

Closed-loop Verification of Medical Devices with Model Abstraction and Refinement*

Zhihao Jiang, Miroslav Pajic, Rajeev Alur and Rahul Mangharam
University of Pennsylvania, Philadelphia PA, USA

Received: date / Revised version: date

Abstract. The design and implementation of software for medical devices is challenging due to the closed-loop interaction with the patient, which is a stochastic physical environment. The safety-critical nature and the lack of existing industry standards for verification, make this an ideal domain for exploring applications of formal modeling and closed-loop analysis. The biggest challenge is that the environment model(s) have to be both complex enough to express the physiological requirements, and general enough to cover all possible inputs to the device. In this effort, we use a dual chamber implantable pacemaker as a case study to demonstrate verification of software specifications of medical devices as timed-automata models in UPPAAL. The pacemaker model is based on the specifications and algorithm descriptions from Boston Scientific. The heart is modeled using timed automata based on the physiology of heart. The model is gradually abstracted with timed simulation to preserve properties. A manual Counter-Example-Guided Abstraction and Refinement (CEGAR) framework has been adapted to refine the heart model when spurious counter-examples are found. To demonstrate the closed-loop nature of the problem and heart model refinement, we investigated two clinical cases of *Pacemaker Mediated Tachycardia* and verified their corresponding correction algorithms in the pacemaker. Along with our tools for code generation from UPPAAL models, this effort enables model-driven design and certification of software for medical devices.

Key words: Medical Devices, Implantable Pacemaker, Software Verification, Cyber-Physical Systems, Model Abstraction and Refinement, CEGAR

1 Introduction

Over the past four decades, cardiac rhythm management devices such as pacemakers have expanded their role from “keeping the patient alive” to “improving the quality of the patient’s life”. The addition of more safety and efficacy features has resulted in increased complexity, inevitably leading to more potential safety issues. From 1996-2006, the percentage of software-related causes in medical device recalls have grown from 10% to 21% [1]. During the first half of 2010, the US Food and Drug Administration (FDA) issued 23 recalls of defective devices, all of which are categorized as Class I, meaning there is a “reasonable probability that use of these products will cause serious adverse health consequences or death.” At least six of the recalls were caused by software defects [2].

Medical devices, such as the implantable cardiac pacemaker, are perfect examples of Cyber-Physical Systems (CPS), in which the controller (the pacemaker) actively interacts with a stochastic plant (the heart). While in other CPS domains like the aviation and automotive industries, standards are enforced during software development, manufacturing, and post-market change [3,4], there are no well-established standards or tools for development of software for medical devices. One reason is because the software design in medical device industry is different from other industries. With physiological control systems, there is a large degree of uncertainty in the model of the organ and physiological process. The modes of operation vary across the population of patients, level of activities, metabolic rates, and so on. Thus, the safety and efficacy of the device should be evaluated in closed-loop based on the well-being of the patient, which relies on extensive domain knowledge on the physical environment. In model-based design, this unique feature results in two contradictory requirements on the environment model.

* This research was partially supported by NSF research grants MRI 0923518, CAREER 1253842, CNS 1035715 and CCF 0915777.

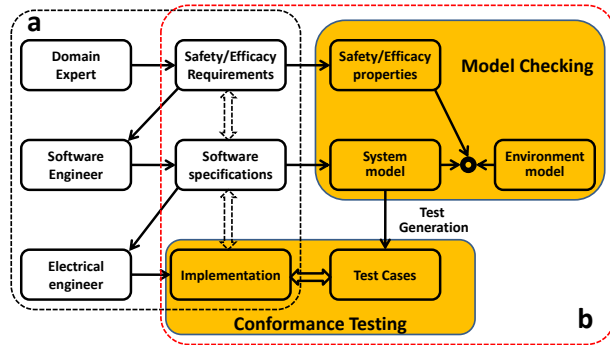


Fig. 1. a. Traditional real-time software development process; b. model-based design framework

1. The environment model has to be complex enough

Since the safety properties are specified based on the patient conditions, the environment model has to be able to represent specific patient conditions and condition groups, regardless of the capability of the device. When a certain algorithm is targeting a very specific patient condition, the environment model should be complex enough to distinguish that particular patient condition from other conditions.

2. The environment model has to be general enough

Unlike products in other industries, most medical device products have to be able to deal with large variety of environmental conditions. Thus the environment model used during safety evaluation has to be general enough to cover all possible scenarios.

The more complex the environment model is, there are usually more constraints on model outputs, which then reduce the coverage for the input to the device. It is contradictory that a single model can be both general and complex. One possible solution is to use multiple models with different complexity. When different models are used to prove different properties, we have to make sure that the models have certain relations so that properties verified with one model are preserved when we use other models.

1.1 Model-based Software Design Framework

In the current practice, medical device software is largely designed informally, as shown in Fig. 1.a. The domain expert, the physician in this case, first comes up with physiological safety/efficacy requirements which describe the objective of the device. Together with the software engineer, they specify the software specifications for the device, which explain how the device software would achieve the requirements. Then, the software engineer and electrical engineer convert the software specifications to physical implementation. However, due to the dis-

crepancies created during manual translations, there is no formal correlation from physiological requirements to specification and then to implementation. It is not guaranteed that the final implementation satisfies all physiological requirements. Thus, the safety and efficacy of the device cannot be ensured.

Fig. 1.b demonstrates how to establish formal correlations throughout the development process using model-based design framework. The software specification of the system is represented by a model. Together with a model of the environment, the closed-loop system is verified using model checking against safety properties which are converted from the safety/efficacy requirements. Satisfaction of the properties ensures that the software specifications satisfies the safety/efficacy requirements. Using automatic code generation, the specification as modeled, can be translated into C code and implemented on hardware. The same system model can be used to generate test cases for conformance testing of the implementation. Satisfaction of all the tests ensures that the implementation successfully conforms to the software specification. In this paper, we use an implantable cardiac pacemaker as an example to demonstrate the use of model checking to verify whether the pacemaker specification satisfies the safety/efficacy requirements.

1.2 Closed-loop Model Checking

Closed-loop model checking is a widely-used technique to formally verify the closed-loop system model against safety and efficacy properties. In this paper, we model the closed-loop system, which consists of the heart and a pacemaker, using networks of timed automata [5]. The closed-loop system model is then verified in model checker UPPAAL[6] against safety properties specified in Timed Computational Tree Logic (TCTL).

Safety properties are translated from physiological requirements learned from cardiac electrophysiology[7] and [8]. Intuitively the objective of a pacemaker is to maintain appropriate heart rate. Allowing too slow a heart rate or driving the heart rate too fast can cause serious adverse effect to the patient. Thus, it is essential to maintain appropriate heart rate within a safe range.

Unsafe conditions are specified as unsafe regions within the state space of the closed-loop system, and their reachability can be checked with a model checker. More importantly, there are closed-loop executions in which the pacemaker inappropriately raises the heart rate up to the boundary of the unsafe regions - even from healthy open-loop heart conditions. These unsafe executions are referred to as Pacemaker Mediated Tachycardia (PMT), during which the pacemaker incorrectly drives the heart into an overly fast rate.

We use these examples to show:

- Whether the anti-PMT algorithms developed by device manufacturers can successfully detect and terminate corresponding PMT.

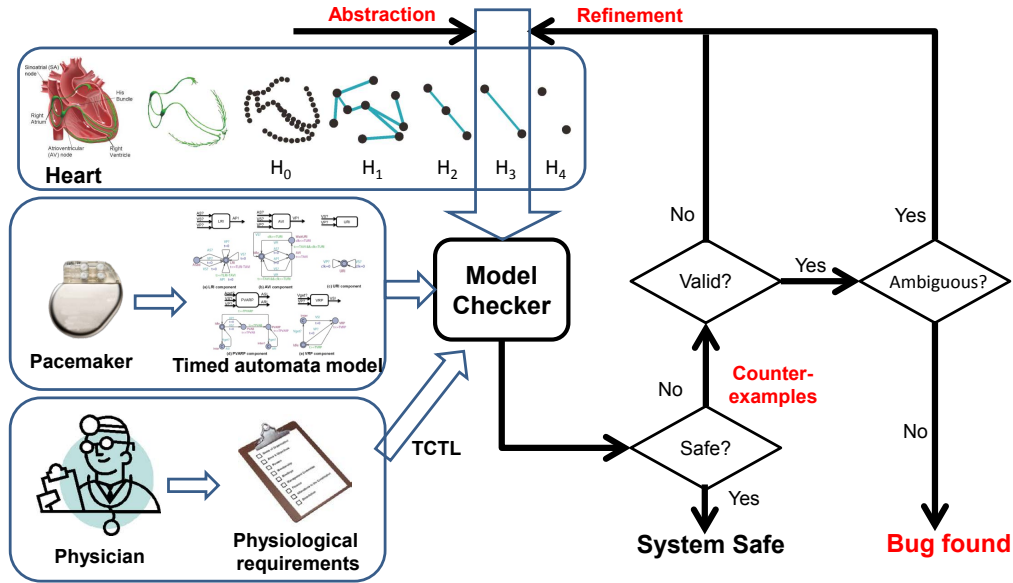


Fig. 2. Counter-Example-Guided Abstraction Refinement (CEGAR) framework

- After introducing these additional algorithms into the previously verified system, whether the combined system has unsafe states.
- Whether the environment model, the heart model in this case, is complex enough to distinguish PMT cases from healthy heart conditions, and at the same time be general enough to represent all possible heart conditions.

1.3 Model Abstraction and Refinement

Model abstraction [9] was originally proposed as a powerful state-reduction technique to alleviate the state explosion problem [10] during model-checking. Models are created as abstraction of the original system. Since the abstracted models have more behaviors than the original system, properties satisfied in an abstract model will also be satisfied in the original system. In this paper, we first derived a timed automata model of the heart tissue by extracting its timing behaviors. With a similar abstraction scheme used in clinical Cardiac Electro-physiology study [7], we then abstract the whole heart as electrical conduction network. The heart model is then further abstracted till its simplest form which covers all possible inputs to the pacemaker. During each abstraction step, we prove that the abstracted heart model and properties specified in ATCTL* are preserved.

Now we have a set of heart models from the cellular level to the whole heart, with abstraction relationship between each level. The question we aim to answer is: What do we determine which environment model to use during verification for each of the different safety

properties for the closed loop system? We base our approach on the Counter-Example-Guided Abstraction Refinement (CEGAR) framework, proposed by Clarke et al. [11], which can systematically and automatically handle model complexity.

In this work, we apply and extend the CEGAR framework to handle the complexity of the heart model during verification of a dual chamber pacemaker model. The framework is shown in Fig. 2. The most abstract heart model is first combined with a timed-automata-based pacemaker model [12] based on the algorithm descriptions from Boston Scientific [13]. The closed-loop system is then verified in model checker UPPAAL [6] against safety properties (e.g. the pacemaker must not allow the heart rate to be too slow and must not drive the heart rate too fast). If the unsafe regions are not reachable and unsafe executions do not exist, the system is safe. Otherwise the model checker returns execution traces as counterexamples which are checked in a more refined heart model for validity.

There are two scenarios where we use more refined heart model for model checking:

- (1) If the trace is not a valid heart-pacemaker interaction, it is referred to as a **spurious counterexample**. In this case we follow the traditional CEGAR framework. The property is checked with a more refined model until the spurious behavior is eliminated.
- (2) If the trace corresponds to a realistic heart-pacemaker interaction, we check further whether the returned traces only contain unsafe conditions that we are looking for. If the traces returned contains other heart conditions, the counter-example is referred to as **ambiguous**. In this case the property is checked with a more refined model

until the ambiguity is removed. Otherwise we confirm a potential bug has been found.

In this paper, we use two Pacemaker Mediated Tachycardia (PMT) (i.e. when the pacemaker drives the heart into an unsafe state) cases as example to demonstrate the two scenarios which require heart model refinement. By using the improved CEGAR framework we are able to check a wide range of physiological properties. The framework also improved the model checking efficiency without losing accuracy. The heart model abstraction enables model checking for more complex systems and the model refinement enables us to check more complex properties. In contrast to the CEGAR framework proposed by Clarke et al., the model abstraction, model refinement and trace validity check are done manually and can be potentially automated in the future work.

1.4 Contributions

This paper builds upon our previous work in [12] in which we described formal modeling of the pacemaker and verification of closed-loop interactions with a simple Random Heart Model and its ad-hoc refinements. The contributions of this paper are thus:

1. We abstracted the heart from top-down till its simplest form, which allows us to verify wide range of physiological properties.
2. The heart model abstraction is formalized and the timed simulation relation between each abstraction level are established and proved.
3. We improved the Counter-Example-Guided Abstraction and Refinement (CEGAR) framework to balance model complexity and expressiveness.
4. The safety properties are checked for all possible combinations of pacemaker parameters.

The UPPAAL models developed in this paper are available online [14]. These models can be used as a starting point for many purposes (e.g. to build models with costs and probabilities for quantitative analysis of the efficacy of pacemaker algorithms; development of patient-specific algorithms). In particular, the verified pacemaker model can be automatically translated from UPPAAL into Stateflow charts in Simulink for test generation and code generation using the UPP2SF tool [15].

