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 capability of autonomously affecting 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.

There are two categories Since the requirements describe the intended behaviors of the closed-loop system consists of the environment and the system, verifying whether the specifications of the system satisfy the requirements requires closed-loop verification. In the medical device industry, closed-loop verification is performed in the form of clinical trials in which the devices are tested on the real patients. Clinical trials can only cover very limited environmental conditions due to extremely high cost. Moreover, clinical trials are often conducted at the last design stage. Fixing bugs at this stage is also very costly.

Model-based design has been proposed to speed up system software design and provide sufficient safety maintaining the safety promises in software requirement. It can potentially enable closed-loop verification of the software requirements at an earlier stage thus reduce cost. In a model-based design framework, the system software is first designed as an abstract model. Together with an environment model the closed-loop system can be verified against software requirements using model-checking techniques. The verified system model can then be rigorously translated into the software implementation (semi-)automatically so that the software implementation also satisfies all the software requirements.

One of the biggest challenge for model-based design is to manage how much detail a model should have during model checking. In general the model should be abstract enough to ignore unnecessary details in order to reduce computational cost, but also detailed enough to distinguish execution paths that 1) satisfy/violate a property, and/or 2) are valid/invalid due to the extra behaviors introduced during model abstraction/approximation. These ambiguities must be removed from model to prevent false positives/negatives during model checking.

1.1 The Model Checking Problem

This section is too long. Reduce to half a page. Also, doesn't belong in the introduction.

For a model M with state space S, we define the behavior of the model as an execution trace in $\delta \in S^*$. The reachable behavior space of 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

$M \models \varphi$. Property violations should be returned as execution traces δ_v so that the designer can analyze and address the problem.

1.2 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 $M' \models M$, and the relationship between reachable behavior spaces as $\mathbb{B}(M) \subset_a \mathbb{B}(M')$. Then we have:

$$\mathbb{B}(M') \subseteq_a \mathbb{B}(M), \mathbb{B}(M) \subseteq_a \mathbb{B}(\varphi) \rightarrow \mathbb{B}(M') \subseteq_a \mathbb{B}(\varphi) \rightarrow M_t \models \varphi$$

Since the properties are preserved In general the state space of S' is smaller than S while still preserve the property, during model checking the more abstract model can be used to check the property.

why give new names to familiar things? Especially since these names don't make sense at this point. E.g. you speak of validity ambiguity, but it is not at all clear what is being validated and why the equation expresses an 'ambiguity'. This is basically a description of what happens when abstracting, which most formal people are familiar with, so merge with section 1.1

Context Ambiguities

Since h is non-surjective, certain states in S will not be distinguishable in S':

$$\exists s_1, s_2 \in S \text{ s.t. } h(s_1) = h(s_2) = s', s' \in S'$$

For behaviors we have:

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

In the abstract model δ_1 and δ_2 are not distinguishable

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

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

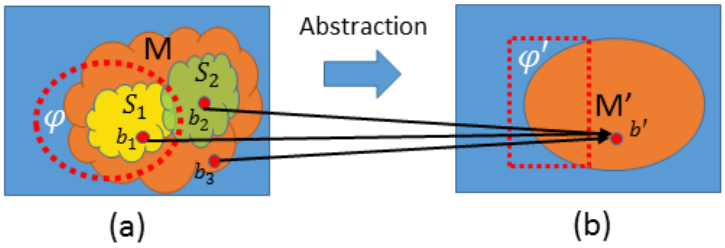


Fig. 1.

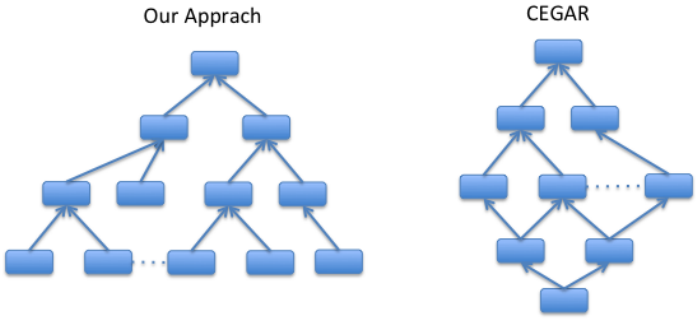


Fig. 2.

