Abstraction-tree-guided Closed-loop Model Checking of Medical Devices

Zhihao Jiang[1], Houssam Abbas[1], Pieter Mosterman[2], and Rahul Mangharam[1]

[1] Department of Electrical Engineering, University of Pennsylvania
[2] Mathworks. Inc

Abstract. Implantable medical devices are designed to diagnose and improve certain adverse physiological conditions, thus the safety and efficacy of the devices have to be evaluated in closed-loop within their physiological context. Model-based design has enabled closed-loop verification early in the design stage and closed-loop model checking has been proposed to provide confidence to the safety and efficacy of the devices. The biggest challenge for closed-loop model-checking of medical devices is modeling the physiological environment. Unlike system modeling in which there is only one concrete system, there are countless number of physiological conditions and different models are required to distinguish behaviors associated with each condition. Over-approximation can be used to not only reduce model complexity, but also cover physiological-relevant behaviors from multiple physiological conditions in an abstract model. However, over-approximation inevitably introduces behaviors that are not physiologically possible, potentially causing false-negatives during model checking. Moreover, by abstracting multiple physiological conditions, behaviors from different physiological conditions become indistinguishable in over-approximated models, causing ambiguities to physicians who are responsible to determine the validity of potential counter-examples. In this paper we propose an abstraction-tree-based framework for closed-loop model checking of medical devices, and use implantable pacemaker as an example. A set of physiological abstraction rules are developed to merge behaviors from models representing different physiological conditions, and ensure majority of behaviors added to the abstract models are also physiological-relevant. By applying the physiological abstraction rules, an abstraction tree is constructed and can be used for closed-loop model checking. An automated search algorithm first select the most abstract models that are appropriate for the physiological requirements as the initial physiological models. In case of property violations, the search algorithm explore the abstraction tree for the most refined models which can still produce property violations, which branches an abstract property violation into possible physiological conditions thus provides helpful physiological context to the physicians. With this framework, a person with expertise in formal method do not need physiological knowledge to use the abstraction tree for model checking, and model checking results can be feedback to the physicians with physiological context. The framework can also be extended to other Cyber Physical System domains to bridge the gap between the cyber domain and the physical domain.

1 Introduction

Implantable medical devices like pacemakers are designed to improve certain undesired physiological conditions with very little human interventions. Their ability of autonomously affect the physiological conditions of the patients makes the medical devices safety-critical, and sufficient evidence on the safety and efficacy of the devices should be provided before the devices can be implanted in the patients. As more functions added to the devices, the complexity of the software component of the device is increasing dramatically, leading to increasing number of potential safety violations due to software bugs. In what follows, we use the word 'device' to refer both to the hardware and the software of the device.

give recall statistics

There are two categories of device bugs: first, the device may fail to conform to its *specification*, i.e. the prescription of how it should react to certain inputs. Secondly, even if it conforms to its specification, the device may fail to improve the health of the patient as promised. The improved state of health is captured in the *physiological requirements*; e.g., for a pacemaker, the heart rate should always be below a certain threshold. Note that requirements are about *the closed-loop system*: they prescribe the behavior of both device and environment (e.g. both pacemaker and heart).

Bugs in the first category (non-conformance to specification) are detected via extensive open-loop testing where a set of input sequences is fed to the device, and its output is observed to see if it matches the expected output. Bugs in the second category (violation of physiological requirements), on the other hand, require the availability of the closed-loop system: e.g., the heart and the pacemaker. In the medical device industry, clinical trials amount to a closed-loop test of the device and its software. In a clinical trial, the actual device is implanted in a human subject and its operation tested over a certain duration. Unfortunately, clinical trials can only cover a very limited range of physiological conditions due to their extremely high cost and long duration. Moreover, clinical trials are often conducted at the last design stage. Fixing bugs at this stage is also very costly.

give dollar amount and duration

On the other hand, using a virtual model of the device's environment, we can conduct *model-based clinical trials*: i.e., the device (or a model thereof) is connected to a virtual model of the environment and this virtual closed loop is verified. Depending on the formalism used to model the environment (and device) and the language used to express the requirements, this allows the usage of formal methods to perform the verification. Environment modeling introduces new challenges and issues that do not arise when modeling the device alone, and this paper aims at addressing these issues in the context of pacemaker verification. The proposed method is also generalizable to other contexts where closed-loop formal verification is desired. In the rest of this paper, we speak therefore of pacemaker as the Device Under Verification (DUV) and of the heart as being the environment, but it is understood that the discussion carries more broadly, with possible domain-specific adjustments.

The first challenge in closed-loop verification of pacemakers is that the human heart displays a large number of different conditions, henceforth referred to as 'physiological conditions'. [3] E.g., one heart may display *atrial fibrillation* where the upper chambers of the heart (the atria) produce an exceedingly fast beat that prevents proper blood pumping. Another heart may display Premature Ventricular Contraction (PVC) where a location in the ventricles produces electrical impulses at erratic time instants. Each such condition will require its own formal model, and some models may display more than one condition. This initial set of models will necessarily be incomplete: the number of conditions is too large, and some of the conditions are too ill-understood for modeling. Thus, unlike system modeling where one typically starts from one ground truth model to be verified, our starting point is an *incomplete set of models*. In this paper, we build such a set of formal heart models using a network of timed automata in Section ???. Model checking for timed automata is decidable [???].