1.5 Organization

The paper is organized as follow: In Section 2, we introduce the basic description for the heart and pacemaker operation. In Section 3, we describe the timed automata model of the heart and its abstractions. In Section 4, we describe the timed automata model of a dual chamber pacemaker. In Section 5, we verify the pacemaker model against two basic safety properties in UPPAAL; In Section 6 and 7, we use model checking to identify

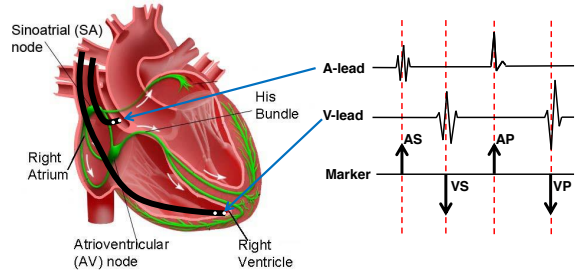


Fig. 3. (a) Electrical Conduction System of the Heart and pacemaker leads location. (b) Electrical signal sensed from the pacemaker leads are converted to event markers (AS,VS). Pacemaker delivers electrical pacing (AP,VP) from corresponding leads when heart rate is slow.

two known cases of PMT, in which the pacemaker inappropriately increases the heart rate. We evaluate the safety and effectiveness of two anti-PMT algorithms corresponding to the two common PMT cases, and demonstrate two scenarios for heart model refinements.

2 Overview of the Heart's Electrical Conduction System

In this section, we review basic concepts related to the heart and its electrical conduction system which interacts with the pacemaker. In-depth material of cardiac electro-physiology can be found in [16].

2.1 Basics of Cardiac Electrophysiology Operation

The coordinated contraction of the heart is governed by its Electrical Conduction System (see Fig. 3). The Sinoatrial (SA) node, which is a collection of specialized tissue at the upper right atrium, periodically spontaneously generates electrical pulses that can cause muscle contraction. The SA node acts as the natural pacemaker of the heart. The electrical pulses first cause both atria to contract, forcing the blood into the ventricles. The electrical conduction is then delayed at the Atrioventricular (AV) node, allowing the ventricles to fill fully. Finally, the fast-conducting His-Pukinje system spreads the electrical activation within both ventricles, causing simultaneous contraction of the ventricular muscles, and pumps the blood out of the heart. The electrical conduction system of the heart is a timed system and appropriate timing is key to proper heart rhythm.

Due to various factors such as aging and disease, the conduction properties of heart tissue may change. These changes often cause timing anomalies in heart rhythm, thus decreasing the blood pumping efficiency of the heart. These timing anomalies are referred to as arrhythmias, and 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. Bradycardia is due to failure of impulse generation with anomalies in the SA node, or failure of impulse propagation where the conduction from atria to the ventricles is delayed or blocked.

2.2 Interfacing the Heart with the Pacemaker

Heart tissue can also be activated by external electrical pulses. **Implantable Pacemakers** have been developed to deliver timely electrical pulses to the heart to maintain an appropriate heart rate and Atrial-Ventricular synchrony. Implantable pacemakers normally have two leads fixed on the wall of the right atrium and the right ventricle respectively (Fig. 3 (a)). These leads are capable of both sensing electrical activity in the heart tissue and are able to emit pacing signals into the tissue. Tissue activation near the leads is sensed by the leads, triggering an Atrial Sense (AS) and Ventricular Sense (VS) events in the pacemaker (see Fig. 3 (b)). Atrial Pacing (AP) and Ventricular Pacing (VP) are delivered if no sensed events occur within pre-specified deadlines. A dual chamber pacemaker only utilizes activation timing information of two small regions in the heart. The “low-resolution” sensing of the pacemaker results in its limited knowledge of current heart condition thus could lead to potential incorrect estimation of the heart’s electrical activity and result in inappropriate therapies.

In order to deal with different heart conditions, modern pacemakers can be programmed to operate in different modes. The modes are labeled using a three character system (e.g. DDD) by the Heart Rhythm Society [16]. The first character describes the pacing locations (i.e. atrium or ventricle or both), the second character describes the sensing locations, and the third character describes how the pacemaker software responds to sensing. For example, the dual-chamber DDD mode stands for sensing in both atrium and ventricle, and pace both of them if needed. In this effort, we describe two most commonly used modes of pacemaker, the dual-chamber DDD mode, that paces both the atrium and the ventricle, senses both chambers, and sensing can both activate or inhibit further pacing. Similarly, the VDI mode paces only in the ventricle, senses both chambers, and inhibits pacing if event is sensed [8]. During certain heart condition changes, the pacemaker has to switch between different modes to achieve better treatment. It is very important to ensure that the mode-switches are performed as intended, and no safety issues can occur during the transition between different modes.

3 Heart Modeling and Abstraction

In this section, we model the timing behaviors of the heart using timed automata [5]. We first model the heart

tissue at cellular level. Through a series of abstractions we end up with a series of heart model with complexity from maximum to the minimum. By proving the timed simulation relationship between each pair of heart model abstractions, the properties proved in the abstract level can be preserved. By specifying the safety properties with a universal subset of Timed Computational Tree Logic (ATCTL)[17], a model checker like UPPAAL [6, 18] can return counter-examples upon violations of the properties. This enables us to balance model complexity and property expressiveness using the Counter-Example-Guided Abstraction and Refinement (CEGAR) framework.

3.1 Timed Automata and Timed Simulation

Timed automata [5] are an extension of a finite automaton with a finite set of real-valued clocks. It has been used for modeling and verifying systems which are triggered by events and have timing constraints between events. UPPAAL is a standard tool for modeling and verification of real-time systems, based on networks of timed automata. The graphical and text-based interface makes modeling more intuitive. Requirements can be specified using Computational Tree Logic (CTL) [19] and violations can be visualized in the simulation environment.

3.1.1 Syntax of Timed Automata

A timed automaton \mathbf{G} is a tuple $\langle S, S_0, \Sigma, X, inv, E \rangle$, where

- S is a finite set of locations.
- $S_0 \in S$ is the set of initial locations.
- Σ is the set of events.
- X is the set of clocks.
- inv is the set of invariants for clock constraints at each location.
- E is the set of edges. Each edge is a tuple $\langle s, \sigma, \Psi, \lambda, s' \rangle$ which consists of a source location s , an event $\sigma \in \Sigma$, clock constraints Ψ , λ as a set of clocks to be reset and the target location s' .

For the clock variables X , the clock constraints $\Psi \in \Psi^X$ can be inductively defined by $\Psi := x \perp c \parallel \Psi_1 \wedge \Psi_2$, where $\perp \in \{\leq, =, \geq\}$, and $c \in \mathbb{N}$.

3.1.2 Semantics of Timed Automata

A state of a timed automaton is a pair $\langle s, v \rangle$ which contains the location $s \in S$ and the valuation v for all clocks. The set of all states is Ω . For all $\lambda \in X$, $v[\lambda := 0]$ denotes the valuation which sets all clocks $x \in \lambda$ as zero and the rest of the clocks unchanged. For all $t \in \mathbf{R}$, $v + t$ denotes the valuation which increase all the clock value by t . There are two kinds of transitions between states.

The discrete transition happens when the condition of an edge has been met. So we have:

$$\begin{aligned} \langle s, \sigma, \Psi, \lambda, s' \rangle \in E, v \models \Psi, v[\lambda := 0] \models \text{inv}(s') \\ \Rightarrow (s, v) \xrightarrow{\sigma} (s', v[\lambda := 0]) \end{aligned}$$

The timed transition happens when the timed automaton can stay in the same location for certain amount of time. We have:

$$\begin{aligned} \delta \in R, \forall \delta' \leq \delta, v + \delta' \models \text{inv}(s) \\ \Rightarrow (s, v) \xrightarrow{\delta} (s, v + \delta) \end{aligned}$$

3.1.3 Timed Simulation

For two timed automata $T^1 = \langle S^1, S_0^1, \Sigma^1, X^1, \text{inv}^1, E^1 \rangle$ and $T^2 = \langle S^2, S_0^2, \Sigma^2, X^2, \text{inv}^2, E^2 \rangle$, a timed simulation relation is a binary relation $\text{sim} \subseteq \Omega^1 \times \Omega^2$ where Ω^1 and Ω^2 are sets of states of T^1 and T^2 . We say T^2 time simulates T^1 ($T^1 \preceq_t T^2$) if the following conditions holds:

- Initial states correspondence: $(\langle s_0^1, \mathbf{0} \rangle, \langle s_0^2, \mathbf{0} \rangle) \in \text{sim}$
- Timed transition: For every $(\langle s_1, v_1 \rangle, \langle s_2, v_2 \rangle) \in \text{sim}$, if $\langle s_1, v_1 \rangle \xrightarrow{\delta} \langle s_1, v_1 + \delta \rangle$, there exists $\langle s_2, v_2 + \delta \rangle$ such that $\langle s_2, v_2 \rangle \xrightarrow{\delta} \langle s_2, v_2 + \delta \rangle$ and $(\langle s_1, v_1 + \delta \rangle, \langle s_2, v_2 + \delta \rangle) \in \text{sim}$.
- Discrete transition: For every $(\langle s_1, v_1 \rangle, \langle s_2, v_2 \rangle) \in \text{sim}$, if $\langle s_1, v_1 \rangle \xrightarrow{\sigma} \langle s_1', v_1' \rangle$, there exists $\langle s_2', v_2' \rangle$ such that $\langle s_2, v_2 \rangle \xrightarrow{\sigma} \langle s_2', v_2' \rangle$ and $(\langle s_1', v_1' \rangle, \langle s_2', v_2' \rangle) \in \text{sim}$.

Certain properties are preserved for timed simulation relation. For $\varphi \in ATCTL$, if $M \preceq_t M'$, we have $M' \models \varphi \Rightarrow M \models \varphi$. [17] However, $M' \not\models \varphi \Rightarrow M \not\models \varphi$ does not hold. Violations of *ATCTL* yield counter-examples and the validity of which need to be checked on more refined model.

It is known that timed simulation relation is also closed under composition [17]. So when we have two heart models $H_1 \preceq_t H_2$ we will have $H_1 \parallel P \preceq_t H_2 \parallel P$ where P is the timed-automata model of the pacemaker. For $\varphi \in ATCTL$, we have $H_2 \parallel P \models \varphi \Rightarrow H_1 \parallel P \models \varphi$. With this property we can verify the pacemaker model with abstract heart model. In the rest of the section, we will describe how we develop our initial heart model from the physiological perspective and abstract the model step by step so that the complexity of the model is reduced for verification. Given two heart models H_1, H_2 and a timed simulation mapping $\text{sim} = \Omega_1 \times \Omega_2$, there are no automated methods to check $H_1 \preceq_t H_2$. In the Appendix, we show the manual proof for the timed simulation relation between two heart models H_2 and H_3 . Other timed simulation relations can be proved similarly.

3.2 Initial Tissue Model

The initial model of the heart abstracts the electrical behaviors of the heart tissue. When the tissue is activated

by an external electrical signal, the voltage across the tissue increases rapidly. When the voltage reaches a certain threshold, the adjacent tissue will be activated. This activation then propagates throughout the heart, causing coordinated contraction of the heart muscles. Fig. 4 (a) shows the voltage change over time across a heart tissue region (solid line) and its adjacent tissue region (dotted line). After the activation, the heart tissue must recover from the activation (i.e. recharge) before it can be activated again. The time interval before the tissue voltage drops back to resting voltage is referred to as the Refractory Period. The refractory period can be divided into the Effective Refractory Period (ERP) and the Relative Refractory Period (RRP) according to their different response to new activation signals [7] [16].

We construct our initial heart model by modeling the timing behaviors of the heart tissue. The timed automaton N_0 is shown in Fig. 4 (b). The tissue starts from the Rest location. The tissue can self-activate after certain period and go to *cond* location which models the short delay before it activates the adjacent tissue. After the *cond* location the model goes to ERP location during which no activation events can be processed. The duration of the *cond* location and ERP location depend on the state of the tissue when it was activated by the previous activation signal. In this model, we use non-determinism to cover all possible duration length. After the ERP finishes, the model goes from the ERP location to the RRP location. If the heart tissue receives activation when it is in the RRP location, it will go to the ERP location with prolonged ERP length, which is covered by non-determinism. The tissue then goes back to the Rest location after the RRP period.

Consider the *Act_node?* event as input and *Act_next!* as output; N_0 covers all possible timing behaviors of a heart tissue. The whole heart can be modeled by composing tissue models with different parameters $H_0 = N_0^1 \parallel N_0^2 \cdots N_0^n$. However, for a real heart the number n is very large and their connectivity and parameter values are difficult, and perhaps impossible, to determine, which makes verification with H_0 infeasible. For the remainder of the section, please refer to both Fig. 4 along with Fig. 5 for progressive abstraction of the heart model. The abbreviations for the locations in Fig. 5 are shown in the table below. The figures are more understandable in color.

