** | Bradycardia | | AV defect | | Bradycardia | | AV defect | |
| **Method:** | Exact | ES | Exact | ES | Exact | ES | Exact | ES |
| **Best:** | 770,300 | 770,640 | 750,480 | 750,320 | 770,320 | 770,320 | 750,630 | 750,350 |
| **Cost:** | 158 | 158 | 400 | 400 | 9.14 | 9.14 | 9.37 | 9.37 |
| **Runtime:** | 2369 | 1101 | 913 | 1268 | 1547 | 118 | 848 | 111 |

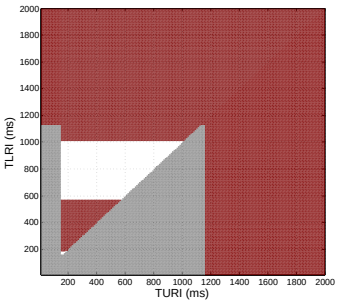
Fig. 7: Unsafe regions (red areas and dots) returned by Alg. 1 in the two experiments. Grey areas indicate pacemaker parameters that are not physiologically relevant and thus are excluded from the search space. The table shows the results of the outer optimisation for the energy and cardiac output objectives (see Sect. 4), comparing the exact and the ES-based methods. The best solutions are in the format TLRI, TURI. Runtimes are in seconds. ES parameters are $\lambda = 100$, $\mu = 50$, $\rho = 2$ and 50 generations.

The two experiments return similar robustness radii: $\epsilon = 240$ for bradycardia and $\epsilon = 250$ for AV defect. In the bradycardia case, we obtain $\mathrm{TLRI} = 770$ ms, i.e. a pacing rate in the atrium of 77.92 BPM, for all objectives and solution methods for the outer problem. In the AV defect case, the synthesis experiments yield a similar TLRI value (750 ms, i.e. 80 BPM) and optimal cardiac output. However, the energy consumption of the pacemaker is much higher since, with this heart condition, impulses from the atrium are not correctly propagated, and thus a higher number of paces is

required in the ventricle. We remark that, with our method, we are able to find the parameters that guarantee a safe heart rhythm despite large perturbations. For instance, the exact solution $\gamma_o$ to the AV defect and energy experiment is $\mathrm{TLRI} = 750$ ms and $\mathrm{TURI} = 480$ ms, which implies that safety holds for all the parameters in $B_\epsilon(\gamma_o)$, i.e., with $\epsilon = 250$, for all $\mathrm{TLRI} \in [500, 1000]$ ms and $\mathrm{TURI} \in [230, 730]$ ms. For all our results, we observe that the nominal parameter values ($\mathrm{TLRI} = 1000$ ms and $\mathrm{TURI} = 500$ ms [37]) are included in $B_\epsilon(\gamma_o)$, meaning that the default pacemaker settings are safe but have a smaller tolerance.




(a) Full Unsafe region for bradycardia




(b) Full Unsafe region for multiple heart parameters

Fig. 8: (a) Full synthesis regiosn for bradycardia (see experiment at Fig. 7 a) and (b) for multiple SA rates and AV delays: $\mathrm{SA\_d} \in [1250, 1500]$ ms and $\mathrm{AVD}_{\min} \in [0, 200]$ ms.

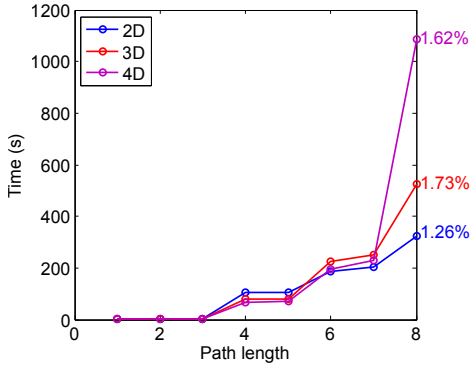
Notably, the evolutionary approach is able to yield the same optimal objective value as the exact method. This is due to the fact that, with two parameters, the solution space of the inner problem (which corresponds to the domain of the outer problem) is quite small. Indeed, we obtain only $107$ $\epsilon$-robust valuations for bradycardia, and $52$ for the AV defect. With the ES algorithm, we also achieve better performance in most cases, and the runtime improvement becomes even more marked with higher-dimensional parameter spaces, as shown at the end of this section. The only exception is the runtime obtained for the energy objective in the AV defect experiment, where the exact method performs slightly better than ES, which is explained by the small number of feasible points in the outer problem.