Next, in typical formal verification practice, when the state space of a model M is too large, *predicate abstraction* is used to reduce the size of the state space while still preserving all the behavior of the original model. This reduction in size may allow model checking where it wasn't possible before. Because the abstract model M' also introduces new behavior that didn't exist in the ground truth M, this new behavior is rejected as *spurious* if it is encountered, and the abstraction is refined. This is the familiar CEGAR procedure[???]. However when modeling the environment, we use abstraction differently. Because we start from an *incomplete* set of models, we would like our abstraction procedures to introduce new *physiologically meaningful* behavior which might actually be produced by heart models not in the initial set. These then correspond to heart conditions not taken explicitly into account. This motivates the introduction of domain-specific abstraction rules R in Section ???: like predicate abstraction, they produce models that over-approximate the behavior of the model they are applied to (i.e., $\mathbb{B}(R(M)) \supset \mathbb{B}(M)$). However, the new behavior they introduce might not be spurious. We demonstrate such a case in Section ???. If model checking returns a counter-example on $R(M)$, the physician can decide whether this is actually physiologically plausible behavior and therefore the pacemaker needs to be debugged, or this is indeed spurious and should be thrown out (and the abstraction refined).

Note also that predicate abstraction deals with only one model. In the proposed framework, the abstraction rules can apply to several models $M_1, \ldots, M_n$ at a time, and can merge them together such that $\cup_{i=1}^{n} R(\mathbb{B}(M_i)) \subset \mathbb{B}(M)$. Moreover, because we start from a set of initial models, and use a set of abstraction rules, we actually produce an *abstraction tree* rather than an abstraction chain. We demonstrate the construction and use of such a tree for the formal verification of pacemakers in Section ???

[3] The medical term is 'arrhythmias'.

1.1 Technical preliminaries

In this section we briefly review the notions of abstraction and over-approximation of system. Let M be a system model and S be its state space (in Section ??? we give a specific formalism in which our systems are modeled). A *trace* $\mathbf{x}$ of M is an infinite sequence of states produced by M: $\mathbf{x} \in S^\omega$.The *behavior*

1.2 The Model Checking Problem

Model-based design has been proposed to speed up system software design and provide safety promises in physiological requirements.] For a model M with state space S, we define a behavior of the model as an execution trace $\delta \in S^*$. The reachable behavior space of the model M is denoted as $\mathbb{B}(M) \subset S^*$. A property φ defines a region in the behavior space within which the property is satisfied, which can be denoted as $\mathbb{B}(\varphi)$. A model checking problem is to use mathematical tool to explore the whole reachable behaviors of a model M against a property φ such that $\mathbb{B}(M) \subseteq \mathbb{B}(\varphi)$. We denote it as $M \models \varphi$. Execution traces $\delta_v \in \mathbb{B}(M)/(\mathbb{B}(M) \cap \mathbb{B}(\varphi))$ will be returned by the model checker as property violations so that the designer can analyze and address the problem.

In closed-loop model checking, the closed-loop system $M_E||M_P$ consists of the system under check M_P, and environment conditions modeled as M_E. In the case of medical devices, the devices are designed to improve certain physiological conditions. In general, a closed-loop requirement φ_C is in the form of $\varphi_E \Rightarrow \varphi_P$, in which φ_E is the open-loop physiological condition that the device encounters, and φ_P is the closed-loop physiological condition that the device should achieve. Then we have:

$$M_E \models \varphi_E, M_E||M_P \models \varphi_P \Rightarrow M_E||M_P \models \varphi_C \tag{1}$$

1.3 Model Abstraction with Over-approximation

The behavior space of the actual system is too large for model checker to exhaustively explore. A model of the system which covers all behaviors of the system can be developed which can not only reduce complexity, but also having the suitable formalism for the model checker. An abstraction function h abstracting model M to M' is a non-surjective function from state space S to the new abstract state space S' such that:

$$\forall s \in S, \exists s' \in S' \text{ s.t. } h(s) = s'$$

This definition can be extended to behaviors $\delta \in \mathbb{B}(M)$, such that:

$$\forall \delta \in \mathbb{B}(M), \exists \delta' \in \mathbb{B}(M') \text{ s.t. } h(\delta) = h(\delta')$$

From the definition, we know that the abstract model M' covers all behaviors of M, which is referred to as *over-approximation*. We represent the over-approximation relationship as $M \lhd_h M'$, and the relationship between reachable behavior spaces as $h(\mathbb{B}(M)) \subseteq \mathbb{B}(M')$. Then we have:

$$h(\mathbb{B}(M)) \subseteq \mathbb{B}(M'), \mathbb{B}(M') \subseteq h(\mathbb{B}(\varphi)) \Rightarrow h(\mathbb{B}(M)) \subseteq h(\mathbb{B}(\varphi)) \Rightarrow M \models \varphi$$

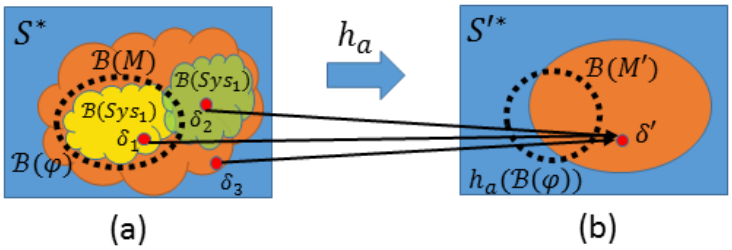

Fig. 1. Two models such that $\{Sys1, Sys2\} \lhd_h M$. An over-approximation function h_a is applied to M to obtain M'. By model checking the abstract model M' against property φ we have $M' \not\models \varphi$ and δ' is returned as counter-example. However, δ' corresponds to 3 different behaviors in the original behavior space: δ_1 satisfied and valid; δ_2 unsatisfied and valid; δ_3 unsatisfied and invalid

The behavior space in which φ is defined need to be clarified.

