Final Project Milestone 1

Group Members

Henry Garant(henrygar - 22505341), Max Brown(maxbrown - 56554496)

Muhammad Asghar(masghar - 71506566), Kwame Owusu(86952693)

UPPAAL model and Verification Queries

Intro to the Model:

We modeled a Heart and PaceMaker in VVI mode using UPPAAL software for timed automata. Specifications of the model were derived from Boston Scientific's system specification for a pacemaker. The model consists of two timed automata: one is a Random-Heart, and the other is a Ventricle controller. The Random-Heart mimics any heart behaviors (e.g., beating just randomly, in a given range.). A Random-Heart in VVI mode receives only pacing signals to the ventricle and sends signals to the controller when a (spontaneous) heartbeat occurs. The Ventricle controller observes the heart by sensing the heart signal and responds by sending a pacing signal to the heart if the heart fails to beat by itself for a certain amount of time.

Three key definitions to understand the model are:

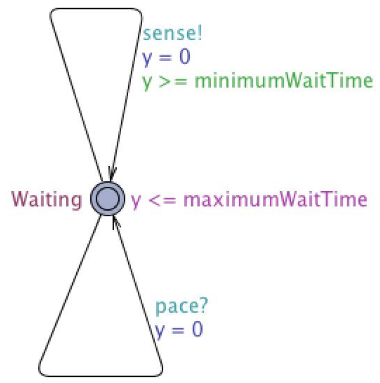
The Lower Rate Limit (LRL) decides the longest length between a V event and the next ventricular pace. This can decide the lower bound of the heart rate,

The Upper Rate Limit (URL) decides the shortest length between a V event and the next ventricular pace. This can decide the upper bound of the heart rate.

The Ventricular Refractory Period (VRP) shall be a programmed time interval following a ventricular event during which time ventricular senses shall not inhibit nor trigger pacing.

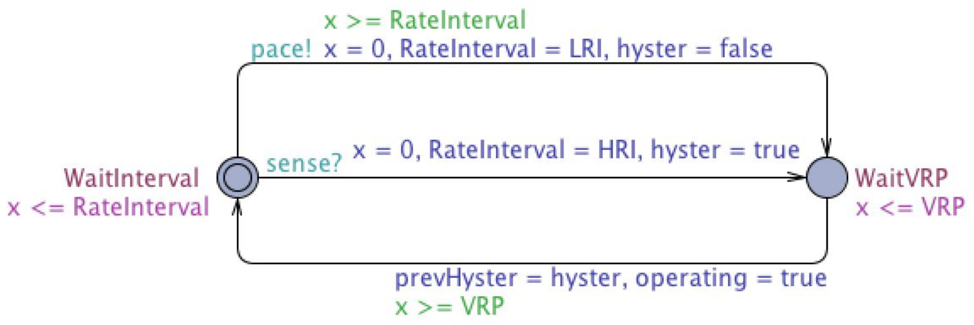
Heart Model:

The entire goal of the heart model is to provide testing for the Ventricle model by mimicking a heart that requires a pacemaker. The single state of the heart, ***Waiting***, is where the model waits for a pacing event from the Pacemaker or sends a sensing event to the Pacemaker. The sensing event sent from the Heart model is between the time constraint bounds of the ***minimumWaitTime***(LRL) and the ***maximumWaitTime***(URL). After an event is sent or detected the Heart reenters the *Waiting* state after resetting it's internal clock, ***y***, for timing.



VentricleController Model:

The entire goal of the VentricleController model is to replicate a Pacemaker in VVI mode. The Model has two states, ***WaitInterval*** and ***WaitVRP. WaitInterval*** state waits for a sense or pace event to be fired. Obviously, if a heart sense event is not detected within the time of the ***RateInterval***, then the VentricleController sends a pacing signal to the heart and the new ***RateInterval*** is ***LRI***. Observe that this transition means the Heart is not in hysteresis pacing. When the heart is in hysteresis pacing, then the High Rate Interval(***HRI***) must be applied. Whenever an event is detected in the VentricleController model, the internal clock, ***x***, is reset for timing consistency. The next state is ***WaitVRP***, where the Pacemaker shall not interrupt or detect signals. In ***WaitVRP***, the VentricleController is waiting for time equivalent to ***VRP*** to begin operating again. Finally, observe that it is critical for the VentricleController to know if it was previously in hysteresis; therefore, the model 'remembers' via the use of a boolean variable ***prevHyster***.


