Pacemaker UPPAAL Abstraction

Sindhu Honnali, Hitali Sheth, Kerem Kazan, David Freifelder

November 17, 2015

1 Modeling

The following automata have been designed to model the pacemaker for this milestone.

1.1 Keyboard

Simulates key presses by generating a random value from 1 to 13 at arbitrary intervals. This value is then received by mode switch automata which select the modes for heart and the pacemaker. The following are the key press values and their corresponding actions.

Pacemaker Controls	
1	Normal Mode
2	Sleep Mode
3	Exercize Mode
4	Manual Mode
5	Generate `APace`
6	Generate `VPace`
7	Modify Observation Interval
Heart Controls	
8	Random Mode
9	Manual Mode
10	Test Mode
11	Generate `ASense`
12	Generate `VSense`
13	Modify Observation Interval

1.2 Mode Switch

This template dictates the modes of pacemaker and heart based on the key pressed values. To change from one mode to another for heart and pacemaker we test the corresponding guard conditions. These guard conditions check both that the correct key was simulated, and that changing this mode does not create inconsistent state (as modifying certain variables at the wrong time may violate invariants.)

1.3 Heart

This automata consists of two parts.

1. The first part models random mode, manual mode and receives signals from the pacemaker. In the manual mode, when the appropriate key value is received, `ASense` and `VSense` signals are generated and clocks are reset. In the random model, `ASense` and `VSense` signals are generated randomly. The heart also receives the signals generated by pacemaker when there is a failure in generating signals by the heart or if pacemaker is in manual mode and resets the clock. This way both heart and pacemaker are synchronized.

2. The second portion is a test mode. In this mode, the heart non-deterministically chooses a test from its entry point and executes it, raising a flag variable if it fails.

   - Normal operation test : In this test the heart functions normally; i.e. the heart generates `ASense` and `VSense` at the correct rate and hence signals are not required to be generated by the pacemaker. If any of the signals are generated by the pacemaker, the test fails.

- `VSense` exceeds : In this test, if `VSense` is not received after VRP and before LRI , then `VPace` gets generated by the pacemaker at the LRI. If `VSense` is received within VRP, then `VSense` is ignored by the heart and by the pacemaker and heart expects the next signal as `APace` at LRI - AVI.
- `ASense` exceeds : In this test, if `ASense` is not received at LRI - AVI, then `APace` is expected. If `ASense` is received after LRI - AVI, then it is ignored and the heart expects the next signal as `VPace` at LRI.
- `ASense` too soon : In this test, if the `ASense` is received within PVARP, then it gets ignored and the heart expects `APace` at LRI - AVI.
- `VSense` too soon: In this test, if `VSense` is asserted before URI or before AVI-min or before VRP, `VSense` needs to be ignored and the heart expects `VPace` at LRI.

Each of the above tests checks the pacemaker's response to the various ways it is expected to react: it does not disrupt correct behavior, triggers if atrial or ventricle signals miss their interval, and ignores signals that occur within a period of uncertainty.

1.4 Pacemaker

The pacemaker model has two main locations: `ANext` and `VNext`. There exist two clocks in the system: one representing time since the last acknowledged atrial event (cA) and the other for the last ventricle event (cV). During the normal, exercise and sleep modes of the pacemaker, when the pacemaker is in `ANext` state, it waits for a properly timed `ASense` signal, and if the signal is not received the pacemaker generates `APace` and moves to the `VNext` state. In the `VNext` state, the pacemaker waits for a properly timed `VSense` signal, and if the signal is not received the pacemaker generates `VPace` and moves to the `ANext` state.

When the pacemaker is in the manual mode, which may be reached from either state, the keys for atrial and ventricle signals generate the corresponding `APace` and `VPace` signals, and the clocks are reset as appropriate. Note that being in manual mode may violate the invariants of the other modes, so changing back to automatic is restricted by those same invariants.

1.5 LED

The four LEDs blink when the corresponding `APace`, `ASense`, `VPace` and `VSense` signals are generated.