In general the state space of S' is smaller than S while preserving the property, thus preferable during model checking.

Over-approximation can also be used to cover the behaviors of multiple models. A model M' is an over-approximation of model M_1 and M_2 if

$$\forall \delta \in \mathbb{B}(M_1) \cup \mathbb{B}(M_2), \exists \delta' \in \mathbb{B}(M') \text{ s.t. } h(\delta) = h(\delta')$$

We denote it as $\{M_1, M_2\} \lhd_h M'$, such that $h(\mathbb{B}(M_1) \cup \mathbb{B}(M_2)) \subseteq \mathbb{B}(M')$.

1.4 Ambiguities Caused by Over-approximation

Validity Ambiguities: Since h is non-surjective, there exists behaviors in M' that do not exists in the original model M:

$$\exists \delta' \in \mathbb{B}(M') \text{ s.t. } h^{-1}(\delta') \not\subset \mathbb{B}(M)$$

These invalid behaviors may contribute to false-negatives during model checking.

Context Ambiguities: For models over-approximated to cover the behaviors of multiple models, distinct behaviors from different models can be indistinguishable in the abstract model. For $\{M_1, M_2\} \lhd_h M'$, we may have :

$$\delta_1 \in \mathbb{B}(M_1), \delta_2 \in \mathbb{B}(M_2) \text{ s.t. } h(\delta_1) = h(\delta_2) = \delta', \delta' \in \mathbb{B}(M')$$

It is not a problem if $M' \models \varphi$, however if $\delta' \not\models \varphi$ is returned as a counter-example, it loses the context in which the property is violated and is hard to interpret the result. A small example is shown in Fig. 1 to illustrate the potential ambiguities due to over-approximation.

1.5 System Modeling vs. Environment Modeling

During closed-loop model checking, there are different focuses on the system model and the model of its environment, thus modeling them require different strategy.

Over-approximation: In closed-loop model checking, there is only one concrete system. However there can be countless number of environmental conditions which require different models to represent. While over-approximating the system model aim to reduce its state space and computational cost, over-approximating the environment models aim to cover the behaviors of multiple environmental conditions that are explicitly modeled, or implicitly included. Since it is impossible to exhaustively model all possible environment conditions, over-approximation is a good way to cover multiple environment condition without increasing computational cost.

Validity of a counter-example: Behaviors introduced into the over-approximation of a system model are all *spurious*, or invalid. For two models such that $M \lhd_h M'$, δ' is spurious if the following condition holds:

$$\nexists \delta \in \mathbb{B}(M) \text{ s.t. } h(\delta) = \delta'$$

However, for environment models such that $\{M_1, M_2, ...M_n\} \lhd_h M'$, and an execution $\delta' \in \mathbb{B}(M')$, the following condition is not enough to prove δ' is spurious.

$$\forall i \ \nexists \delta \in \mathbb{B}(M_i) \text{ s.t. } h(\delta) = \delta'$$

Since there may be a valid environment model $M_c \notin \{M_1, M_2, ...M_n\}$ and $\mathbb{B}(M_c) \subset \mathbb{B}(M')$ and $\delta' \in \mathbb{B}(M_c)$. It is thus up to the domain experts to determine the validity of the counter-example.

1.6 Counter-Example-Guided Abstraction and Refinement (CEGAR)

In [4] the authors proposed a framework to over-approximate the system using proposition abstraction. Upon property violation the abstract counter-example is checked for its validity on the system. If the counter-example is spurious the model is then refined to eliminate the spurious counter-example. This process is then resumed on the refined model until either a valid counter-example returns or no counter-examples are returned.

From the above procedure we can see that CEGAR framework works on system modeling. However, CEGAR cannot be applied for environment modeling for the following reasons. First, the proposition abstraction can not over-approximate multiple models into one abstract model. With multiple environment models the validity of counter-examples cannot be determined by concretizing them on the set of environment models, as discussed in the last section. If over-approximation is used for environment modeling, there needs to be a more suitable framework to balance the abstraction and refinement of the environment models.

1.7 Abstraction-tree-based Model Abstraction for Environment Modeling

In this paper we propose framework for environment modeling in closed-loop model checking of medical device software. An incomplete set of physiological models are first developed to represent different physiological conditions.

A set of physiological abstraction rules are then developed based on physiological knowledge, which ensure the physiological relevance of the behaviors introduced into the abstract models.

Then the rules are applied in certain order onto the set of physiological models, resulting an abstraction tree $G = (V, E)$. Each leaf in the tree is a physiological model and the edges are applications of an abstraction rule. The abstraction tree can then be used as environment models for closed-loop model checking.

The closed-loop requirement $\varphi_c : (\varphi_E \Rightarrow \varphi_P)$ has constraints on the environment in φ_E. During the abstraction steps, certain sub-states or transitions of the environment models may be removed or merged. If the variables mentioned in φ_E, which is denoted as $Var(\varphi_E)$, is not a subset of the variables of an environment model M, which is denoted as $Var(M)$, the model M is not appropriate for the requirement φ_c. The first step for closed-loop model checking is to choose the most abstract environment model(s) from the tree which are appropriate for the requirement. These models will be used as the initial environment models M_E during model checking.