System Model vs. Environment Model

1.3 Closed-loop Model Checking

1.4 Contributions

2 Formalizing Physiological Knowledge

The physiological knowledge required during closed-loop model checking of medical devices are associated with the *physiological models* and the *physiological requirements*. It is thus important to maintain the physiological context of the models and link model behaviors to physiological requirements. In this section we use heart modeling as example to demonstrate physiological knowledge encoding.

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. 4), the voltage outside the tissue changes over

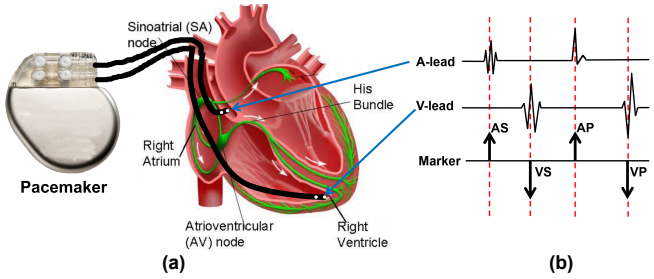


Fig. 3. (a) Lead placement for a dual chamber pacemaker (b) Electrogram (EGM) signals from pacemaker leads and corresponding internal event markers

time, which is referred to as *Action Potential* (Fig. 4.(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. 4). 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)*[7].

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 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. 3).

2.2 Timed-automata Models of the Heart

Models of different physiological conditions have to be developed to be able to interact with the medical devices. This ~~initial~~ set of ~~physiological~~ models should be able to distinguish the physiological conditions that each model represents. Ideally the models are developed using formalisms that are suitable for provable abstraction and model checking. The timing behaviors of the heart can be modeled as timed automata [1]. In [6] and [4], we proposed an Electrophysiology

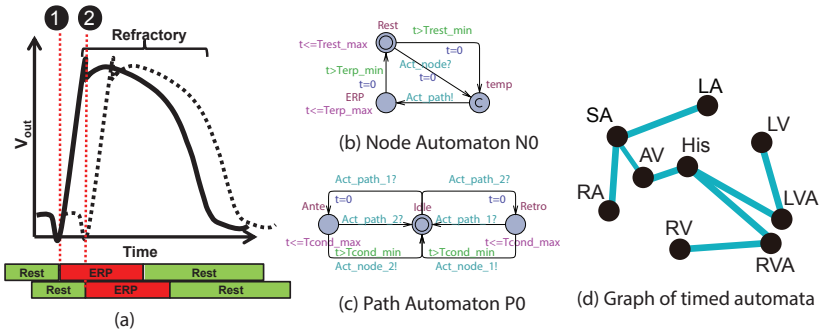


Fig. 4. (a) Action potential for a heart tissue and its tissue nearby (dashed). (b) Node automaton. (c) Path automaton. (d) An example model of the heart consist of a network of node and path automata

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. 4).

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. 4.(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* $A : V \cup E \to TA$ *where* TA *is the set of parametrized timed automata. The function* A *labels each vertex with a* node au-

tomaton, *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 [5,3].

2.3 Describe Model Behaviors with Physiological Context

During model abstractions, the abstract model covers all transitions of the more refined model. Transitions corresponding to physiological behaviors may be merged or replaced by other transitions. Without documenting these information, the abstract models lose their physiological context. In the heart models we modeled physiological timing periods using locations in timed automata. The minimum time an automaton can stay in those locations is limited by the guard on a transition out of the location, and the maximum time is limited by the clock invariant of the state. The transitions of heart tissue can be categorized into 3 basic transition groups:

- **self:** The self-activation of the node automata. The *min* and *max* parameters equal to $Trest\_min$ and $Trest\_max$ parameters in the node automata, which specify the minimum and maximum intervals between consecutive self-activation events.
- **block:** The blocking property of the node automata. The *min* and *max* parameters equal to $Terp\_min$ and $Terp\_max$ parameters in the node automata, which specify the minimum and maximum intervals between consecutive activations that can trigger path conduction.
- **cond:** The conduction property of the path automata. The *min* and *max* parameters equal to $Tcond\_min$ and $Tcond\_max$ parameters in the node automata, which specify the minimum and maximum delays between a node activation on one end and the activation on the other end.