3.3 Abstraction 1: Model Conduction Delay with Path

During the Electro-physiology testing procedure [16], the physician places catheters with multiple electrodes into different locations of the patient's heart to perform a "timing analysis" of the propagation of electrical signals through heart. The local electrical activity of heart tissue at the locations of the electrodes can be monitored. The physician then evaluates the heart condition by examining the refractory properties of local tissue, and the

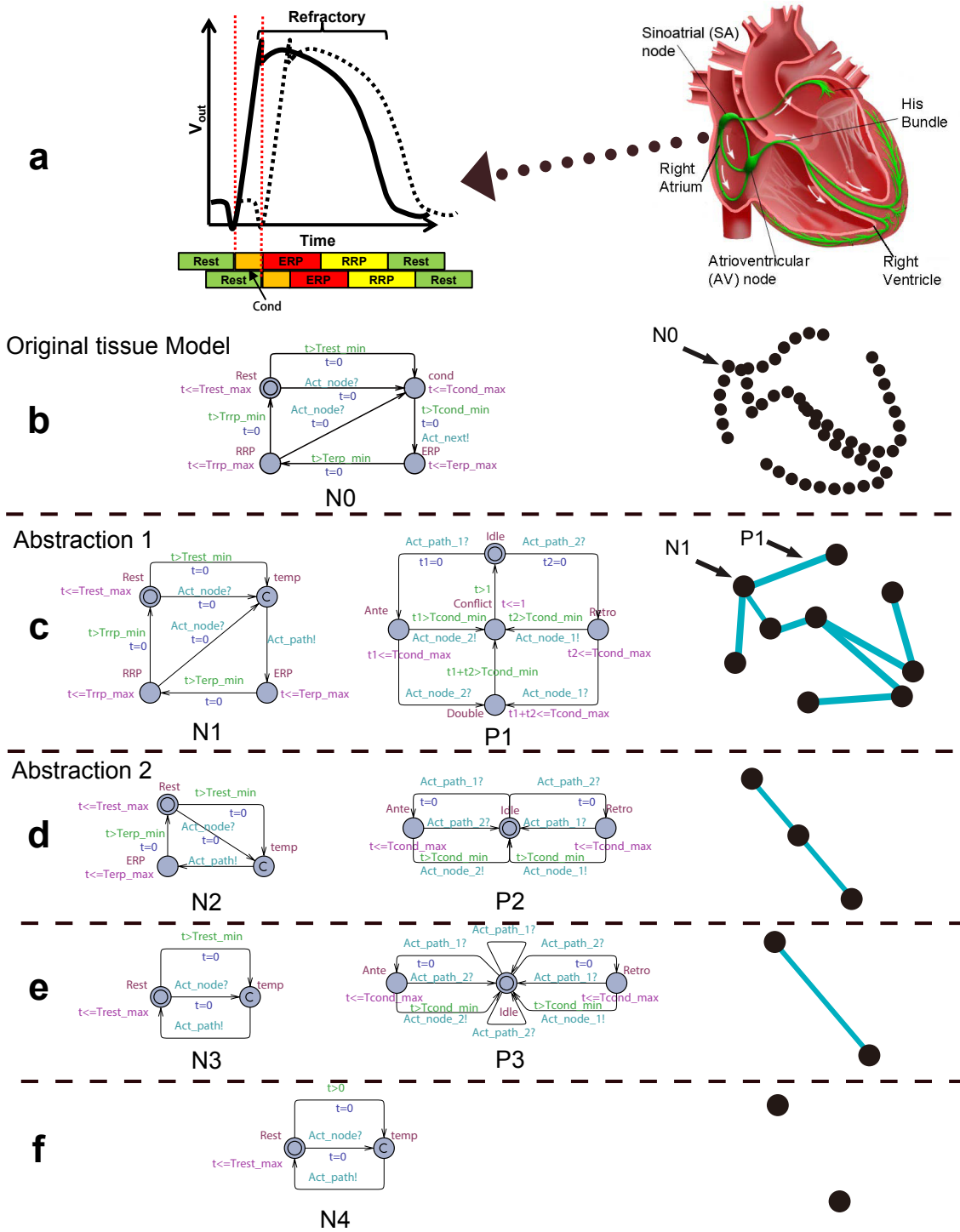


Fig. 4. (a) Electrical voltage change on the surface of the heart tissue and its adjacent tissue region (dotted) upon activation. The whole process is divided into timing periods with different behaviors. (b) The original tissue model which captures the timing behaviors of the heart tissue. (c) The conduction property is separated to the path automaton and the heart can be modeled as conduction network. (d) Equivalent locations are merged. (e) The blocking property of the ERP location is replaced by a non-deterministic transition in the path automaton. (f) The correlation between the two nodes is replaced by allowing the node automaton to have more behaviors.

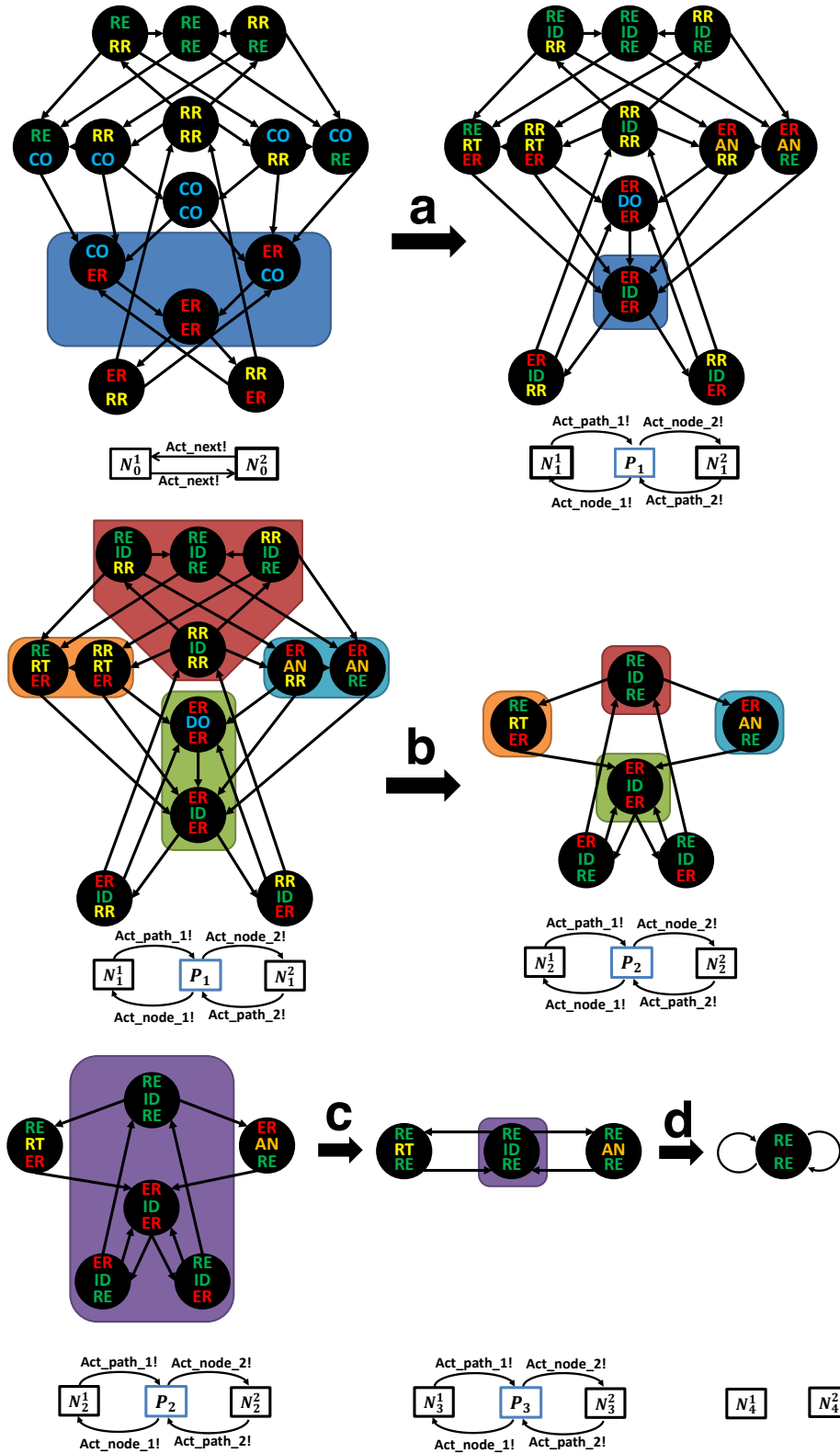


Fig. 5. (a) Model conduction delay between nodes as path; (b) Merging RRP with Rest; (c) Replace blocking property of ERP with non-deterministic conduction; (d) Use self activation to replace conduction

Location	Abbreviation	Description
ERP	ER	Effective Refractory Period
RRP	RR	Relative Refractory Period
Rest	RE	Rest Period
Cond	CO	Conduction Period
Idle	ID	No Conduction
Ante	AN	Antegrade Conduction
Retro	RT	Retrograde Conduction
Double	DO	Both Way Conduction

Table 1. Abbreviations used in heart models

timing delays for an electrical activation to conduct from one electrode to another. Due to the limited number of electrodes that can be placed into the heart, the physician can examine the refractory properties at the locations of the electrodes, and only consider the conduction delays between the electrodes. The procedure give us the intuition to abstract the heart as a conduction network as shown in Fig. 4 (c).

3.3.1 Tissue level abstraction

At the tissue level, we model the conduction delay between node automata using a path automaton. So two adjacent node automata $N_0^1 \parallel N_0^2$ can be simulated by two node automata and a path automaton ($N_1^1 \parallel P_1 \parallel N_1^2$) (Fig. 4 (c)). In the new node automaton N_1 , location *cond* is replaced by a committed location *temp*. In *cond* location, N_0 does not have a transition for *Act_node?*, which is equivalent to the ERP location. So the time for the *cond* location is added to the new ERP location:

$$N_1.Terp_{max} = N_0.Terp_{max} + N_0.Tcond_{max}$$

$$N_1.Terp_{min} = N_0.Terp_{min} + N_0.Tcond_{min}$$

The path automaton P_1 models the conduction property of between N_0^1 and N_0^2 with

$$P_1.Tcond_{min} = \min(N_0^1.Tcond_{min}, N_0^2.Tcond_{min})$$

$$P_1.Tcond_{max} = \max(N_0^1.Tcond_{max}, N_0^2.Tcond_{max})$$

The location correspondence and abstraction are shown in Fig. 5 (a). A typical timed execution where a node N_0^1 got activated and activates the other node N_0^2 after delay is shown below:

$$RE \parallel RE \xrightarrow{Act_node_1?} CO \parallel RE \xrightarrow[\text{Act_next!}]{\delta \geq Tcond_{min}} ER \parallel CO$$

Correspondingly, in $N_1^1 \parallel P_1 \parallel N_1^2$, the path P_1 is immediately activated when the node N_1^1 is activated, and after the same conduction delay the node N_1^2 on the other end of the path is activated.

$$RE \parallel ID \parallel RE \xrightarrow[\text{Act_path_1!}]{Act_node_1?} ER \parallel AN \parallel RE \xrightarrow[\text{Act_node_2!}]{\delta \geq Tcond_{min}} ER \parallel CF \parallel ER$$

We use *Double* location in path automaton to model when both node automata are in *Cond* location. When

one of the nodes finishes conducting, it will send *Act_next!* event to the adjacent node and enter ERP location. Since the other node is in *Cond* or ERP location, there is no transition for the *Act_next!* event and the conduction is blocked. So the locations for $CO \parallel ER, ER \parallel CO, ER \parallel ER$ are abstracted as a single location $ER \parallel ID \parallel ER$. We claim that $N_0^1 \parallel N_0^2 \preceq_t N_1^1 \parallel P_1 \parallel N_1^2$ and the heart can be modeled as $H_1 = N_1^1 \parallel P_1^1 \parallel N_1^2 \cdots P_1^m \parallel N_1^n$. This model transformation does not simplify the heart model, but it provides an abstract conduction path of arbitrary length with node and path automata.

3.3.2 Conduction Path Abstraction

A node automaton can generate *Act_path!* events by self-activation, and prevent *Act_node!* event to trigger *Act_path!* during ERP. If the self-activation period is too long for a node automaton N_1^i , and its ERP is short enough not to block any *Act_node!* event, it can be removed from the model and the paths connect to it can be merged. A conduction path with $N^1 \parallel P^1 \parallel N^2 \parallel P^2 \parallel N^3$ can be abstracted into $N^4 \parallel P^3 \parallel N^5$ with the following assumptions:

$$\min(N^1.Terp_{min}, N^3.Terp_{min}) \geq N^2.Terp_{max}$$

$$\max(N^1.Trest_{max}, N^3.Trest_{max}) < N^2.Trest_{min}$$

The first assumption infers that when the location of N^3 changes from ERP to RRP, it is guaranteed that $N_1^1.ERP \ \&\& \ N_1^2.ERP$. Under this assumption N^2 won't block any activation signal travel from N^1 to N^3 and vice versa. The second assumption guarantees that the clock *t* of N^2 will be reset before reaching *Trest_min* in *Rest* location, so there will not be self-activation from N^2 . Under these two assumptions we can have $N^4 = N^1, N^5 = N^3, P^3.Tcond_{min} = P^1.Tcond_{min} + P^2.Tcond_{min}$

$P^3.Tcond_{max} = P^1.Tcond_{max} + P^2.Tcond_{max}$
This can be generalized for acyclic conduction paths of arbitrary length. The first assumption can be achieved by assigning node automata to the locations with longer ERP. Since the pacemaker only has two leads monitoring the timing delay between heart activation events, and the AV node has the longest ERP period, the heart structure can be abstracted with two nodes and a path $H_1^1 = N_1^1 \parallel P_1^1 \parallel N_1^2 \parallel P_1^2 \parallel N_1^3$, with the *Act_node* events from N_1^1 and N_1^3 as input to the pacemaker.