For a system model M_S and a physiological requirement φ_c, closed-loop model checking is performed such that:

$$\forall M \in M_E, \text{ check } M||M_S \models \varphi_c$$

If the requirement is all satisfied, the system model M_S satisfy φ_c under environment condition covered by the models in M_E. Upon violation of the requirement, the model checker returns a counter-example $\delta_c \in M_c \in M_E$. However, counter-examples at the abstract level are difficult to interpret and there may exists ambiguities. To concretize the counter-example and enumerate possible physiological context, we explore the abstraction tree. Model checking is then performed such that:

$$\forall M \in Child(M_c), \text{ check } M||M_S \models \varphi_c$$

The procedure recurs until 1) the leaves of the tree is reached, or 2) there is no violations in the child nodes. The counter-examples returned from the most refined models are then submitted to the physicians for analysis.

1.8 Contributions

2 Formal Models of the Environment

To perform closed-loop model checking of medical devices, we need formal models of their physiological environment to represent different physiological conditions

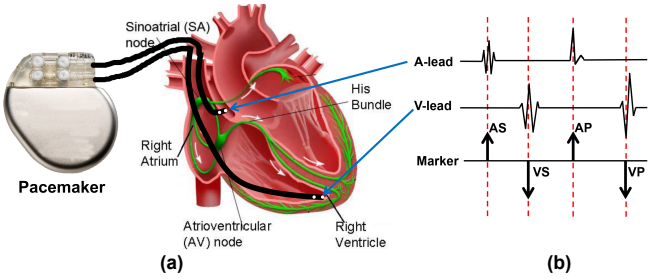

Fig. 2. (a) Lead placement for a dual chamber pacemaker (b) Electrogram (EGM) signals from pacemaker leads and corresponding internal event markers

the devices may encounter. Physiological requirements which the closed-loop system should satisfy are also formalized. In this section, we use implantable pacemakers as example. Timed-automata [2] models of the human heart are developed as the environment model. Physiological requirements are formalized with monitors and TCTL formula [1]. Model checking can then be performed on the closed-loop system in model checker UPPAAL [10].

2.1 Electrophysiology (EP) Basics

At cellular level, a heart tissue can be activated by external voltage. Certain tissue also has capability to self-activate, which contribute to natural heart beats. Once activated (Marker 1 in Fig. 3), the voltage outside the tissue changes over time, which is referred to as *Action Potential* (Fig. 3.(a)). The action potential can be divided into two functional timing periods: The *Effective Refractory Period (ERP)*, during which the tissue cannot be triggered by another activation; and the *Rest* period, during which the tissue can be activated and at the end of which the tissue will self-activate. The voltage change of the heart tissue will activate the tissue nearby with certain delay (Marker 2 in Fig. 3). A healthy heart generates periodic electrical impulses to control heart rates according to physiological needs. These impulses conduct through the heart, triggering coordinated muscle contractions and pump blood to the rest of the body. The underlying pattern and timing of these impulses determine the heart's rhythm and are the key to proper heart functions. Derangements in this rhythm are referred to as *arrhythmia*, which impair the heart's ability to pump blood and compromise the patients' health. Arrhythmia are categorized into so-called Tachycardia and Bradycardia. Tachycardia features undesirable fast heart rate which results in inefficient blood pumping. Bradycardia features slow heart rate which results in insufficient blood supply. Different heart conditions can be distinguished by the timing of generation and conduction of electrical signals, which are researched in clinical setting referred to as *Electrophysiology (EP)*[9].

The implantable cardiac pacemakers are rhythm management devices designed to treat bradycardia. A typical dual chamber pacemaker has two leads inserted into the heart through the veins which can measure the local electrical

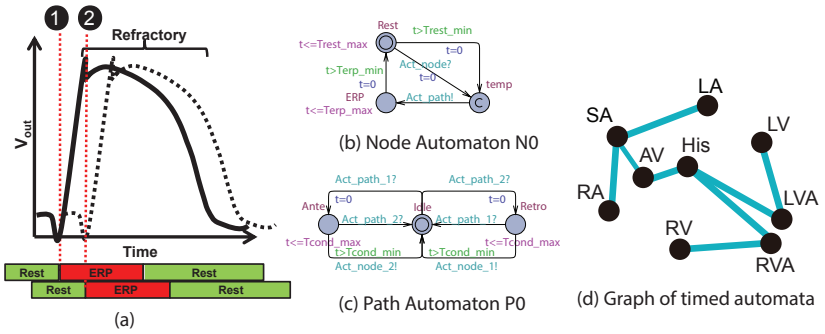

Fig. 3. (a) Action potential for a heart tissue and its tissue nearby (dashed). (b) Node automaton. (c) Path automaton. (d) An example model of the heart consist of a network of node and path automata

activities of the right atrium and right ventricle, respectively. According to the timing between sensed impulses the pacemaker can deliver electrical pacing to the corresponding chamber to maintain proper heart rhythm (Fig. 2).

2.2 Timed-automata Models of the Heart

Different heart conditions can be distinguished by the *timing* of the electrical conduction, and the *topology* of the electrical conduction system of the heart. It is thus possible to develop a spatial and temporal model of the heart using networks of timed-automata [2]. In [8] and [6], we proposed an Electrophysiology heart model to represent the timing behaviors of the electrical conduction system. Heart tissue can be modeled as *Node automata* and the conduction delays between nodes are modeled as *Path automata* (Fig. 3). Heart conditions can be modeled with node and path automata with different topology and timing parameters.