As an example, a node automata NA has two transition groups, which can be represented by $NA.self$ and $NA.block$

2.4 Encoding Physiological Requirements

Devices are designed to improve certain physiological conditions, the performance of the devices is evaluated on the difference between the patient conditions without the device and with the device. The device should also avoid deteriorating certain patient conditions, physiological requirements are specified in the form of:

$$C_{pre} \rightarrow C_{post}$$

in which C_{pre} is the physiological conditions without the device, and C_{post} is the physiological condition with the device. For model-based closed-loop verification, C_{pre} is often in form of a set of constraints on patient parameters. As a special case, C_{pre} can equal to $true$, means that C_{post} should be satisfied under all possible conditions.

Physiological requirements are constraints on physiological behaviors. In [2] we

2.5 Monitors

To simplify the requirement, [2]

3 Physiological Abstraction rules

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. During the rule applications the transition groups incorporated with the models are also merged. These information maintains the physiological meanings of the transitions of the abstract models and are helpful to determine whether the models are appropriate for physiological requirements.

3.1 Rule 1: Convert Reentry Circuits to Activation Nodes

Within the conduction network of the heart, there can be multiple pathways between two locations, forming conduction loops. If the timing parameters of the tissue along the loop satisfy certain property, there can be scenarios in which an depolarization wave circling the circuit. The circuits are referred to as *Reentry Circuits*. Since the time interval for an activation wave to circle a reentry circuit is usually less than the intrinsic heart cycle length, the heart rate will be "'hijacked"' by the reentry circuit once the cycling is triggered, causing tachycardia. Reentry is the most common mechanism for tachycardia which can be modeled by our heart models [3].

The effect of reentry tachycardia is that activation signals coming out of the circuit with cycle length equals to the sum of conduction delays of the conduction paths forming the circuit. It is therefore reasonable to model a reentry circuit as a self-activation node with the self-activation range equal to the sum of conduction delays. For more complex structures with multiple circuits, the self-activation range will be the minimum of the shortest circuit to the maximum of the longest circuit. The detailed rule description and implementation can be found in

3.2 Rule 2: Remove Irrelevant Structures

The network of node and path automata can be viewed as a graph,with nodes as vertices, paths as edges with conduction delay as weight. After the loops within the topology are removed, the topology of the heart model is in form of tree. Within the network there are certain nodes that are more important in terms of model behaviors, we denote them as *Nodes of Interests*, which include:

– Nodes with self-activations
– Nodes which interact with the pacemaker

Graph algorithm can be performed on the heart model to identify the core structure. Shortest paths can be calculated among nodes of interests. All the nodes and paths along the shortest paths are regarded as core structure. All the other nodes and paths can be then removed without affecting the behaviors of the model.

3.3 Rule 3: Removing Unnecessary Non-self-activation Nodes

The effect of non-self-activation nodes is blocking electrical events with interval shorter than its ERP period. If the self-activation nodes at both ends of a core path have self-activation interval longer than the maximum ERP period of nodes along the core path, the nodes can be removed.

For a core path from a self-activation node N_1 to another core node N_2, for any structure $P_1 - N_n - P_2$ which N_n is a non-self-activation node, if $N_n.ERP_{max} < min(N_1.Rest_{min}, N_2.Rest_{min})$, replace $P_1 - N_n - P_2$ with P_3 so that:

$$P_3.cond_{min} = P_1.cond_{min} + P_2.cond_{min}$$

$$P_3.cond_{max} = P_1.cond_{max} + P_2.cond_{max}$$

3.4 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 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

3.5 Rule 5: Merge Self-activation Nodes with Interaction Nodes

The effect of self-activation nodes on the interaction of the pacemaker is triggering sensing events within certain delay. In this rule we merge all the self-activation nodes to their neariest interaction nodes. If there exists multiple self-activation nodes merging to the same interaction node, the parameters of the new model are determined following Rule 3.

3.6 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:

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

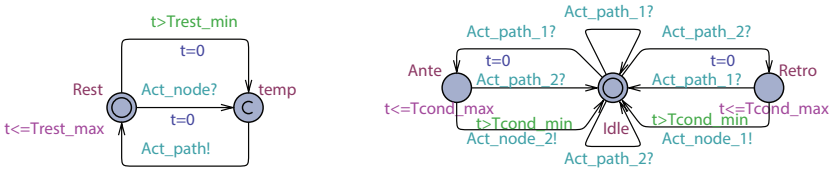
$$P.cond_{max} = P_1.cond_{max} + P_2.cond_{max}$$

For any self-activation node N:

what is N_1 and what is replacing? what is N

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

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



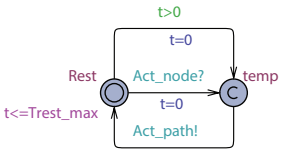
(a) Node automaton N1 (b) Path automaton P1

Fig. 5. Heart Model Abstractions

3.7 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.