3.3.3 Resolve Non-determinism with Linear interpolation

Abstraction 1 can be used in verification but not testing due to the non-deterministic choices of parameters. It captures all behaviors of the heart tissue but is not able to simulate a particular behavior, especially over multiple heart cycles. In [20], we developed the Virtual Heart Model (VHM) which resolved the non-determinism in Abstraction 1 with linear interpolations and can be used

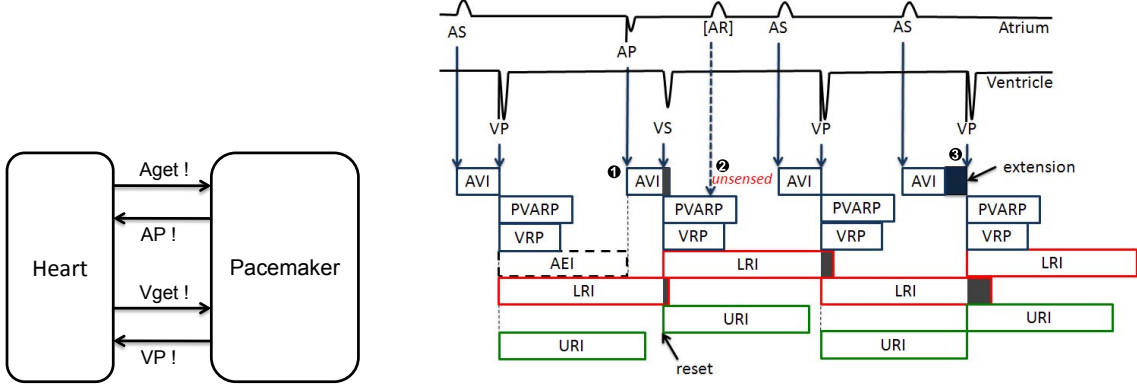


Fig. 6. (a) The heart model generates heart events (Aget,Vget) and responds to pacemaker events (AP,VP). (b) Basic 5 timing cycles of DDD pacemaker which maintain minimal heart rate, minimal A-V delay and filter noises

to mimic the behavior of different heart conditions. For example, the non-deterministic ERP range $[T_{min}, T_{max}]$ is interpolated as a linear function:

$$T_{min} + (1 - (1 - t/Trrp)^3) \cdot (T_{max} - T_{min}) \quad (1)$$

where t is the time since the node enters the RRP location.

3.4 Abstraction 2: Merging Equivalent Locations

The heart model H'_1 still has equivalent locations. In Abstraction 2 we further abstract the node and path automata. With non-determinism, the node automaton react to $Act_node!$ event the same way in RRP and Rest state. In the new node automaton N_2 we merge the RRP state with the Rest state with:

$$N_2.Trest_min = N_1.Trest_min + N_1.Trrp_min$$

$$N_2.Trest_max = N_1.Trest_max + N_1.Trrp_max$$

Since during ERP state, the node automata won't react to any $Act_node!$ event, for a $N_1^1 \parallel P_1 \parallel N_1^2$ setting, the symbolic state $ER \parallel DO \parallel ER$ is equivalent to $ER \parallel ID \parallel ER$. Under the assumption that the ERP period of a node automaton is much longer than the conduction delay of a path automaton, it is guaranteed that both node automata will be in ERP location when the path exits the Double location. In the path automaton, the Double and Conflict location is merged with the Idle state. The location abstraction and mapping is in Fig. 5 (b). The heart can be modeled as $H_2 = N_2^1 \parallel P_2^1 \parallel N_2^2 \parallel P_2^2 \parallel N_2^3$.

3.5 Abstraction 3: Replacing Blocking with Non-deterministic Conduction

In Abstraction 3, we replace the blocking behavior of the ERP location of the node with non-deterministic conduction in the path automaton. There exists a transition

$$RE \parallel ID \parallel RE \xrightarrow[Act_path.!]{Act_node.!?} RE \parallel ID \parallel RE$$

in $N_3^1 \parallel P_3^1 \parallel N_3^2$ to replace transition

$$ER \parallel ID \parallel ER \xrightarrow[Act_node.!?]{Act_node.!?} ER \parallel ID \parallel ER$$

in $N_2^1 \parallel P_2^1 \parallel N_2^2$

Without the ERP constraint the AV node is no longer needed and the heart can be modeled as $H_3 = N_3^1 \parallel P_3 \parallel N_3^2$. The location abstraction and mapping is in Fig. 5 (c). The detailed proof for this timed simulation relation is in the appendix.

3.6 Abstraction 4: Random Heart Model (RHM)

In Abstraction 4, we further simplify the heart model by removing the conduction path between two nodes. By setting $Trest_min$ for both nodes to 0 the new heart model $H_4 = N_3^1 \parallel N_3^2$ covers all possible behavior of H_3 . The location abstraction and mapping is in Fig. 5 (d). This random heart model with two nodes is the most abstract model that will be used at the beginning of the closed-loop verification process in Section 5.

Eventually we have a series of heart models with:

$$\begin{aligned} & N_0^1 \parallel N_0^2 \dots N_0^n \\ \preceq_t & N_1^1 \parallel P_1^1 \parallel N_1^2 \dots P_1^m \parallel N_1^n \\ \preceq_t & N_1^1 \parallel P_1^1 \parallel N_1^2 \parallel P_1^2 \parallel N_1^3 \\ \preceq_t & N_2^1 \parallel P_2^1 \parallel N_2^2 \parallel P_2^2 \parallel N_2^2 \\ \preceq_t & N_3^1 \parallel P_3 \parallel N_3^2 \\ \preceq_t & N_4^1 \parallel N_4^2 \end{aligned}$$

4 Pacemaker Modeling

In this section, we discuss our timed-automata model of the pacemaker. The overview of the closed-loop system is showed in Fig. 6(a). The interactions between the heart and the pacemaker are modeled by using binary event channels. The activation of N^1 of the heart is the input to the atrial lead

$$N^1.Act_path! \rightarrow Aget!$$

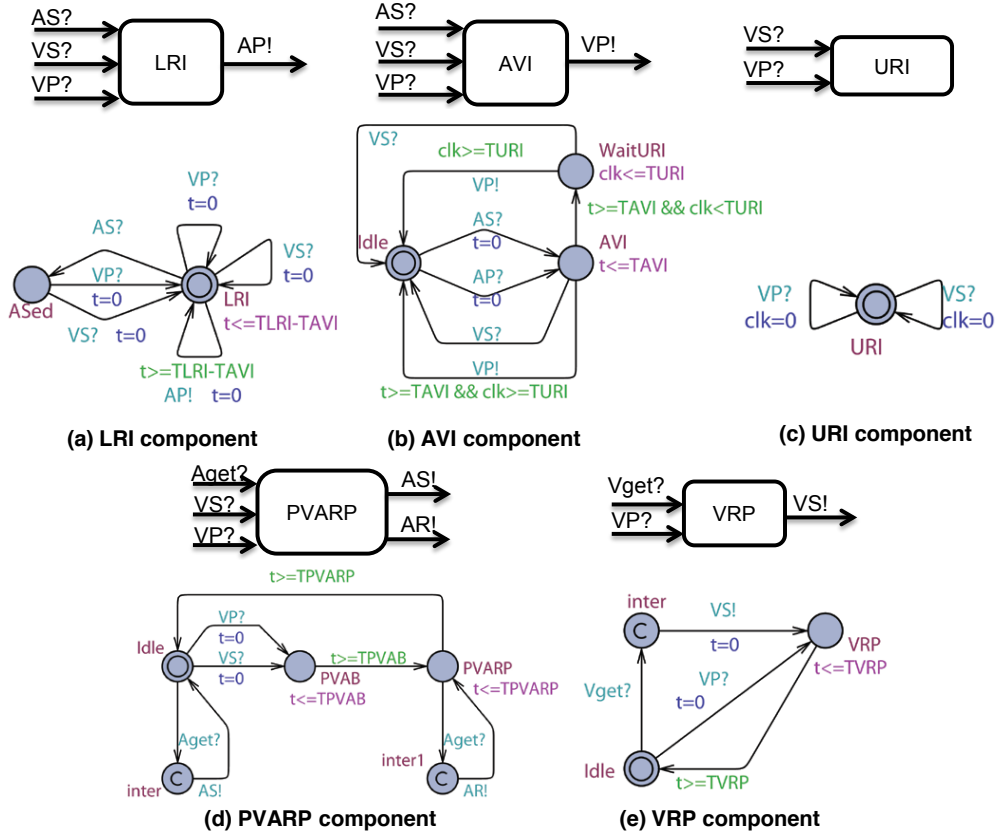


Fig. 7. (a)LRI component delivers AP! event if the V-A delay exceeds TLRI-TAVI; (b)AVI component delivers VP! if the A-V delay exceeds TAVI while the V-V delay is longer than TURI; (c)URI component keeps track of the V-V delay; (d)PVARP component filters certain Aget! from the heart and generates AS!; (e)VRP component filters certain Vget! from the heart and generates VS!

and for ventricular lead
 $N^2.Act_path! \rightarrow Vget!$

The pacemaker accordingly generates atrial or ventricular pacing actions

$AP! \rightarrow N^1.Act_node!$

$VP! \rightarrow N^2.Act_node!$

to the corresponding nodes in the heart. We now present our pacemaker model within the closed-loop heart-pacemaker system.

4.1 Basic DDD pacemaker modeling

The DDD pacemaker has 5 basic timing cycles triggered by events, as shown in Fig. 6(b). We decomposed our pacemaker model into 5 components which correspond to the 5 counters. $P = LRI \parallel AVI \parallel URI \parallel PVARP \parallel VRP$. These components synchronize with each other using broadcast channels and shared variables (as shown in Fig. 7).

4.1.1 Lower Rate Interval (LRI)

The Lower Rate Interval (LRI) component is shown in Fig. 7(a). This component defines the longest interval allowed between two ventricular events, thus keeps the heart rate above a minimum value. In DDD mode, the LRI interval is divided into a V-A interval (TLRI-TAVI) and a A-V interval (TAVI). The LRI component maintains a maximum V-A delay while the AVI component maintains a maximum A-V delay so together they maintain the maximum V-V delay. In the LRI component, the clock is reset when a ventricular event (VS, VP) is received. If no atrial event has been sensed (AS), the component will deliver atrial pacing (AP) after TLRI-TAVI. The UPPAAL design of LRI component is shown in Fig. 7(a).

4.1.2 Atrio-Ventricular Interval (AVI) and Upper Rate Interval (URI)

The function of the AVI component is to maintain the appropriate delay between the atrial activation and the ventricular activation. It defines the longest interval be-

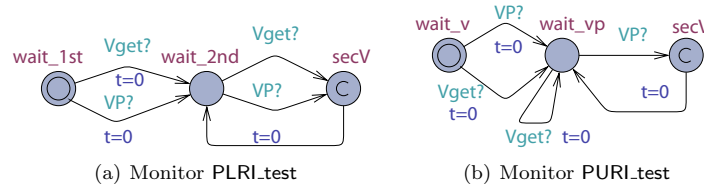


Fig. 8. (a) Monitor for LRL: Interval between two ventricular events should be less than TLRI, (b) Monitor for URL: Interval between a ventricular event and a VP should be longer than TURI

tween an atrial event and a ventricular event. If no ventricular event has been sensed (VS) within TAVI after an atrial event (AS, AP), the component will deliver ventricular pacing (VP). In order to prevent the pacemaker from pacing the ventricle too fast, a URI component uses a global clock clk to track the time after a ventricular event (VS, VP). The URI limits the ventricular pacing rate by enforcing a lower bound on the times between consecutive ventricle events. If the global clock value is less than TURI when the AVI component is about to deliver VP, AVI will hold VP and deliver it after the global clock reaches TURI. The UPPAAL design of AVI and URI component is shown in Fig. 7(b) and (c).

4.1.3 Post Ventricular Atrial Refractory Period (PVARP) and Post Ventricular Atrial Blanking (PVAB)

Ventricular events, especially Ventricular Pace (VP) are sometimes so strong that the atrial lead can sense the activation as well. This signal may be falsely recognized as an atrial event and disrupt normal pacemaker function. This scenario is called crosstalk and was discussed in our previous work [21]. In order to prevent this undesired behavior, there is a blanking period (PVAB) followed by a refractory period (PVARP) for the atrial events after each ventricular event (VS, VP). Atrial events during PVAB are ignored and atrial events during PVARP trigger AR! event which can be used in some advanced diagnostic algorithms. The UPPAAL design of PVARP component is shown in Fig. 7(d).

4.1.4 Ventricular Refractory Period (VRP)

Correspondingly, the VRP follows each ventricular event (VP, VS) to filter noise and early events in the ventricular channel which could otherwise cause undesired pacemaker behavior. Fig. 7(e) shows the UPPAAL design of VRP component.

4.1.5 Parameter Selection

In order to cope with the large variety of patient conditions, the timing parameters of the pacemaker can be programmed to discrete values. In our previous work [12], we picked a set of default values for the parameters and verified safety properties on it. In order to evaluate

the safety of the pacemaker software, it is important to check all possible parameter combinations. Since there are only a finite number (i.e. 8000) of different parameter combinations [13], in this paper we enumerate all of them and check all properties on each combination.

5 Reachability of Unsafe Regions

In this section, we define unsafe regions regarding bradycardia and tachycardia and specify two fundamental safety properties. These two fundamental safety properties are strict so that they must be satisfied by any pacemaker under all heart conditions.

5.1 Lower Rate Limit

The most essential function for the pacemaker is to treat bradycardia by maintaining the ventricular rate above a certain threshold. We define the region where the ventricular rate is slow, as *unsafe*. The monitor `PLRI_test` is designed to measure interval between ventricular events and is shown in Fig. 8(a). For property $\varphi_{LRI} = A[] (\text{PLRI_test.secV} \text{ imply } \text{PLRI_test.t} \leq \text{TLRI})$ we have $H_4 \| P \| \text{PLRI_test} \models \varphi_{LRI}$.

5.2 Upper Rate Limit

The pacemaker is not designed to treat tachycardia so it can only pace the heart to increase its rate and cannot slow it down. However, it is still important to guarantee it does not pace the ventricles beyond a maximum rate to ensure safe operations. To this effect, an Upper Rate Interval (URI) is specified such that the pacemaker can increase the ventricular rate up to this limit.