The node automaton initializes with Rest state. From Rest state, the node can either self-activate or be activated by external activations (indicated by Act_node). Upon activation the node transition to the ERP state and activate all the paths connecting to the node (indicated by Act_path). In the ERP state the node does not respond to external activations. At the end of ERP state the node transition to the Rest state.

The initial state of a path automaton is Idle, which corresponds to no conduction. A path has two conduction directions, forward and backward. These are represented by the states Ante and Retro, named after their standard physiological terms Antegrade and Retrograde. If Act_path event is received from one of the nodes (1 or 2) connected to the path, the a transition to either Ante or Retro state will occur in the path automaton. At the end of Ante and Retro state the path will transition to Idle state and send Act_node signal to the node automaton connected to the other end of the path (2 or 1).

The spatial and temporal properties of a given human heart condition can be modeled by a network of node and path automata with different parameters (Fig. 3.(d)). We use node automata to represent key structures of the heart and the path automata specifies the connectivities of those key structures and the conduction delays among them. The network can be viewed as a labeled directed graph: the vertices of the graph represent nodes, or locations, in the heart, while edges represent conduction paths between the nodes.

Definition 21 (Labeled graph) *A **labeled graph** is a directed graph* $G = (V, E, A)$ *where* V *is a finite set of vertices,* $E \subset V \times V$ *is a finite set of directed edges, and* A *is a total labeling function* $A : V \cup E \rightarrow TA$ *where* TA *is the set of parametrized timed automata. The function* A *labels each vertex with a* node automaton*, and each edge with an* edge automaton*. All node (resp. path) automata share the same structure, and differ only in the values of their parameters.*

The heart model structure has been used to model varies heart conditions and all of them have been validated by Electrophysiologists [7,5].

2.3 Formalizing Physiological Requirements

Physiological requirements need to be formalized for closed-loop model checking. A requirement $\varphi_C : \varphi_E \Rightarrow \varphi_P$ contains open-loop environmental constraints in φ_E and desired closed-loop condition specified in φ_P. In [3] we mapped physiological heart conditions to parameter ranges in the heart model, which can be used to enforce φ_E on the heart models. In general, φ_P are timing constraints on closed-loop timing events. In [3] we developed general monitors in Stateflow for closed-loop testing of physiological requirements. They can be translated to timed-automata which are shown in Fig. 4. The $M_{sing}(event, thresh\_min, thresh\_max)$ enforces the time interval between two *event* signals within $[thresh\_min, thresh\_max]$. $M_{doub}(event1, event2, thresh\_min, thresh\_max)$ enforces the time interval between $event1$ and $event2$ signals within $[thresh\_min, thresh\_max]$. Model checking is performed on the closed-loop system include the heart model M_H, the pacemaker model M_P and the monitor M. φ_P can be then represented with TCTL formula:
A[] (not M.Err)

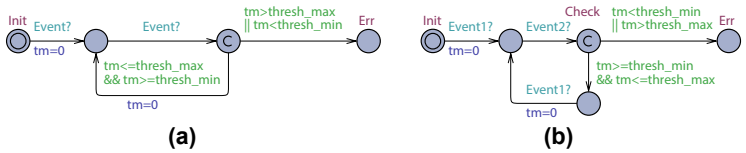

Fig. 4. (a) M_{sing} for single event; (b) M_{doub} for two events

3 Physiological Abstraction rules

During closed-loop model checking, the environment model(s) should cover During over-approximation additional behaviors are introduced to the abstract model. There are a lot of abstraction functions for a model, but not all the abstractions introduce physiological-relevant behaviors. In this section, we introduce 7 physiological abstraction rules which introduce physiological-relevant behaviors to the heart models. The rules are based on the combination of physiological domain knowledge and formal method.

3.1 Why Predicate Abstraction Does Not Work

3.2 Rule 4: Merge Parameter Ranges

For heart models with the same node and path topology $(H_i, i = [1, p])$, they can be abstracted by a single heart model H_a such that:

$$\forall i, j, H_a.\theta^j_{min} = min(H_i.\theta^j_{min})$$

$$\forall i, j, H_a.\theta^j_{max} = max(H_i.\theta^j_{max})$$

H_a covers all possible behaviors, thus is an abstraction of $H_i, i \in [1, p]$. The added behaviors in the abstract model do not affect the number of transition groups

3.3 Rule 6: Replace Blocking With Non-deterministic Conduction

The effect of a node automaton blocking an activation signal is equivalent to a path not conducting. So we designed new abstract node and path automata $N1$ and $P1$. For any $P_1 - N_n - P_2$, it can be replaced by a P with:

$$P.cond_{min} = P_1.cond_{min} + P_2.cond_{min}$$

$$P.cond_{max} = P_1.cond_{max} + P_2.cond_{max}$$

For any self-activation node N:

what is N_1 and what is replacing? what is N

$$N'.rest_{min} = N.ERP_{min} + N.rest_{min}$$
$$N'.rest_{max} = N.ERP_{max} + N.rest_{max}$$

3.4 Rule 7: Replace Conductions With Self-activation

The effect of conduction path is to connect activations from a self-activation node to another. By setting the interval between self-activations to $[0, \infty]$ the observable behaviors of the heart will not change.

