INF575 (Lab 3 & 4)

Quentin LAO

1 Uppaal development

In Lab3&4, we designed the software component for a DDD pacemaker on Uppaal.

1.1 Structure of the system

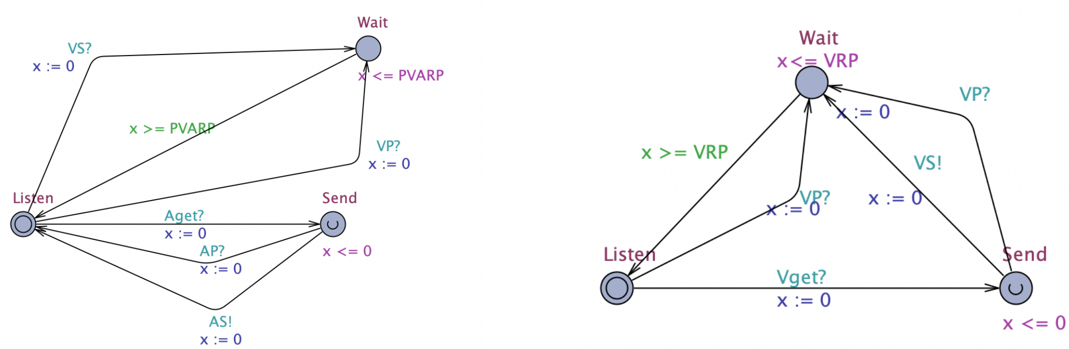
The system is composed by 6 principles automatons :

- **HeartA & HeartV** : they model the random heart as two non-deterministic automatons, one for atrial part and the second one for the ventricular part. HeartA sends *Aget* when atria want to beat naturally and HeartV sends *Vget* when ventricles want to beat naturally.
- **FilterA & FilterV** : they filter the natural signals received by the heart (*Aget* and *Vget*) so that it doesn't beat too fast.
- **PaceA & PaceV** : they models the DDD pacemaker behaviours. PaceA sends *AP* when the pacemaker makes atria beat and PaceV sends *VP* when the pacemaker makes ventricles beat.

(a) HeartA

(b) HeartV

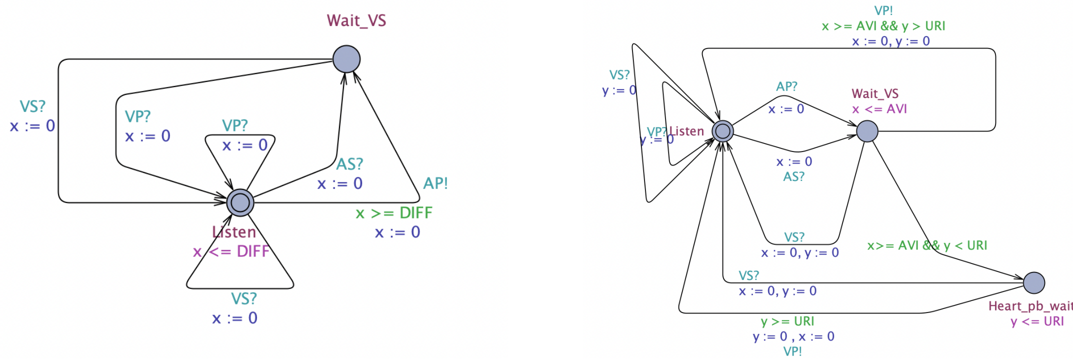
Figure 1: HeartA & HeartV in Uppaal



(a) FilterA

(b) FilterV

Figure 2: FilterA & FilterV in Uppaal



(a) PaceA (b) PaceV

Figure 3: PaceA & PaceV in Uppaal

1.2 Choice of constants

The global constants are given by the paper and can be directly defined in the declaration tab in Uppaal :

```
//Constants for PaceV
const int AVI = 150;
const int URI = 400;

//Constants for PaceA
const int LRI = 1000;
const int DIFF = LRI - AVI;

//Constants for FilterV
const int VRP = 100;

//Constants for FilterA
const int PVARP = 100;
```

For **HeartA** and **HeartV**, 4 local constants have to be defined by us : L_A, U_A, L_V and U_V.

- L_A & L_V : these two variables represent the minimum duration between two "*get* events" (*Aget* and *Vget*) of the random heart. We can set L_A and L_V smaller than 100 but it is useless as the constants PVARP = VRP = 100 (PVARP and VRP allow to ignore the "*get* events" that are temporarily too closed to another "*get* event").
  So, I set $L_A = L_V = 100$.

- U_A & U_V : these variables represent the maximum duration between two "*get* events" of the random heart. I set them to 2000 to have enough freedom when I'll be building specific traces. If U_A and U_V are too close to L_A and L_V, the heart is becoming deterministic and the pacemaker doesn't need to be as complex (heart's behaviours are predictable).

1.3 Trace example

In figure 4, we have an example of interactions between the random heart and the pacemaker.