In Fig. 8a, we illustrate the full synthesis region for the bradycardia experiment, obtained without restricting the search space for CEs (lines 9-13 of Alg. 1). By comparing with the region in Fig. 7a, we observe that the algorithm explores considerably fewer CEs, thus improving the runtime. We also report that the abstraction of real-valued and non-linear variables is adequate, in the sense that only a few CEs are *potentially* spurious, i.e. such that Eq. 5 does not hold. These constitute only $0.31\%$ of the parameter space and are given by the set

$$\{\gamma \in \mathcal{V}(\Gamma) \mid (\gamma(\mathrm{TLRI}) = 510 \wedge \gamma(\mathrm{TURI}) \in [150, 510]) \vee (\gamma(\mathrm{TLRI}) = 1020 \wedge \gamma(\mathrm{TURI}) \in [150, 1010])\}$$

*Synthesis for ranges of heart conditions.* With our approach, we can also synthesise parameters that are safe for a range of possible heart conditions, thus taking into account uncertainty in the heart dynamics. In Sect. 5.1, we extended TIOAs with non-deterministic variables in order to bound real-valued and non-linear functions. Non-deterministic variables can also be used to express uncertainty in the model's dynamics, so that we can synthesise parameters that are safe *for all* the admissible valuations of such variables. In this way, we can conveniently approximate also probabilistic heart parameters with non-deterministic variables that range within an adequate truncation of the corresponding probability distribution, e.g. based on percentile rank. Or, we can express time tolerance for a range of pacemaker parameters. For instance, we can synthesise parameters that are safe for multiple bradycardia SA node firing rates and AV delays, as shown in Fig. 8b.

*Performance evaluation* We analyse the performance of the synthesis algorithm considering a 2D (TLRI and TURI), a 3D (TLRI, TURI and TAVI) and a 4D (TLRI, TURI, TAVI and TPVARP) parameter space. Experiments were performed with a path length of 8 for the inner problem and with the bradycardia parameters (see Sect. 6), and are reported in Fig. 9.




| | Exact | ES |
|---|---|---|
| 2D | 117 | 69 |
| 3D | 12697 | 123 |
| 4D | >20000 | 3134 |

Fig. 9: Synthesis algorithm performance for different numbers of parameters. The figure shows the incremental runtimes (seconds) for the inner problem. Percentanges indicate the fraction of time taken by Alg. 3. The table compares the computation time (seconds) of the exact method and the evolutionary strategy in solving the outer problem, performed at path length 8. The time for the exact solution in the 4D case exceeded the timeout of 20000 seconds. ES parameters are $\lambda = 100$, $\mu = 50$, $\rho = 2$ and 50 generations.

## 7 Conclusion and Future Work

We have studied the problem of robust optimal parameter synthesis for networks of TIOAs with priorities and data and proposed a solution based on SMT solving and evolutionary strategies. We have applied the method to synthesise pacemaker parameters that are both safe and robust, while optimising energy consumption or cardiac output. As the main property specification language, we have considered the safety and reacha-

bility fragments of CMTL, which are sufficient to express relevant properties for cardiac pacemakers.

As future work we plan to apply the approach to additional safety properties, validate synthesis results with cardiologists and include advanced pacemaker features like rate-modulation [29], hysteresis and a battery model for optimising the expected lifetime of the device.