We require that a ventricle pace (VP) can only occur at least $TURI$ after a ventricle event (VS, VP). The monitor `PURI_test` is shown in Fig. 8(b). For the property $\varphi_{URI} = A[] (\text{PURI_test.secV} \text{ imply } \text{PURI_test.t} \geq \text{TURI})$ we have $H_4 \| P \| \text{PURI_test} \models \varphi_{URI}$.

Since φ_{LRI} and φ_{URI} belong to ATCTL*, the verified properties are preserved in the real heart.

In the following two sections, we discuss two closed-loop unsafe executions where the pacemaker inappropriately increases the heart rate. Such executions are referred to as Pacemaker Mediated Tachycardia (PMT).

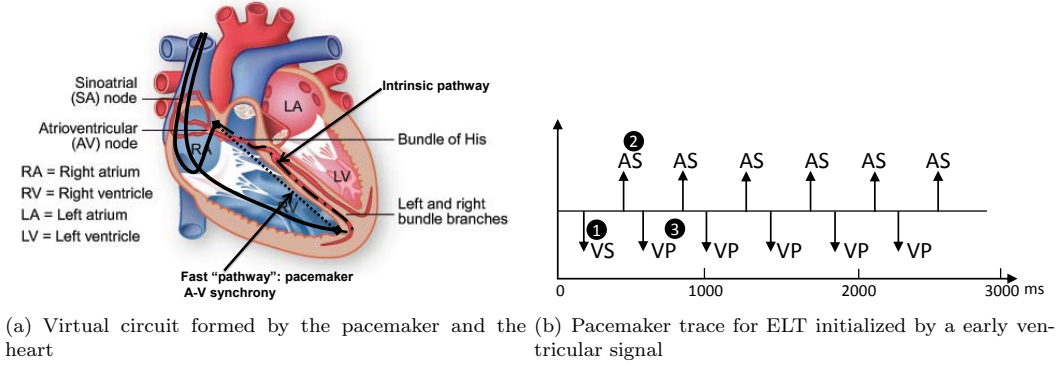


Fig. 9. Endless Loop Tachycardia case study demonstrating the situation when the pacemaker drives the heart into an unsafe state [22]

We use existential properties $P_E \in ETCTL^*$ to show the existence of these executions and use $\neg P_E$ to prove the correctness of the corresponding Anti-PMT algorithms. Since $\mathbf{AG}\varphi \equiv \neg\mathbf{EF}(\neg\varphi)$, the properties also belong to \mathbf{ATCTL}^* and are preserved for the real heart.

6 Endless Loop Tachycardia

The AVI component of a dual-chamber pacemaker introduces a virtual A-V conduction pathway. This forms a timing loop with the intrinsic (physiological) A-V conduction pathway (see Fig. 9(a)). A Premature Ventricular Contraction (PVC), i.e. an early extra beat in the ventricular, may trigger another ventricular event (VS) and initiate a V-A conduction through the intrinsic pathway (Marker 1 in Fig. 9(b)). The pacemaker registers this signal as an Atrial Sense (AS) (Marker 2 in Fig. 9(b)). This event triggers a VP after TAVI, as if the signal conducts through the “virtual” A-V pathway (Marker 3 in Fig. 9(b)). We call it “virtual” pathway as the “conduction” delay is fulfilled by a timer in the pacemaker instead of a physical signal propagation along the heart tissue. The VP will trigger another V-A conduction and this VP-AS-VP-AS looping behavior will continue (see Fig. 9(b)). The interval between atrial events is TAVI plus the V-A conduction delay, which is normally shorter than the delay between intrinsic heart beats, thus will drive the ventricular rate as high as the Upper Rate Limit. During ELT, the heart rate is not only high, but also fixed, which is unsafe scenario.

Due to the limited information the pacemaker has about the heart, the pacemaker cannot distinguish a retrograde atrial event from an intrinsic atrial event which is triggered by the SA node. From the pacemaker’s point of view, the pacemaker paces the ventricles as specified for every AS. That is why open-loop testing is unable to detect this closed-loop behavior.

Modern pacemakers are equipped with anti-ELT algorithms to identify and terminate potential ELT. One common algorithm identifies ELT by the ELT pattern

and terminates ELT by increasing TPVARP time once to block the AS caused by the V-A conduction. By increasing the blocking interval after a ventricular event, the pacemaker effectively ignores the early atrial signal detected due to the PVC.

6.1 Existence of ELT

Two monitors were designed to show the existence of ELT. One monitor, $PELT_det$, shows the persistence of the VP-AS pattern and the other monitor, Pvv , shows that the ventricular rate is always no slower than the upper rate limit (Fig. 10). The property $\varphi_{ELT} = \mathbf{E}[] ((\text{not } PELT_det.err) \ \&\& \ (\text{not } Pvv.err))$ checks the existence of ELT behavior. After our initial verification we have:

$$H_4 \parallel P \parallel PELT_det \parallel Pvv \models \varphi_{ELT}$$

The evidence trace shows the behavior of the system.

For simplicity we only show the state of the heart:

$$RE \parallel RE \xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node.2!}}{N^1.t \geq 0}$$

$$RE \parallel RE \xrightarrow{\frac{Act_path.1! \rightarrow AS!}{VP! \rightarrow Act_node.2!}}{N^1.t \geq 0}$$

$$RE \parallel RE \xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node.2!}}{N^1.t \geq 0}$$

$$RE \parallel RE \xrightarrow{\frac{Act_path.1! \rightarrow AS!}{VP! \rightarrow Act_node.2!}}{N^1.t \geq 0}$$

$$RE \parallel RE$$

6.2 Trace Validation and Heart Model Refinement

We check this trace on more refined $H_3 \parallel P$ and discovered that the trace above can correspond to two different scenarios:

$$RE \parallel AN \parallel RE \xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node.2!}}{N^1.t \geq N^1.Trest_min}$$

$$RE \parallel ID \parallel RE \xrightarrow{\frac{Act_path.1! \rightarrow AS!}{VP! \rightarrow Act_node.2!}}{N^1.t \geq N^1.Trest_min}$$

$$RE \parallel AN \parallel RE \xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node.2!}}{N^1.t \geq N^1.Trest_min}$$

$$RE \parallel ID \parallel RE \xrightarrow{\frac{Act_path.1! \rightarrow AS!}{VP! \rightarrow Act_node.2!}}{N^1.t \geq N^1.Trest_min} \parallel RE \parallel AN \parallel RE$$

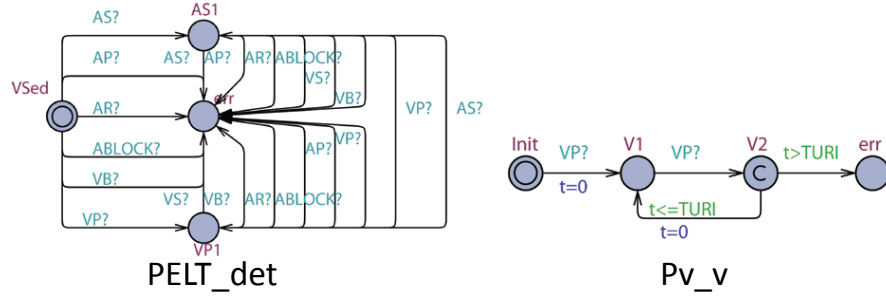


Fig. 10. (a) Any pattern other than VP-AS will result in error state (b) If the ventricular rate is slower than the Upper Rate Limit will go to error state

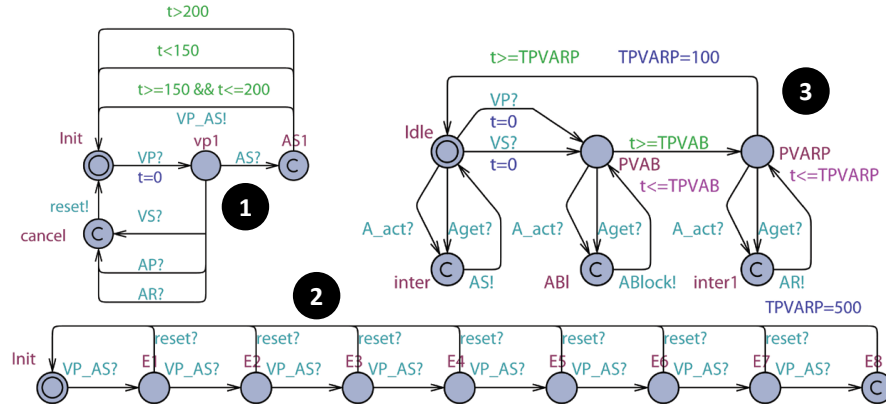


Fig. 11. (1) The component PVAS sends VP_AS! event when a VP-AS pattern with delay between [150,200] is detected; (2) Component ELTct. After 8 VP-AS pattern, the algorithm increase TPVAB to 500ms. (3) Modified PVARP' component. TPVAB can only be set to 500 for one timing cycle.

$$\begin{aligned}
 &\text{and} \\
 RE \parallel ID \parallel RE &\xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node_2!}} \\
 RE \parallel RT \parallel RE &\xrightarrow{\frac{P.t \geq P.Tcond_min}{Act_node_1! \rightarrow Act_path_1! \rightarrow AS!}} \\
 RE \parallel ID \parallel RE &\xrightarrow{\frac{AVI.t \geq TAVI \wedge URI.t \geq TURI}{VP! \rightarrow Act_node_2!}} \\
 RE \parallel RT \parallel RE &\xrightarrow{\frac{P.t \geq P.Tcond_min}{Act_node_1! \rightarrow Act_path_1! \rightarrow AS!}} \\
 RE \parallel ID \parallel RE &
 \end{aligned}$$

Both traces correspond to actual clinical scenarios. However, the second trace corresponds to the ELT behavior which inappropriately increased the heart rate. By setting $N^1.Trest_min \geq TURI$ in H_3 we can model the healthy heart and the first scenario will be eliminated. However, in H_4 the $Trest_min$ is set to 0 so the two cases cannot be distinguished. So we use the refined heart model H_3 with $N^1.Trest_min \geq TURI$. With the new heart model we have:

$$H_3 \parallel P \parallel PELT_det \parallel Pv_v \models \varphi_{ELT}$$

The evidence trace, which is exactly the second trace above, shows exactly the ELT scenario.

6.3 ELT termination algorithm

The ELT will persists without intervention and the patient's heart is forced to beat at a fast rate approaching the Upper Rate Limit. Thus, device manufacturers require a way to identify ELT and terminate it despite of the limited information the pacemaker can get. The ELT detection algorithm by Boston Scientific [23] utilizes these three features:

- Ventricular rate at Upper Rate Limit
- VP-AS pattern
- Fixed V-A conduction delay

The pacemaker first monitors VP-AS pattern with ventricular rate at upper rate limit. Then it compares the VP-AS interval with previous intervals. ELT is confirmed if the difference between the current VP-AS interval and the first VP-AS interval are within $\pm 32ms$ for 8 consecutive times. Then the pacemaker increases the PVARP period to 500ms once so that the next AS will be blocked and will not trigger a VP. ELT will then be terminated. As the V-A conduction delays are patient-specific, the algorithm compares the VP-AS interval to a previously sensed value instead of an absolute value. Since we can

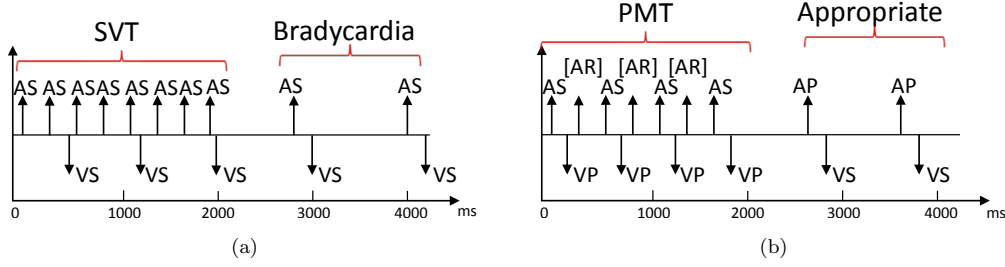


Fig. 12. (a) Open-loop: 3:1 A-V conduction during SVT and low ventricular rate without SVT; (b) With DDD pacemaker: the pacemaker paces the ventricle for every atrial sense (AS), thus increase the ventricular rate inappropriately

not store past clock values in UPPAAL, we can not explicitly model this ELT detection algorithm. However, since the conduction delay in our heart model is within a known range, we can compare the VP-AS interval with this range. The VP-AS pattern detection module *VPAS* for our anti-ELT algorithm is shown in Fig. 11 (1). It detects the VP-AS pattern with ventricular rate at the Upper Rate Limit and sends out a *VP_AS* event if the interval qualifies.

A counter *ELTct* counts the number of qualified VP-AS patterns. It increases the PVARP period to 500ms if eight consecutive VP-AS patterns are detected. (Fig. 11 (2)) The PVARP component is also modified so that the PVARP period can only be changed once by the anti-ELT algorithm. (Fig. 11 (3))

6.4 Verification of the algorithm:

With the new pacemaker model $P_1 = LRI \parallel AVI \parallel URI \parallel PVARP' \parallel VVP \parallel ELTct \parallel VPAS$ we first check whether the two fundamental safety properties still hold when the anti-ELT algorithm is introduced. We have

$$H_3 \parallel P_1 \parallel PLRI_{test} \models \varphi_{LRI}$$

$$H_3 \parallel P_1 \parallel PURI_{test} \models \varphi_{URI}$$

Then we check the existence of ELT and we have:

$$H_3 \parallel P_1 \parallel PELT_{det} \parallel Pvv \not\models \varphi_{ELT}$$

which indicates the algorithm successfully eliminated all ELT executions.