For all node N, $N.Rest_{min} = 0, N.Rest_{max} = \infty$ and delete all paths.

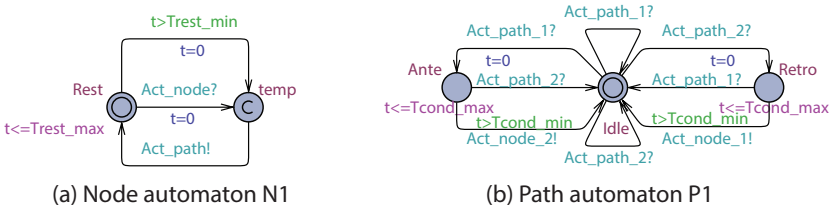

Fig. 5. Heart Model Abstractions

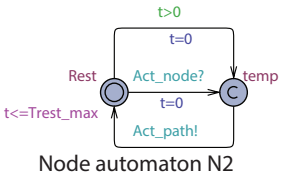

Fig. 6. Heart Model Abstractions

3.5 Heart Model Abstraction Tree

We first develop a list of concrete heart models corresponding to common heart conditions. By systematically applying rules from Section 3 we created an abstraction tree HM_tree for the heart (Fig. 7). Note that applying rules in different order results different abstraction tree. The order used to obtain HM_tree is based on the domain knowledge that certain heart conditions may have similar behaviors and similar inputs to the pacemaker. This systematic grouping maintains the physiological-relevance of the heart model even at higher abstraction levels, and reduce the necessity to resolve ambiguities at lower abstraction levels when model checking certain requirements.

Besides model structure changes, applying abstraction rules also merges the behaviors of the model, which is documented for each abstraction. Here we demonstrate the behavior merging process on a subset of the abstraction tree (Fig. 7). In H''_{vt}, we have self activation behavior $NA.self$ and $NV.self$ for node NA and conduction behavior NV,$(NA, AV').cond$ and $(AV', NV).cond$ for path (NA, AV') and (AV', NV). After applying Rule 6 we obtain H''_{vt} with

$$NA'.self = \{NA.self\}, NV'.self = \{NV.self\}$$

$$(NA', NV').cond = \{AV'.block, (NA, AV').cond, (AV', NV).cond\}$$

After applying Rule 7 we obtain H_{all} with:

$$NA''.self = \{NA.self, (NA', NV').cond\}$$

$$NV''.self = \{NV.self, (NA', NV').cond\}$$

These information on behavior abstractions are useful to determine whether a model is appropriate for a requirement.

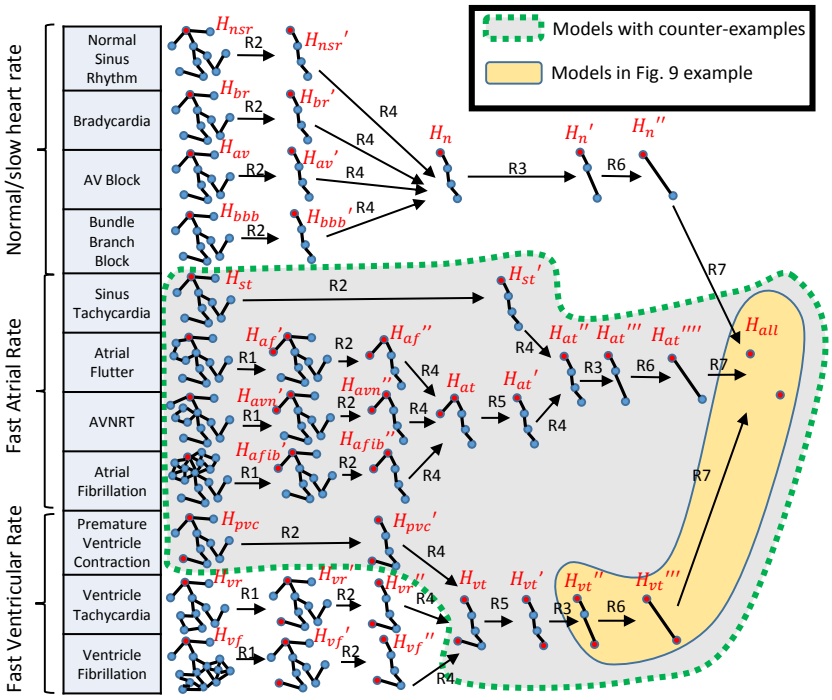

Fig. 7. Heart Model Abstractions

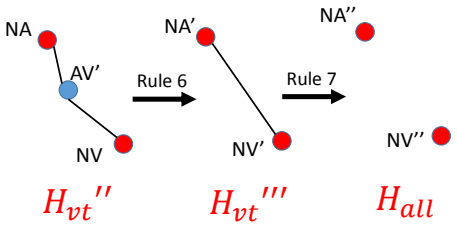

Fig. 8. Abstraction Example

4 Closed-loop Model Checking Using the Abstraction Tree

After the abstraction tree is built, it can be used for closed-loop model checking. The next question is how to navigate through the abstraction tree so that the most appropriate model(s) are selected for different requirements, and provide the most concrete counter-examples, possibly under multiple physiological conditions, for the physicians to determine the validity of the counter-examples. [10]