1.6 Display

A pair of these templates are initialized, corresponding to the heart and pacemaker controls. When `VSense` or `VPace` signal is received within the observation interval of the appropriate target, the counter is incremented to keep count of the heart beat, and later the heart rate is displayed.

The representation here is somewhat restricted by state-space concerns; since the number of heartbeats can become arbitrarily great due to the random heart or user input, the state-space required for verification becomes too great. Thus, the counter is instead modeled with a channel to increment and a channel to display and reset. In an implementation of this section, these channels would be replaced by the appropriate methods.[1]

1.7 Alarm

If the `VPace` or `VSense` is received at time less than URI, `alarmFast` is set. Similarly, if the `VPace` or `VSense` is received at time greater than LRI, `alarmSlow` is set. These boolean variables correspond to the appropriate alarm sounds that would be triggered in an actual implementation.

[1]In practice, the verification is further restricted by general complexity. Adding a pair of clocks for the purpose of display causes exponential run-time increases in simulation; for this reason we provide the replacement template DisplayFast, which provides the same abstraction at a fraction of the simulation rate. The difference between these two templates is purely internal; using one or the other should not change the results of the verification queries.

2 Verification

The following queries are presented to verify the various safety and liveness properties of the system.

1. `A[] not deadlock` : Checks for any deadlock in the system.
2. `A[] not fail` : In the heart's test mode, `fail` is asserted when an unwanted `VPace` or `APace` is detected, or is not detected but should have been. This query verification asserts that all the tests pass, since any may be entered non-deterministically, and also shows that within the bounds of any particular pacemaker mode, the pacemaker performs as it should, satisfying the conditions below:
   - If the heart acts as normal, no pacemaker activity is generated
   - `ASense` may not be acknowledged within PVARP
   - `VSense` may not be acknowledged within VRP
   - If `ASense` is not received before LRI - AVI, then `APace` is generated
   - If `VSense` is not received before LRI, then `VPace` is generated by the pacemaker
3. `A.VReceived && cV > LRI[paceMode] --> alarmSlow` : `alarmSlow` is set when a ventricular signal is received after the expected interval between ventricular signals, LRI, is surpassed.
4. `A.VReceived && cV < URI[paceMode] --> alarmFast` : `alarmFast` is set when the expected minimum interval between ventricular signals, URI, is not yet over and still a ventricular signal is received.
5. `A[] (L.BlinkAP imply cA == 0)`
   `A[] (L.BlinkVP imply cV == 0)`
   `A[] (L.BlinkAS imply ASC.c == 0)`
   `A[] (L.BlinkVS imply VSC.c == 0)` : cA and cV are clocks which are set when atrial or ventricular signals take place. The first two queries check that if the atrial or ventricular signals are generated by the pacemaker, then the corresponding LED blinks. The second two queries do the same for the signals from the heart, making use of a helper template to keep track of the signal.
6. `A[] P.APSend imply (cV == LRI[paceMode] - AVI-min) and (cV >= PVARP)`

   This test verifies the conditions required for the pacemaker to send the `APace` signal. The time since the last ventricle event must be greater than PVARP, and the pacemaker should not intervene if the clock has not exceeded the amount of time within which the heart should have triggered a sense event. The latter is shown by the upper bound between cycles (LRI of the current pace mode), minus the time needed before the next ventricle event, AVI-min.
7. `A[] P.VPSend imply (cALast == AVI-max and cALast > AVI-min) or (cVLast == LRI[paceMode] and cVLast > URI[paceMode] and cVLast > VRP)`

   This test verifies the conditions required for the pacemaker to send the `VPace` signal. There are two reasons this may occur; either the time since the last atrial event is too great, or the time since the last ventricle event. If the former is true, the time must be at the upper bound between atrial and ventricle events (AVI-max) and exceed the minimum. If the former is true, the time between ventricle events must be at its maximum (LRI of the current pace mode), and must exceed both the lower bound of that range and VRP.

As we have asserted that the pacemaker sends signals when required, and that it only sends signals when required, we have verified that the pacemaker sends signals *if and only if the specifications say that it should.* That these queries can be successfully verified shows both the correctness and the liveness properties of the system.