x >= RateInterval
pace! x = 0, RateInterval = LRI, hyster = false
sense? x = 0, RateInterval = HRI, hyster = true
WaitInterval
x <= RateInterval
WaitVRP
x <= VRP
prevHyster = hyster, operating = true
x >= VRP

Verification Queries:

1. **A[] (not deadlock)**

This query ensures there is no deadlock in the model. This is especially important for the Pacemaker because deadlock could result in not properly regulating the heart as it would need to be for a patient ALWAYS.

2. **A[] (!PaceMaker.prevHyster imply PaceMaker.x <= PaceMaker.LRI)**
   This query ensures that when the hysteresis is disabled i.e the heart is working properly, the pacemaker value of RateInterval = LRI and in the case when the heart does not function properly, it triggers a pacemaker response. This is done before the expiration of the RateInterval period. In this case LRI = RateInterval.

3. **A[] (PaceMaker.prevHyster imply PaceMaker.x <= PaceMaker.HRI)**
   This query ensures that when the heart is in hysteresis, the pacemaker makes use of the value of HRI that we pre-define. This, in turn, causes the pacemaker to generate the pace before the RateInterval period expires. Thus helping the heart function normally.

4. **A[] (!PaceMaker.operating imply PaceMaker.x <= PaceMaker.VRP)**

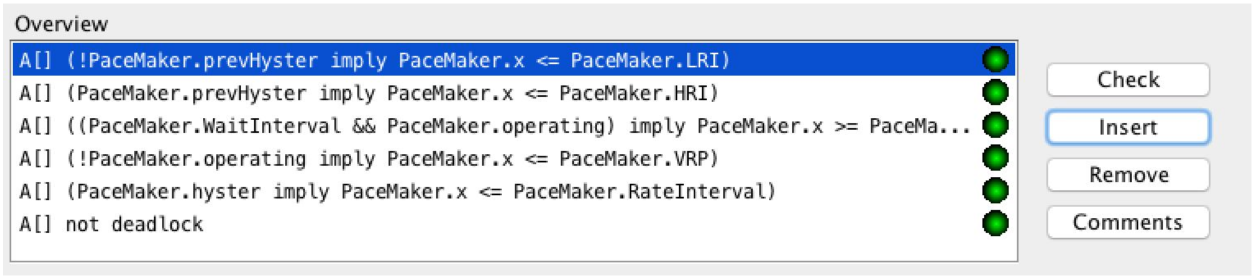
This query ensures that if the PaceMaker is not operating then the PaceMaker is in VRP period. This property is necessary to show that the PaceMaker is always active and ready for signal(operating) unless it is in VRP period, in which case the PaceMaker shall not interfere with the heart.

5. **A[] ((PaceMaker.WaitInterval && PaceMaker.operating) imply PaceMaker.x >= PaceMaker.VRP)**

After an event, The VentricleController should not be sensing events i.e. in VRP period. This query ensures that the VentricleController satisfies the VRP period time requirement. This is done by testing if in the WaitInterval state that the internal clock has already waited for VRP period amount of time.

6. **A[] (PaceMaker.hyster imply PaceMaker.x <= PaceMaker.RateInterval)**

This query ensures that the pacemaker does not fire when a sensing event occurs within the rate interval.



```
A[] (!PaceMaker.prevHyster imply PaceMaker.x <= PaceMaker.LRI)                          ●
A[] (PaceMaker.prevHyster imply PaceMaker.x <= PaceMaker.HRI)                           ●
A[] ((PaceMaker.WaitInterval && PaceMaker.operating) imply PaceMaker.x >= PaceMa...     ●
A[] (!PaceMaker.operating imply PaceMaker.x <= PaceMaker.VRP)                           ●
A[] (PaceMaker.hyster imply PaceMaker.x <= PaceMaker.RateInterval)                      ●
A[] not deadlock                                                                        ●
```