4.1 Select Initial Abstraction(s) Appropriate For the Requirement

Physiological requirements are in general conditional in the sense that they require conditions to hold under certain open-loop physiological constraints. These constraints map to parameters for certain transitions of the physiological models, which may be merged during the abstraction process. The most abstract model in the abstraction tree is built to cover the input space to the device as much as possible, thus it may not have enough details to constrain the behaviors of the model according to the constraints in the requirement. Thus the first step of closed-loop model checking is to select the most abstract models which are appropriate for the requirement. A model M is appropriate for a requirement Req if the environment transitions mentioned in the requirement, denoted as $EnvT(Req)$, is a subset of the environment transitions associated with the model $EnvT(M)$. The following algorithm finds the most abstract heart models in the abstraction tree HM_tree that are appropriate for a requirement Req.

```
Algorithm 1
function [HM]=eligible(HM_tree,Req)
BM = root of HM_tree
while (BM is not empty)
 For every model M in BM
  If (EnvT(M) \cap EnvT(Req) == EnvT(Req))
   Remove M from BM
   save M in HM
  else
   add children of M in BM
  endif
 endfor
endwhile
Return HM
```

4.2 Obtaining Concrete Counter-examples Under Corresponding Physiological Conditions

One challenge for modeling physiological environment of medical devices is the large variety of physiological conditions that the devices may encounter. The set of concrete models (the leave nodes in the abstraction tree) only represent a subset of all possible physiological conditions, thus model checking the device on all of the concrete models is not enough to guarantee the satisfaction of the requirement. By applying physiological abstraction rules, additional physiological-relevant behaviors are incorporated into the abstract models by over-approximation, providing more coverage. However, if model checking on an abstract model returns a counter-example, if is difficult to determine whether the counter-example is a valid execution. Due to the incomplete nature of the set of concrete models, if the counter-example cannot be concretized on all the concrete models, it does not mean it is invalid. Therefore it is up to the physician to decide whether a counter-example is valid or not. The algorithm below

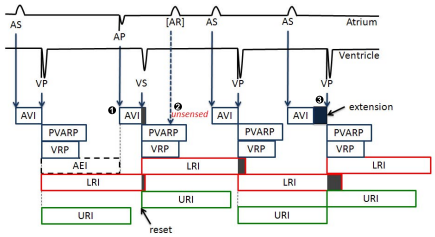

Fig. 9. Heart Model Abstractions

explore the abstraction tree during model checking and provide the physician the most concrete counter-example(s), if there exists any.

```
Algorithm 2
Input: system model PM, abstraction tree for environment HM_tree, requirement Req
Output: Counter examples CE and corresponding model refinements
[HM]=eligible(HM_tree,Req);
Mc= HM;
 while (Mc is not empty)
  For all M in Mc
   [satisfied,CE]=ModelChecking(M,PM,Req);
         Remove M from Mc
         If satisfied==0
          add the children of M to Mc
                cache CE
        else
                save CE from the parent model
        endif
 endfor
endwhile
Return all saved CEs and their corresponding models
```

5 Case Study: Closed-loop Model Checking of a Dual Chamber Pacemaker

5.1 Pacemaker Model

5.2 Requirement Encoding

Physiological requirements can be automatically mapped to model behaviors and parameters. In general, a requirement has *pre-conditions* under which the requirement should hold. The pre-conditions are open-loop physiological conditions which are in terms of constraints on model behaviors. The *post-conditions* are closed-loop behaviors that the device should achieve. The requirement below is designed to prevent the pacemaker from pacing too fast.

- Pre-condition: Atrial self-activation rate (60bpm - 200bpm)
- Post-condition: Intervals between ventricular paces should be no shorter than 500ms

With behavior mapping and a monitor (Fig. 4), the requirement can be translated to:

$$Req1 : NA.self.min = 300 \&\& NA.self.max = 1000 \Rightarrow notM.Err$$

The environment behavior specified in $Req1$ is $EnvB(Req1) = NA.self$.

5.3 Choosing Appropriate Heart Model For the Requirement

To verify the closed-loop system with pacemaker model PM and heart model abstraction tree HM_tree (Fig. 8) against requirement $Req1$, we start by searching for the most abstract appropriate models from the abstraction tree. We call the function specified in Algorithm 1: $[HM] = eligible(HM\_tree, Req1)$. At the root level heart model H_{all}, $NA.self$ is abstracted with $(NA', NV').cond$. As the result, H_{all} is not appropriate for $Req1$. $NA.self$ is not abstracted with other behaviors in the children of H_{all}: $H''_n, H''''_{at}, H'''_{vt}$, thus these 3 heart models are outputted as the most abstract appropriate models for $Req1$.

5.4 Providing Meaningful Counter-examples to the Physicians

After we choose the appropriate models for $Req1$, we have:

$$HM = \{H''_n, H''''_{at}, H'''_{vt}\}$$

Then we run Algorithm 2. By model checking on all 3 initial models in UPPAAL we have:

$$[1, []] = ModelChecking(H''_n, PM, Req1)$$
$$[0, CE_1] = ModelChecking(H''''_{at}, PM, Req1)$$
$$[0, CE_2] = ModelChecking(H'''_{vt}, PM, Req1)$$

The algorithm keeps going down the abstraction tree, and at certain intermediate step we have:

$$[0, CE_a] = ModelChecking(H'_{st}, PM, Req1)$$
$$[0, CE_b] = ModelChecking(H'_{pvc}, PM, Req1)$$
$$[0, CE_c] = ModelChecking(H_{at}, PM, Req1)$$
$$[1, []] = ModelChecking(H''_{vr}, PM, Req1)$$
$$[1, []] = ModelChecking(H''_{vf}, PM, Req1)$$