Node automaton N2

Fig. 6. Heart Model Abstractions

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.

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

The 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 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 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. 8). 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

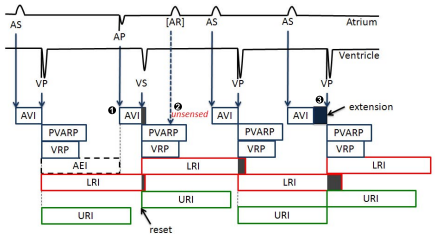


Fig. 7. Heart Model Abstractions

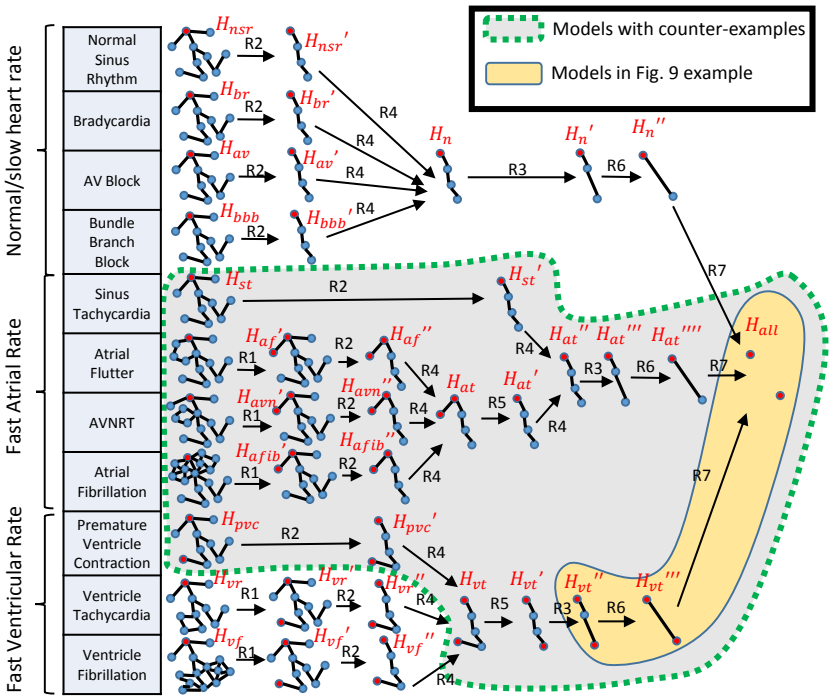


Fig. 8. Heart Model Abstractions

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. 8). 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

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

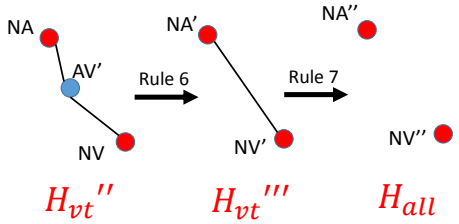


Fig. 9. Abstraction Example

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

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.

5.3 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

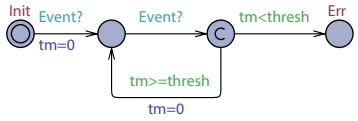


Fig. 10. monitor

With behavior mapping and a monitor (Fig. 10), 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 $EnvB(Req1) = NA.self$.

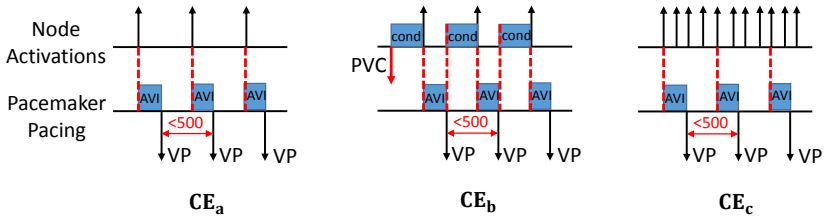


Fig. 11. Counter-examples

5.4 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. 9) 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 $Req1$.

5.5 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

that can trigger the violations 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. 11. 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 and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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