7 Verification of the Mode-Switch Algorithm

7.1 Supraventricular tachycardia (SVT)

SVT is an arrhythmia which features an abnormally fast atrial rate. Typically, in the open loop case, the AV node, which has a long refractory period, can filter most of the fast atrial activations during SVT thus the ventricular rate remains relatively normal. Fig. 12(a) demonstrates

a pacemaker event trace during SVT, with a ODO mode pacemaker which just sensing in both channels. In this particular case, every 3 atrial events (AS) correspond to 1 ventricular event (VS) during SVT. As an arrhythmia, SVT is still considered as a safe heart condition since the ventricles operate under normal rate can and still maintain adequate cardiac output.

However, in the closed loop case with the pacemaker, the AVI component of a dual chamber pacemaker is equivalent to a virtual pathway in addition to the intrinsic conduction pathway between the atria and the ventricles. The pacemaker tries to maintain 1:1 A-V conduction and thus increases the ventricular rate inappropriately to match the atrial rate. Fig. 12(b) shows the pacemaker trace of the same SVT case with DDD pacemaker. Although half of the fast atrial events are filtered by the PVARP period ([AR]s), the DDD pacemaker still drives the closed-loop system into 2:1 A-V conduction with faster ventricular rate, which is inappropriate. This problem can be resolved by switching the pacemaker from the dual chamber mode, which couples the atrial and ventricular rates, into a single chamber mode to maintain appropriate ventricular rate independent of the atrial rate.

7.2 Existence of PMT during SVT

The monitor *Pv.v* is designed to show existence of PMT during SVT. It goes to the error state if the ventricular rate drops below the Upper Rate Limit (Fig. 15).

We specify

$$\varphi_{MS} = E \parallel (not Pv.v.err)$$

which verifies the existence of PMT. To identify the PMT scenario, we first set $H_3.N^1.Trest_{min} < 100$ so that the atrial rate can be high and $H_3.N^2.Trest_{min} > TURI$ so that the intrinsic heart rate is less than TURI. The property is first verified on pacemaker without the mode-switch algorithm. We have

$$H_3 \parallel P \parallel Pv.v \models \varphi_{MS}$$

The property is satisfied and the evidence trace shows the behavior of the system.

$$RE \parallel ID \parallel RE \xrightarrow{N^1.t \geq N^1.Trest_{min}}_{Act_path_1! \rightarrow AS!}$$

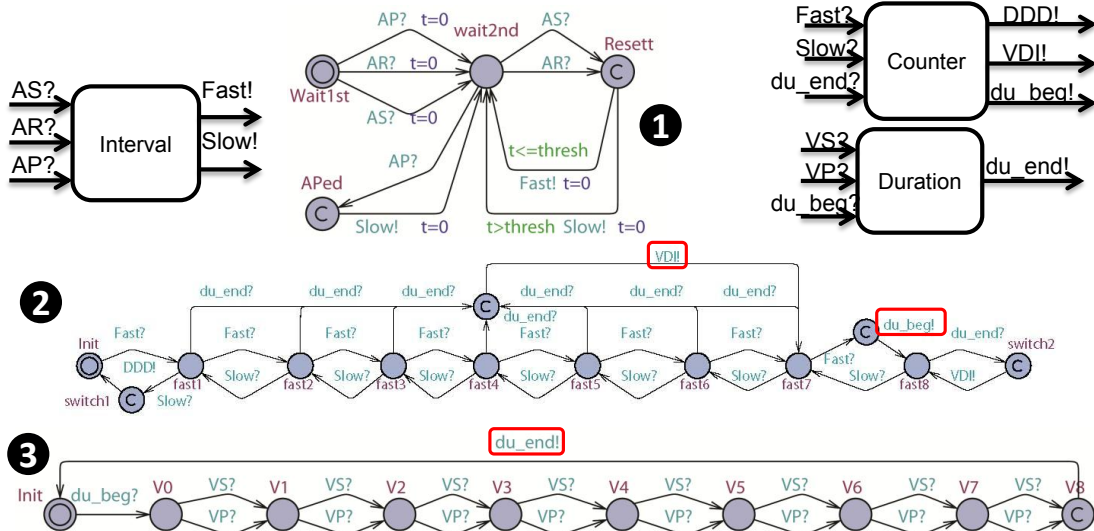


Fig. 13. (a) Component INT An atrial event (AS,AR) arrive before *thresh* after the previous atrial event is regarded as fast event. Atrial event arrive after *thresh* and AP are regarded as slow event; (b) Component CNT After 8 fast event the algorithm will start a duration by sending *du_beg* and will switch to VDI mode when the duration ends (*du_end*); (c) Component DUR The duration length is 8 ventricular events (VS,VP)

$$RE \parallel AN \parallel RE \xrightarrow[\text{Act_node.2!} \rightarrow \text{Act_path.1!} \rightarrow VS!]{P.t \geq P.Tcond_min}$$

$$RE \parallel ID \parallel RE \xrightarrow[\text{Act_path.1!} \rightarrow AS!]{N^1.t \geq N^1.Trest_min}$$

$$RE \parallel AN \parallel RE \xrightarrow[\text{Act_node.2!} \rightarrow \text{Act_path.1!} \rightarrow VS!]{P.t \geq P.Tcond_min}$$

$$RE \parallel ID \parallel RE$$

The time between two VS! events are less than TURI so the property holds. But that is not exactly the PMT case we are looking for.

7.3 Trace Validation and Heart Model Refinement

When a path P_3 receives a $Act_path.1!$ event, there are two non-deterministic transitions. $ID \xrightarrow{Act_path.1!} ID$ and $ID \xrightarrow{Act_path.1!} AN$. From the trace above we can see that every atrial event triggers the path conduction. When the trace is validated against $H_2 = N_2^1 \parallel P_2^1 \parallel N_2^2 \parallel P_2^2 \parallel N_2^3$ we have:

$$RE \parallel ID \parallel RE \parallel ID \parallel RE \xrightarrow[\text{Act_path.1!} \rightarrow AS!]{N^1.t \geq N^1.Trest_min}$$

$$ER \parallel AN \parallel RE \parallel ID \parallel RE \xrightarrow[\text{Act_node.2!} \rightarrow \text{Act_path.2!}]{P^1.t \geq P^1.Tcond_min}$$

$$ER \parallel ID \parallel ER \parallel AN \parallel RE \xrightarrow{N^1.t \geq N^1.Terp_min}$$

$$RE \parallel ID \parallel ER \parallel AN \parallel RE \xrightarrow[\text{Act_path.1!} \rightarrow AS!]{N^1.t \geq N^1.Trest_min}$$

$$ER \parallel AN \parallel ER \parallel AN \parallel RE \xrightarrow[\text{Act_node.2!}]{P^1.t \geq P^1.Tcond_min}$$

$$ER \parallel ID \parallel ER \parallel AN \parallel RE$$

The second $Act_node.2!$ is blocked by N_3^2 so it won't conduct to N_3^3 and trigger VS!. So it turns out to be a spurious counter-example. $ER \parallel AN \parallel ER \parallel AN \parallel RE$ is the

dead-end state where the trace cannot proceed in the refined model and $ER \parallel AN \parallel RE \parallel AN \parallel RE$ is the bad state which is in the same abstract state as the dead-end state and enables the trace to proceed in the abstract model. The heart model needs to be refined so that the bad state and the dead-end state are separated and the spurious counter-example is eliminated. So we refine the heart model from H_3 to H_2 with $H_3 \preceq_t H_2$. Our second verification effort shows that:

$$H_3 \parallel P \parallel Pv.v \models \varphi_{MS}$$

The property is satisfied and the evidence trace shows the behavior of the system.

$$RE \parallel ID \parallel RE \parallel ID \parallel RE \xrightarrow[\text{Act_path.1!} \rightarrow AS!]{N^1.t \geq N^1.Trest_min}$$

$$ER \parallel AN \parallel RE \parallel ID \parallel RE \xrightarrow[\text{Act_node.2!} \rightarrow \text{Act_path.2!}]{P^1.t \geq P^1.Tcond_min}$$

$$ER \parallel ID \parallel ER \parallel AN \parallel RE \xrightarrow{N^1.t \geq N^1.Terp_min}$$

$$RE \parallel ID \parallel ER \parallel AN \parallel RE \xrightarrow[\text{Act_path.1!} \rightarrow AS!]{N^1.t \geq N^1.Trest_min}$$

$$ER \parallel AN \parallel ER \parallel AN \parallel RE \xrightarrow[\text{Act_node.3!} \rightarrow VP!]{AVI.t \geq TAVI \wedge URI.t \geq TURI}$$

$$ER \parallel AN \parallel ER \parallel DO \parallel ER$$

In the trace, the pacemaker delivers VP for every atrial event, increasing the ventricular rate, which is exactly the PMT case.

7.4 Mode-Switch Algorithm

We now analyze the approach use in pacemakers to prevent prolonged ventricular pacing under SVT. Intuitively, the mode-switch algorithm first detects SVT. After confirmed detection, it switches the pacemaker from a dual-chamber mode to a single-chamber mode. During the

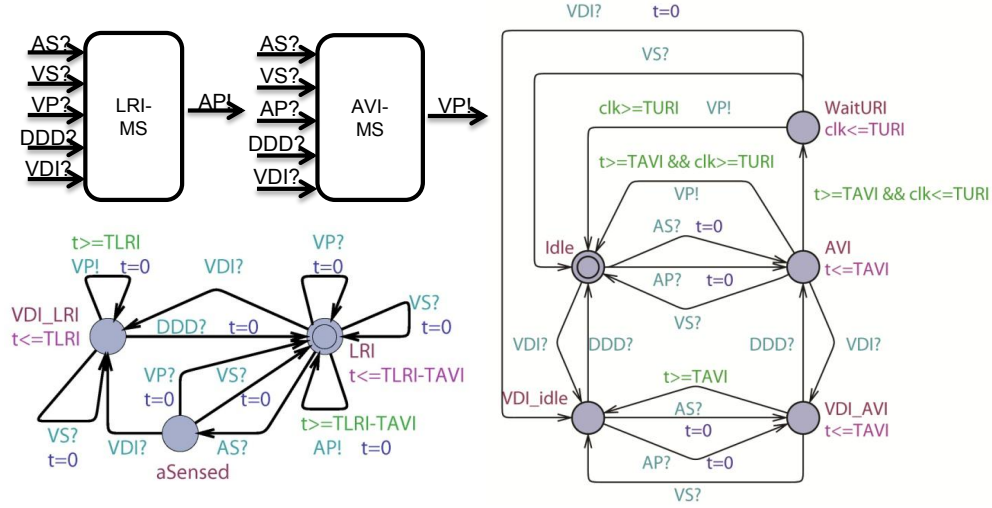


Fig. 14. (a) After switching to VDI mode, the new LRI component LRI' maintains a minimum V-V interval; (b) After switching to VDI mode, the new AVI component AVI' keeps track of the time after each atrial events.

single-chamber mode, the A-V synchrony function of the pacemaker is deactivated thus the ventricular rate is decoupled from the fast atrial rate. After the algorithm determines the end of SVT, it will switch the pacemaker back to the dual chamber mode.

The mode-switch algorithm specification we use is similar to the one described in the Boston Scientific pacemakers manual [13]. The algorithm first measures the interval between atrial events outside the blanking period (AS, AR). The interval is considered as *fast* if it is above a threshold (*Trigger Rate*) and *slow* otherwise. In our UPPAAL model we model it as *INT* (see Fig. 13 (1)). A counter *CNT* increments for *fast* events and decrements for *slow* events (see Fig. 13 (2)). After the counter value reaches the *Entry Count*, the algorithm will start a *Duration (DUR)* which is a time interval used to confirm the detection of SVT (see Fig. 13 (3)). In the *Duration*, the counter keeps counting. If the counter value is still positive after the *Duration*, the pacemaker will switch to the VDI mode (*Fallback mode*). In the VDI mode, the pacemaker only senses and paces the ventricle. At any time if the counter reaches zero, the *Duration* will terminate and the pacemaker is switched back to DDD mode. In our UPPAAL model of the mode-switch algorithm, we use nominal parameter values from the clinical setting. We define *trigger rate* at 170bpm (350ms), *entry count* at 8, *duration* for 8 ventricular events and *fallback mode* as VDI.

In order to model both DDD and VDI modes and the switching between them, we made modifications to the AVI and LRI components. In each component two copies for both modes are modeled, and switch between each other when switching events (DDD, VDI) are received. During VDI mode, VP is delivered by the LRI component instead of the AVI component. The clock values are

shared between both copies in order to preserve essential intervals even after switching. The modified AVI (*AVI'*) and LRI (*LRI'*) components are shown in Fig. 14. So the new pacemaker model is:

$$P_2 = \text{LRI}' \parallel \text{AVI}' \parallel \text{URI} \parallel \text{PVARP} \parallel \text{VRP} \parallel \text{INT} \parallel \text{CNT} \parallel \text{DUR}$$

7.5 Verification against fundamental safety properties

We verify the same fundamental safety properties on the pacemaker model with mode-switch algorithm. We have:

$$H_2 \parallel P_2 \parallel \text{PURI}_{test} \models \varphi_{URI}$$

$$H_2 \parallel P_2 \parallel \text{PLRI}_{test} \not\models \varphi_{LRI}$$

The Upper Rate Limit property still holds but the Lower Rate Limit property is violated. The counterexample is proved to be valid after checking the trace of more refined heart models. By analyzing the trace we found that when the pacemaker is switching from VDI mode to DDD mode, the responsibility to deliver VP switched from LRI component to AVI component. Since the clock reference is different (Ventricular events in LRI component and Atrial events in AVI component), the clock value for delivering the next VP is not preserved. As a result, if an atrial event which triggered the mode-switch from VDI to DDD happens within $[\text{TLRI}-\text{TAVI}, \text{TLRI}]$ after the last ventricular event, the next ventricular pacing will be delayed by at most TAVI time, which violates the Lower Rate Limit property (Fig. 16(a)).