**Acknowledgments.** This work is supported by the ERC AdG VERIWARE and ERC PoC VERIPACE.

## References

1. 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.
2. É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP*, pages 76–90. Springer, 2010.
3. T. Arai, K. Lee, and R. J. Cohen. Cardiac output and stroke volume estimation using a hybrid of three windkessel models. In *EMBC*, pages 4971–4974. IEEE, 2010.
4. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *STTT*, 11(1):69–83, 2009.
5. B. Barbot, M. Kwiatkowska, A. Mereacre, and N. Paoletti. Estimation and verification of hybrid heart models for personalised medical and wearable devices. In *13th International Conference on Computational Methods in Systems Biology (CMSB 2015)*, volume 9308 of *LNCS*, pages 3–7. Springer, 2015.
6. H.-G. Beyer and H.-P. Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.
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. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative verification of implantable cardiac pacemakers over hybrid heart models. *Information and Computation*, 236:87–101, 2014.
10. A. Cimatti, S. Mover, and S. Tonetta. SMT-Based Verification of Hybrid Systems. In J. Hoffmann and B. Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.
11. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.
12. B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.
13. L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
14. K. Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2):311–338, 2000.
15. M. Diciolla, C. H. P. Kim, M. Kwiatkowska, and A. Mereacre. Synthesising Optimal Timing Delays for Timed I/O Automata. In *EMSOFT'14*. ACM, 2014.
16. L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.

17. N. Fazeli and J.-O. Hahn. Estimation of cardiac output and peripheral resistance using square-wave-approximated aortic flow signal. *Frontiers in physiology*, 3, 2012.
18. S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo ODEs. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 105–112. IEEE, 2013.
19. A. O. Gomes and M. V. M. Oliveira. Formal specification of a cardiac pacing system. In *FM 2009: Formal Methods*, pages 692–707. Springer, 2009.
20. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification*, pages 190–203. Springer, 2008.
21. Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–203. Springer, 2012.
22. A. Jovanović and M. Kwiatkowska. Parameter synthesis for probabilistic timed automata using stochastic games. In *RP'14*, volume 8762 of *LNCS*, pages 176–189, 2014.
23. A. Jovanović, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. In *TACAS*, pages 401–415. Springer, 2013.
24. D. R. Kerner. Solving windkessel models with mlab. http://www.civilized.com/mlabexamples/windkesmodel.htmld, 2007.
25. R. Kindermann, T. Junttila, and I. Niemelä. Beyond lassos: Complete SMT-based bounded model checking for timed automata. In *Formal Techniques for Distributed Systems*, pages 84–100. Springer, 2012.
26. R. Kindermann, T. Junttila, and I. Niemelä. SMT-based induction methods for timed systems. In *Formal Modeling and Analysis of Timed Systems*, pages 171–187. Springer, 2012.
27. 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.
28. G. Kovásznai, A. Fröhlich, and A. Biere. On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In *SMT*, pages 44–56, 2012.
29. M. Kwiatkowska, H. Lea-Banks, A. Mereacre, and N. Paoletti. Formal modelling and validation of rate-adaptive pacemakers. In *ICHI*, pages 23–32. IEEE, 2014.
30. J. Lian, H. Krätschmer, D. Müssig, and L. Stotts. Open source modeling of heart rhythm and cardiac pacing. *Open Pacing Electrophysiol Ther J*, 3:4, 2010.
31. D. Méry and N. K. Singh. Closed-loop modeling of cardiac pacemaker and heart. In *Foundations of Health Information Engineering and Systems*, pages 151–166. Springer, 2013.
32. J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 188–197. IEEE, 2005.
33. A. Rabinovich. Complexity of metric temporal logics with counting and the pnueli modalities. *Theoretical Computer Science*, 411(22):2331–2342, 2010.
34. G. Rudolph. An evolutionary algorithm for integer programming. In *Parallel Problem Solving from Nature—PPSN III*, pages 139–148. Springer, 1994.
35. G. Rudolph. Self-adaptive mutations may lead to premature convergence. *Evolutionary Computation, IEEE Transactions on*, 5(4):410–414, 2001.
36. H.-P. P. Schwefel. *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., 1993.
37. B. Scientific. Pacemaker system specification. *Boston Scientific*, 2007.
38. T. Sturm and A. Tiwari. Verification and synthesis using real quantifier elimination. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation*, pages 329–336. ACM, 2011.
39. L.-M. Traonouez. A parametric counterexample refinement approach for robust timed specifications. *arXiv preprint arXiv:1207.4269*, 2012.