The counter-examples from more refined models provide more detailed mechanism of the requirement violations, and distinguish the physiological conditions

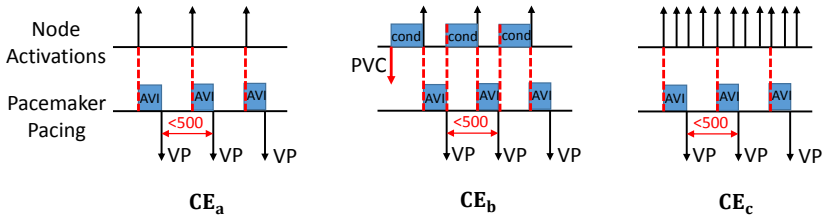

Fig. 10. Counter-examples

that can trigger the violations. It is much easier for the physician to determine the validity of the counter-example. The counter-examples from the example above are illustrated in Fig. 10. CE_a corresponds to fast intrinsic heart rate thus is a safe execution of the pacemaker. CE_b corresponds to a scenario called Endless Loop Tachycardia during which the pacemaker and the heart forms a conduction loop that increases the heart rate inappropriately. In CE_c the pacemaker extends fast atrial rate to more dangerous fast ventricular rate, which is referred to as Atrial Tachycardia Response of a pacemaker. Both CE_b and CE_c are inappropriate executions of the pacemaker .CE_a and CE_b can have the same input-output executions on the pacemaker side and can only be differentiated on the heart model side. After the physician examines the counter-example the programmer can work on debugging.

6 Conclusions and future research

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 414–425, 1990.

[2] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *ACM/IEEE International Conference on Cyber-Physical Systems*, pages 33–38, 2010.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counter Example-Guided Abstraction Refinement for Symbolic Model Checking. *J. ACM*, 50(5):752–794, 2003.

[5] Z. Jiang, A. Connolly, and R. Mangharam. Using the Virtual Heart Model to Validate the Mode-Switch Pacemaker Operation. *IEEE Engineering in Medicine and Biology Society*, pages 6690 –6693, 2010.

[6] Z. Jiang, M. Pajic, R. Alur, and R. Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, pages 1–23, 2013.

[7] Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam. Real-Time Heart Model for Implantable Cardiac Device Validation and Verification. In *22nd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 239 –248, July 2010.

[8] Z. Jiang, M. Pajic, and R. Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceeding of IEEE Special Issue on Cyber-Physical Systems*, 2011.

[9] M. Josephson. *Clinical Cardiac Electrophysiology.* Lippincot Williams and Wilkins, 2008.

[10] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, pages 134–152, 1997.

7 outline

1. Introduction
   – The difference between software specification and physiological requirement
   – The necessity of closed-loop verification
   – Model-based design enables closed-loop verification at earlier design stage.
   – Closed-loop model checking
   – Difference between system modeling and environment modeling
   –– One concrete system vs. countless concrete systems
   –– Abstraction for simplicity vs. generality
   – CEGAR does not work on environment modeling
   –– Physiological requirements focus on environment conditions, thus predicate abstraction does not work (example)
   –– Random abstraction functions introduce behaviors that are not physiological possible
   –– Validity of counter-examples cannot be checked
   – Abstraction tree based environment model abstraction and refinement
   –– Physiological abstraction rules introduce physiological behaviors into the abstract model
   –– Documenting merging of physiological transitions to maintain the physiological-relevance of the abstract model
   –– Abstraction tree keeps track of abstraction-refinement relations
   –– Simple algorithm to select appropriate initial model for requirements
   –– Algorithm to obtain the most concrete counter-examples corresponding to multiple physiological conditions.
   –– Physician determine the validity of the counter-examples
   – Some elements of the proposed method can be generalized and have a wider applicability: namely the abstraction tree and MC on the tree (abstractness measure, search procedure)
   – Give overview of method in a block diagram and briefly explain it in text.
2. Formal models of the environment
   (a) Heart basics, enough to understand the proposed models
   (b) Emphasis: different heart conditions necessitate different models.
   (c) Creation of initial formal heart models: graph structure + node and path automata
   (d) Emphasis: our initial set of models is not necessarily complete. That is, there are more heart conditions that could be modeled, but aren't. It is always a work in progress.
      – Therefore we want to add behavior to try and cover things not covered by the models. Adding models i not the answer since it will always be an incomplete set. Not to mention it's a manual process.
   (e) Formalization of physiological requirements
3. Abstraction rules: how to add behavior to the initial set of models

(a) Why predicate-based abstraction can be inadequate?
  – If we get a cex which is spurious by the classical definition, it is not necessarily spurious physiologically. Remember we are enriching the behavior precisely to cover new conditions.
  – So it's up to the physician to decide if the cex is spurious or not. But predicate-based abstraction might give hard-to-understand cex.
  – Example
  – Therefore, need the added behavior to be physiologically meaningful so it's understandable.

(b) Physiologically meaningful abstraction rules
  – Give formal definition of a rule as a graph-to-graph function $R : G \to G'$
  – Why do you call it abstraction?
  – Given 2 concrete examples (more in tech report)

(c) Example of rule application.

(d) Abstraction tree

4. Model-checking with the abstraction tree
   (a) Measure of abstractness; define 'appropriate for the requirement'
   (b) Search procedure
   (c) What happens if a counter-example is found: give the physician the most concrete model on which the cex still shows up.
   (d) Case study: i.e., elaborate example
5. Conclusions and future research