7.6 Verification Result

After implementing the Mode-switch algorithm, we verified the model against the same existence property. We expect the violation of this property, since during VDI

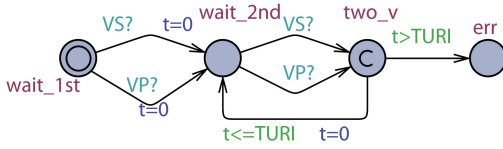


Fig. 15. Monitor Pv_v for SVT: There exists an endless sequence in which interval between ventricular events is at most TURI

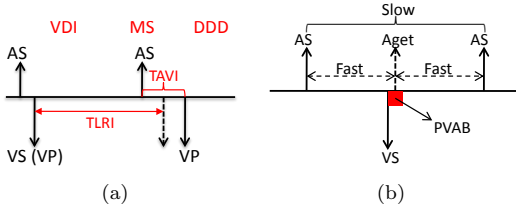


Fig. 16. (a) Safety Violation: VP is delayed due to the reset of timer during mode-switch, (b) Correctness Violation: The blocking period may block some atrial events, turning two *Fast* events to one *Slow* event [12]

mode the ventricular rate of the heart model is less than the Upper Rate Limit and will not trigger ventricular pacing. However, this property is still satisfied, indicating the mode-switch algorithm failed to eliminate the PMT scenario. The evidence trace returned by UPPAAL shows that a subset of atrial events fall into the blanking period after a ventricular event (see Fig. 16(b)). As a result, two fast events are reduced to one slow event and mode switch may never happen. This scenario does exist in all our refined heart models, we conclude that the trace is physiologically feasible. The mode-switch algorithm in our pacemaker model can not terminate all PMT behaviors as specified.

7.7 Trace Validation on Real Pacemaker and Pacemaker Refinement

In the previous subsections, we have found two potential safety violations in our pacemaker model. However, this does not mean the actual pacemaker has the same violation. Jiang et al. have implemented the VHM model onto a programmable integrated circuit (FPGA) platform and had it interact with a real pacemaker at run-time [24]. The closed-loop behavior can be checked in closed-loop with a real pacemaker. If the trace is not feasible in the closed-loop system, the pacemaker model needs to be refined to eliminate the execution. However, refining the pacemaker model requires more detailed representation of the pacemaker software, which was not available to us at that time.

8 Related Work

Chen et. al [25] extended our verification work [12]. They developed a hybrid heart model which is able to simulate action potential at tissue level. The model is a more

refined model than our Virtual Heart Model [20], with linear dynamics on each state of the heart tissue. They also developed a probability model to simulate natural pacemaker function. They then used the combined heart model for quantitative verification of the pacemaker. However, since the pacemaker only sense the timing of the heart tissue activation, their hybrid extension for action potential does not bring much benefit but increased model complexity dramatically. As a result, they have to use bounded model checking thus sacrificed accuracy.

Jee et. al present a safety assured development approach of real-time software using pacemaker as their case study in [26]. They formally model and verify a single chamber VVI pacemaker using UPPAAL and then implement it and check the preservation of properties transferred from model to implementation code.

Tuan et. al propose an RTS formal model for pacemaker and its environment and verified it against number of safety properties and timed constraints using the PAT model checker [27]. They have modeled the pacemaker for all 18 operating modes as described in Boston scientific, but their work lacks specification and analysis of complex behaviors of the pacemaker, such as mode-switch.

Wiggelinkhuizen uses mCRL2 and UPPAAL to formally model the pacemaker from the firmware design of Vitatron’s DA+ pacemaker [28]. Two main approaches have been used to investigate the feasibility of applying formal model checking to the design of device firmware. The main approach consists of verifying the firmware model in context of a formal heart model and a formal model of a hardware module which fails for high heart rates because of the state explosion. Another approach is to verify a part of firmware design which was feasible and was able to detect a known deadlock rather soon.

Macedo et. al have developed a concurrent and distributed real-time model for a cardiac pacemaker through a pragmatic incremental approach [29]. The models are expressed using the VDM and are validated primarily by scenario-based test, where test scenarios are defined to model interesting situations such as the absence of input pulses. The models cover 8 modes of pacemaker operation.

Gomes et. al present a formal specification of pacemaker system using the Z notation in [30]. They have also tried to validate that the formal specification satisfies the informal requirements of Boston Scientific by using a theorem prover, ProofPower-Z. They have partially checked the consistency of their specification through reasoning. No validation experiment regarding safety conditions were performed yet.

Mery et. al in [31], formally model all operational modes of a single electrode pacemaker system using event-B and prove them. They use an incremental proof-based approach to refine the basic abstract model of the system and add more functional and timing properties. They use

the ProB tool to validate their models in different situations such as absence of input pulses.

9 Conclusion and Future Work

In this paper, we first extend our TACAS'12 paper [12] by formalizing the heart model abstraction process. By establishing the timed simulation relation between abstraction levels the safety properties specified in ATCTL* are preserved during refinement. Based on the heart model formulation we proposed a Counter-Example-Guided Abstraction and Refinement (CEGAR) framework to balance model complexity and fidelity for pacemaker verification. We demonstrated how to identify two unsafe Pacemaker Mediated Tachycardia (PMT) executions using model checking and heart model refinement for more specific heart conditions. We also verified the safety and effectiveness of the corresponding Anti-PMT algorithms.

The work in this paper demonstrated the application of formal methods and model checking techniques for evaluating safety and efficacy of Cyber-Physical Systems like medical devices. This is a step toward certification of verified models and automated synthesis to verified code for medical device regulatory efforts. As future work, there are several directions worth exploration:

1. The CEGAR framework enables us to explore more complex pacemaker algorithms for more specific heart conditions.
2. In the CEGAR framework, model abstraction, model refinement and trace validity check are performed manually using physiological reasoning. The domain knowledge can be potentially formalized into rules and the abstraction and refinement process can be automated.
3. In this paper we only evaluated binary safety properties. We are interested in quantitative properties such as power consumption and physiological efficacy. Our current efforts are on extending this framework to incorporate tools and techniques for quantitative verification of closed-loop medical devices.
4. In this work we only investigate the interaction between the heart and the pacemaker so that only simple rate-based properties can be specified. Investigating the interaction between the heart and the oxygen demands from the rest of the body will give us better metric for the efficacy of the devices.

Acknowledgements. The authors would like to thank Ashutosh Trivedi, from the University of Pennsylvania and the Indian Institute of Technology, Mumbai, for fruitful discussions during the preparation of this paper.

References

1. List of Device Recalls, U.S. Food and Drug Admin., (last visited Jul. 19, 2010).
2. K. Sandler, L. Ohrstrom, L. Moy, and R. McVay. Killed by Code: Software Transparency in Implantable Medical Devices. *Software Freedom Law Center*, 2010.
3. AUTOSAR website: <http://www.autosar.org/>.
4. AVSI website: <http://www.avsi.aero>.
5. R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
6. K.G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, pages 134–152, 1997.
7. M.E. Josephson. *Clinical Cardiac Electrophysiology*. Lippincott Williams and Wilkins, 2008.
8. S. Barold, R. Stroobandt, and A. Sinnaeve. *Cardiac Pacemakers Step by Step*. Blackwell Futura, 2004.
9. E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.
10. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. 2000.
11. Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counter Example-Guided Abstraction Refinement for Symbolic Model Checking. *J. ACM*, 50(5):752–794, 2003.
12. Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. *Tools and Algorithms for the Construction and Analysis of Systems*, 7214:188–203, 2012.
13. The Compass - Technical Guide to Boston Scientific Cardiac Rhythm Management Products. 2007.
14. Zhihao Jiang, Miroslav Pajic, Rajeev Alur, and Rahul Mangharam. Pacemaker UPPAAL model download: <http://mlab.seas.upenn.edu>.
15. Miroslav Pajic, Zhihao Jiang, Insup Lee, Oleg Sokolsky, and Rahul Mangharam. From Verification to Implementation: A Model Translation Tool and a Pacemaker Case Study. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, RTAS '12*, pages 173–184, 2012.
16. R.N. Fogoros. *EP Testing*. Blackwell Science, 1999.
17. Satoshi Yamane. Timed Weak Simulation Verification and its Application to Stepwise Refinement of Real Time Software. *International Journal of Computer Science and Network Security*, 6, 2006.
18. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on UPPAAL. *Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science*, pages 200–236, 2004.
19. Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.
20. Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceedings of the IEEE*, 100(1):122–137, Jan. 2012.
21. Zhihao Jiang and Rahul Mangharam. Modeling Cardiac Pacemaker Malfunctions with the Virtual Heart Model. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 263–266, Sept 2011.

22. Z. Jiang, M. Pajic, and R. Mangharam. Model-based Closed-loop Testing of Implantable Pacemakers. In *IC-CPS'11: ACM/IEEE 2nd Intl. Conf. on Cyber-Physical Systems*, 2011.
23. PACEMAKER System Specification. Boston Scientific. 2007.
24. Zhihao Jiang, Sriram Radhakrishnan, Varun Sampath, Shilpa Sarode, Miroslav Pajic, and Rahul Mangharam. Heart-on-a-Chip: A Closed-loop Testing Platform for Implantable Pacemakers. In *Third Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (Cy-Phy)*, CPS Week, 2013.
25. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. In *Hybrid Systems: Computation and Control (HSCC 2013)*, 2013.
26. E. Jee, S. Wang, J. K. Kim, J. Lee, O. Sokolsky, and I. Lee. A Safety-Assured Development Approach for Real-Time Software. *The Proceedings of 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 133–142, 2010.
27. L. A. Tuan, M. C. Zheng, and Q. T. Tho. Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker. *Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 23–32, 2010.
28. J. E. Wiggelinkhuizen. Feasibility of Formal Model Checking in the Vitatron Environment. *Master thesis, Eindhoven University of Technology*, 2007.
29. Macedo H. D., Larsen P. G., and Fitzgerald J. Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System using VDM. *Formal Methods*, pages 28–30, 2008.
30. A. O. Gomes and M. V. Oliveira. Formal Specification of a Cardiac Pacing System. In *Proceedings of the 2nd World Congress on Formal Methods (FM '09)*, pages 692–707, 2009.
31. D. Mery and N. K. Singh. Pacemaker's Functional Behaviors in Event-B. *Research report, INRIA*, 2009.

A Appendix

Proof for $N_2^1 \parallel P_2 \parallel N_2^2 \preceq_t N_3^1 \parallel P_3 \parallel N_3^2$

In this subsection we manually prove the timed simulation relation $N_2^1 \parallel P_2 \parallel N_2^2 \preceq_t N_3^1 \parallel P_3 \parallel N_3^2$. The other relations can be proved accordingly.

A.1 Timed Simulation Relation *sim*

For two timed automata T^1 and T^2 , we write timed simulation relation *sim* is defined on $\Omega_1 \times \Omega_2$ where $(s, v) \in \Omega_1$ and v is the valuation of all clocks $t_1 \in X_1$. $((s, v), (s', v'))$ is in *sim* if and only if $v' = v(\lambda + \delta)$, for $\delta \in D$, such that $\text{sim}(s, v) = (s', v(\lambda + D))$

Let $T^1 = N_2^1 \parallel P_2 \parallel N_2^2$ and $T^2 = N_3^1 \parallel P_3 \parallel N_3^2$. $X_1 = \{tn_2^1, tp_2, tn_2^2\}$ and $X_2 = \{tn_3^1, tp_3, tn_3^2\}$, the state mapping for the timed simulation relation is shown below:

$$\text{sim}(RE \parallel ID \parallel RE, v) = (RE \parallel ID \parallel RE, v(tn_3^1 + D_1, tn_3^2 + D_2)) \text{ where } D_1 = [N_2^1.Terp_min, N_2^1.Terp_max], \\ D_2 = [N_2^2.Terp_min, N_2^2.Terp_max]$$

A special condition for this mapping is the initial state: $\text{sim}(RE \parallel ID \parallel RE, v(tn_3^1 := 0, tp_3 := 0, tn_3^2 := 0)) = (RE \parallel ID \parallel RE, v(tn_3^1 + D_1, tn_3^2 + D_2))$

$$\text{sim}(ER \parallel AN \parallel RE, v) = (RE \parallel AN \parallel RE, v)$$

$$\text{sim}(RE \parallel RT \parallel ER, v) = (RE \parallel RT \parallel RE, v)$$

$$\text{sim}(ER \parallel ID \parallel ER, v) = (RE \parallel ID \parallel RE, v)$$

$$\text{sim}(ER \parallel ID \parallel RE, v) = (RE \parallel ID \parallel RE, v(tn_3^2 + D)) \text{ where } D = [N_2^2.Terp_min, N_2^2.Terp_max]$$

$$\text{sim}(RE \parallel ID \parallel ER, v) = (RE \parallel ID \parallel RE, v(tn_3^1 + D)) \text{ where } D = [N_2^1.Terp_min, N_2^1.Terp_max]$$

The location mapping is shown in Fig. 5.

A.2 Timed Transitions

Here we want to ensure every timed transition for T^1 has a corresponding timed transition in T^2 . We denote $S = (s, v)$ and $S_\delta = (s, v + \delta)$. For each location in T^1 we have $S \xrightarrow{\delta} S_\delta$ for $\forall \delta \in \mathbb{R}$ under condition $v + \delta \models \text{inv}(s)$. In T^2 we use $S' = (s', v')$ and $S'_\delta = (s', v' + \delta)$. For $(S, S') \in \text{sim}$ we show that there exists S'_δ such that $s' \xrightarrow{\delta} s'_\delta$ and $(s_\delta, s'_\delta) \in \text{sim}$. For every location we list all the corresponding components and provide proof.

For location $RE \parallel ID \parallel RE$ in T^1 we have:

$$S = (RE \parallel ID \parallel RE, v)$$

$$S_\delta = (RE \parallel ID \parallel RE, v + \delta)$$

$$\text{inv}(s) = tn_2^1 + \delta \leq N_2^1.Trest_max \ \&\&$$

$$\begin{aligned}
&tn_2^2 + \delta \leq N_2^2.Trest_max \\
&D_1 = [N_2^1.Terp_min, N_2^1.Terp_max], \\
&D_2 = [N_2^2.Terp_min, N_2^2.Terp_max]. \\
&S' = (RE\|ID\|RE, v(tn_3^1 + D_1, tn_3^2 + D_2)) \\
&S'_\delta = (RE\|ID\|RE, v(tn_3^1 + D_1, tn_3^2 + D_2) + \delta) \\
&inv(s') = tn_3^1 + D_1 + \delta \leq N_3^1.Trest_max \text{ and } tn_3^2 + D_2 + \\
&\delta \leq N_3^2.Trest_max
\end{aligned}$$

Since

$$N_3.Trest_max = N_2.Trest_max + N_2.Terp_max$$

we have $inv(s) \equiv inv(s')$ so the correspondence holds.

For location $ER\|AN\|RE$ in T^1 we have:

$$\begin{aligned}
&S = (ER\|AN\|RE, v) \\
&S_\delta = (ER\|AN\|RE, v + \delta) \\
&inv(s) = tp_2 + \delta \leq P_2.Tcond_max \\
&S' = (RE\|AN\|RE, v) \\
&S'_\delta = (RE\|AN\|RE, v + \delta) \\
&inv(s') = tp_3 + \delta \leq P_3.Tcond_max
\end{aligned}$$

Since $P_2.Tcond_max == P_3.Tcond_max$, we have $inv(s) \equiv inv(s')$ so the correspondence holds.

For location $RE\|RT\|ER$ in T^1 we have:

$$\begin{aligned}
&S = (RE\|RT\|ER, v) \\
&S_\delta = (RE\|RT\|ER, v + \delta) \\
&inv(s) = tp_2 + \delta \leq P_2.Tcond_max \\
&S' = (RE\|RT\|RE, v) \\
&S'_\delta = (RE\|RT\|RE, v + \delta) \\
&inv(s') = tp_3 + \delta \leq P_3.Tcond_max
\end{aligned}$$

Since $P_2.Tcond_max == P_3.Tcond_max$, we have $inv(s) \equiv inv(s')$ so the correspondence holds.

For location $ER\|ID\|ER$ in T^1 we have:

$$\begin{aligned}
&S = (ER\|ID\|ER, v) \\
&S_\delta = (ER\|ID\|ER, v + \delta) \\
&inv(s) = tn_2^1 + \delta \leq N_2^1.Terp_max \&\& \\
&tn_2^2 + \delta \leq N_2^2.Terp_max \\
&S' = (RE\|ID\|RE, v) \\
&S'_\delta = (RE\|ID\|RE, v + \delta) \\
&inv(s') = tn_3^1 + \delta \leq N_3^1.Trest_max \&\& \\
&tn_3^2 + \delta \leq N_3^2.Trest_max
\end{aligned}$$

Since

$$N_3.Trest_max = N_2.Trest_max + N_2.Terp_max$$

We have $inv(s) \subseteq inv(s')$ so the correspondence holds.

For location $ER\|ID\|RE$ in T^1 we have:

$$\begin{aligned}
&S = (ER\|ID\|RE, v) \\
&S_\delta = (ER\|ID\|RE, v + \delta) \\
&inv(s) = tn_2^1 + \delta \leq N_2^1.Terp_max \&\& \\
&tn_2^2 + \delta \leq N_2^2.Trest_max \\
&D = [N_2^2.Terp_min, N_2^2.Terp_max] \\
&S' = (RE\|ID\|RE, v(tn_3^2 + D)) \\
&S'_\delta = (RE\|ID\|RE, v(tn_3^2 + D) + \delta) \\
&inv(s') = tn_3^1 + \delta \leq N_3^1.Trest_max \&\& \\
&tn_3^2 + D + \delta \leq N_3^2.Trest_max
\end{aligned}$$

Since

$$N_3.Trest_max = N_2.Trest_max + N_2.Terp_max$$

We have $inv(s) \subseteq inv(s')$ so the correspondence holds.

For location $RE\|ID\|ER$ in T^1 we have:

$$\begin{aligned}
&S = (RE\|ID\|ER, v) \\
&S_\delta = (RE\|ID\|ER, v + \delta) \\
&inv(s) = tn_2^1 + \delta \leq N_2^1.Trest_max \&\& \\
&tn_2^2 + \delta \leq N_2^2.Terp_max \\
&D = [N_2^1.Terp_min, N_2^1.Terp_max] \\
&S' = (RE\|ID\|RE, v(tn_3^1 + D)) \\
&S'_\delta = (RE\|ID\|RE, v(tn_3^1 + D) + \delta) \\
&inv(s') = tn_3^1 + D + \delta \leq N_3^1.Trest_max \&\& \\
&tn_3^2 + \delta \leq N_3^2.Trest_max
\end{aligned}$$

Since

$$N_3.Trest_max = N_2.Trest_max + N_2.Terp_max$$

We have $inv(s) \subseteq inv(s')$ so the correspondence holds.

A.3 Discrete Transitions

During a discrete transition, a state $S = (s, v)$ proceed to $S_\lambda = (s', v(\lambda := 0))$. Here we prove that for every discrete transition $S \xrightarrow{\sigma} S_\lambda$ in T^1 , there exists $S_1 = (s_1, v)$ such that $(S, S_1) \in \text{sim}$, $S_1 \xrightarrow{\sigma} S_{\lambda,1}$ for $S_{\lambda,1} = (s'_1, v(\lambda' := 0))$ and $(S_\lambda, S_{\lambda,1}) \in \text{sim}$.

Self-activation for N_2^1 triggers antegrade conduction, we have $N_3.Trest = N_2.Terp + N_2.Trest$

$$\begin{aligned}
&(RE\|ID\|RE, v) \xrightarrow[\text{Act_path.1!}]{tn_2^1 > N_2^1.Trest_min \| \text{Act_node.1?}} \\
&(ER\|AN\|RE, v(tn_2^1 := 0, tp_2 := 0)) \\
&\Downarrow \\
&(RE\|ID\|RE, v) \xrightarrow[\text{Act_path.1!}]{tn_3^1 > N_3^1.Trest_min \| \text{Act_node.1?}} \\
&(ER\|AN\|RE, v(tn_3^1 := 0, tp_3 := 0))
\end{aligned}$$

Self-activation for N_2^2 triggers retrograde conduction, we have $N_3.Trest = N_2.Terp + N_2.Trest$.

$$\begin{aligned}
&(RE\|ID\|RE, v) \xrightarrow[\text{Act_path.2!}]{tn_2^2 > N_2^2.Trest_min \| \text{Act_node.2?}} \\
&(RE\|RT\|ER, v(tp_2 := 0, tn_2^2 := 0)) \\
&\Downarrow \\
&(RE\|ID\|RE, v) \xrightarrow[\text{Act_path.2!}]{tn_3^2 > N_3^2.Trest_min \| \text{Act_node.2?}} \\
&(RE\|RT\|RE, v(tp_3 := 0, tn_3^2 := 0))
\end{aligned}$$

N_2^2 activated after antegrade conduction

$$\begin{aligned}
&(ER\|AN\|RE, v) \xrightarrow[\text{Act_node.2!}]{tp_2 > Tcond_min} \\
&(ER\|ID\|ER, v(tn_2^2 := 0)) \\
&\Downarrow \\
&(RE\|AN\|RE, v) \xrightarrow[\text{Act_node.2!}]{tp_3 > Tcond_min} \\
&(RE\|ID\|RE, v(tn_3^2 := 0))
\end{aligned}$$

N_2^1 activated after retrograde conduction

$$\begin{aligned} & (RE\|RT\|ER, v) \xrightarrow[\text{Act_node.1!}]{tp_2 > Tcond_min} \\ & (ER\|ID\|ER, v (tn_2^1 := 0)) \\ & \downarrow \\ & (RE\|RT\|RE, v) \xrightarrow[\text{Act_node.1!}]{tp_3 > Tcond_min} \\ & (RE\|ID\|RE, v (tn_3^1 := 0)) \end{aligned}$$

ERP of N_2^1 finishes first

$$\begin{aligned} & (ER\|ID\|ER, v) \xrightarrow[\text{Act_node.1!}]{tn_2^1 > N_2^1.Terp_min} \\ & (RE\|ID\|ER, v (tn_2^1 := 0)) \\ & \downarrow \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_node.1!}]{tn_3^1 > N_3^1.Terp_min} \\ & (RE\|ID\|RE, v) \end{aligned}$$

ERP of N_2^2 finishes first

$$\begin{aligned} & (ER\|ID\|ER, v) \xrightarrow[\text{Act_node.1!}]{tn_2^2 > N_2^2.Terp_min} \\ & (ER\|ID\|RE, v (tn_2^2 := 0)) \\ & \downarrow \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_node.1!}]{tn_3^2 > N_3^2.Terp_min} \\ & (RE\|ID\|RE, v) \end{aligned}$$

ERP of N_2^2 finishes after N_2^1

$$\begin{aligned} & (RE\|ID\|ER, v) \xrightarrow[\text{Act_node.1!}]{tn_2^2 > N_2^2.Terp_min} \\ & (RE\|ID\|RE, v (tn_2^2 := 0)) \\ & \downarrow \\ & D = [N_2^1.Terp_min, N_2^1.Terp_max] \\ & (RE\|ID\|RE, v (tn_3^1 + D)) \xrightarrow[\text{Act_node.1!}]{tn_3^2 > N_3^2.Terp_min} \\ & (RE\|ID\|RE, v (tn_3^1 + D)) \end{aligned}$$

ERP of N_2^1 finishes after N_2^2

$$\begin{aligned} & (ER\|ID\|RE, v) \xrightarrow[\text{Act_node.1!}]{tn_2^1 > N_2^1.Terp_min} \\ & (RE\|ID\|RE, v (tn_2^1 := 0)) \\ & \downarrow \\ & D = [N_3^1.Terp_min, N_3^1.Terp_max] \\ & (RE\|ID\|RE, v (tn_3^2 + D)) \xrightarrow[\text{Act_node.1!}]{tn_3^1 > N_3^1.Terp_min} \\ & (RE\|ID\|RE, v (tn_3^2 + D)) \end{aligned}$$

N_2^2 is activated when N_2^1 is in ERP

$$\begin{aligned} & (ER\|ID\|RE, v) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} \\ & (ER\|ID\|ER, v (tn_2^2 := 0)) \\ & (\downarrow) \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} \\ & (RE\|ID\|RE, v (tn_3^2 := 0)) \end{aligned}$$

N_2^2 is activated when N_2^1 is in ERP

$$\begin{aligned} & (RE\|ID\|ER, v) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} \\ & (ER\|ID\|ER, v (tn_2^1 := 0)) \\ & (\downarrow) \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} \end{aligned}$$

$$(RE\|ID\|RE, v (tn_3^1 := 0))$$

Blocking during ERP is simulated by a non-deterministic transition in the path

$$\begin{aligned} & (ER\|ID\|ER, v) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} (ER\|ID\|ER, v) \\ & \downarrow \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} \xrightarrow[\text{Act_path.1!}]{\text{Act_path.1?}} \\ & (RE\|ID\|RE, v) \\ & \\ & (ER\|ID\|ER, v) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} (ER\|ID\|ER, v) \\ & \downarrow \\ & (RE\|ID\|RE, v) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} \xrightarrow[\text{Act_path.2!}]{\text{Act_path.2?}} \\ & (RE\|ID\|RE, v) \\ & \\ & (ER\|ID\|RE, v) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} (ER\|ID\|RE, v) \\ & \downarrow \\ & D = [N_2^2.Terp_min, N_2^2.Terp_max] \\ & (RE\|ID\|RE, v (tn_3^2 + D)) \xrightarrow[\text{Act_path.1!}]{\text{Act_node.1?}} \xrightarrow[\text{Act_path.1!}]{\text{Act_path.1?}} \\ & (RE\|ID\|RE, v (tn_3^2 + D)) \\ & \\ & (RE\|ID\|ER, v) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} (RE\|ID\|ER, v) \\ & \downarrow \\ & D = [N_2^1.Terp_min, N_2^1.Terp_max] \\ & (RE\|ID\|RE, v (tn_3^1 + D)) \xrightarrow[\text{Act_path.2!}]{\text{Act_node.2?}} \xrightarrow[\text{Act_path.2!}]{\text{Act_path.2?}} \\ & (RE\|ID\|RE, v (tn_3^1 + D)) \end{aligned}$$