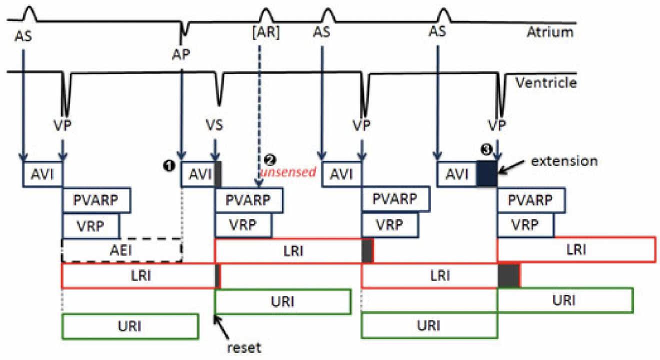


Figure 4: Example of Heart-Pacemaker interactions

I replicate these interactions in Uppaal. Here is the trace i've got in figure 5.
Each red arrow represent a sync signal (broadcast channel). If we focus only on AS/AP/VS/VP arrows, we can verify this trace corresponds to the example of interactions shown in figure 4. AS/VS are natural pulse whereas AP/VP are pulses coming from the pacemaker. Each AS/VS are preceded by a "*get* events" (*Aget* or *Vget*). Some additional comments :

- (1) : This *Aget* arrow is not linked to any other automaton. Indeed, this *Aget* corresponds to [AR] in figure 4. This *Aget* is unsensed/ignored thanks to the automaton **FilterA** because it is too closed to a previous ventricular event.
- (2) : **PaceV** is in state "Heart_pb_wait". This state corresponds to the extension arrow in figure 4. The duration AVI has been reached but not URI. **PaceV** is awaiting for URI to be reach before sending the signal *VP*.

2 Model-checking

2.1 Deadlock-freeness

We can check the property in Uppaal:

```
A[] not deadlock
```

The property is satisfied.

2.2 Time will always pass

Observer creation To check that time will always pass, I created an additional automaton **Observer** (figure 6) where t is a constant :

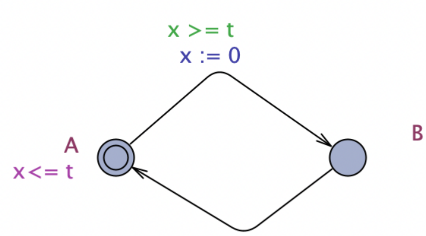
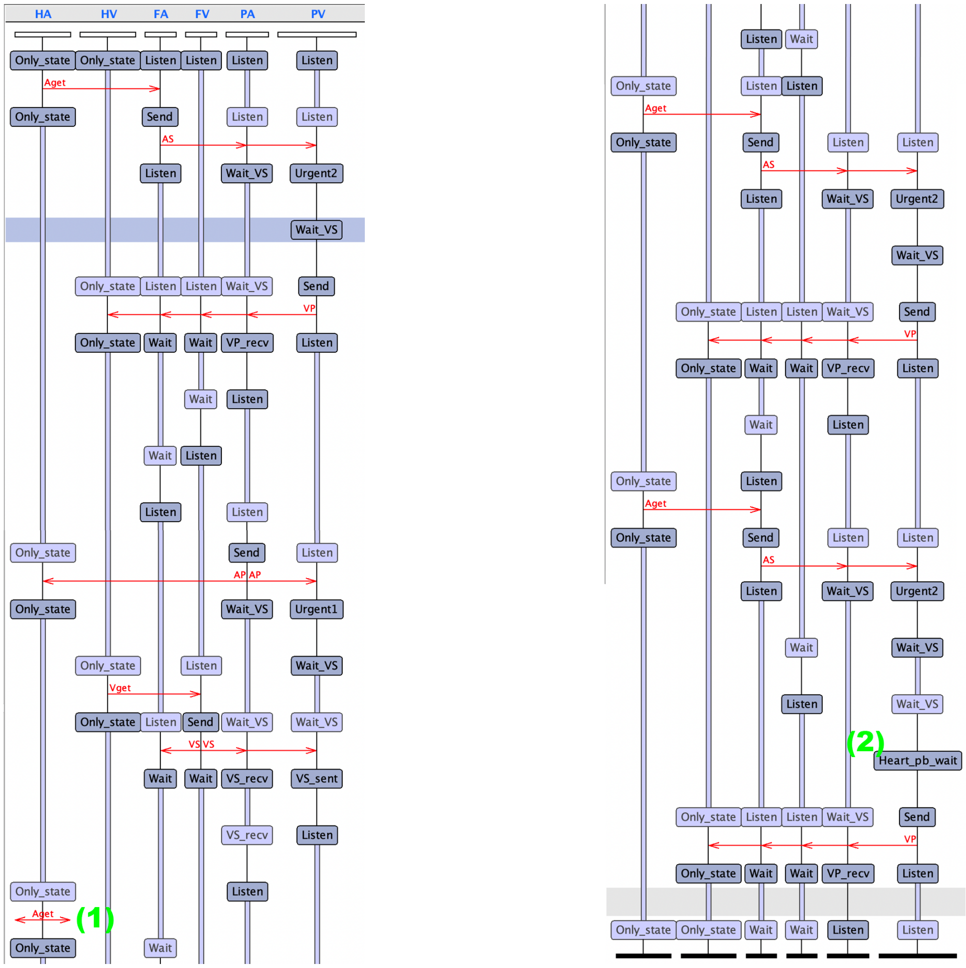


Figure 6: Observer

Then, we can check the following liveness property in Uppaal :

```
Observer.A --> Observer.B.          (1)
```



(a) First part of the trace

(b) Second part of the trace

Figure 5: Trace example

Choice of t Note that the choice of the constant t is quite important. As t is small, the **Observer** will be more often in state A which would require more checkings from Uppaal and thus more time verification. A good choice for t would be a t that fits well with the orders of magnitude of the constants in the system. I choose $t = 100$.

The property (1) is satisfied.

Limit of the verification The formula (1) is actually implementing :

```
A[]  (Observer.A => A<>Observer.B)
```

Notice that the validity of this property is only a sufficient condition for a timelock-free system and not a necessary condition[1]. The condition is too strong. Indeed, a timelock-free system is defined as "if an **Observer** is in state A, then it exists a future path where **Observer** jumps to state B" and so verifies instead the property :

```
A[]  (Observer.A => E<>Observer.B)
```

However, such a formula can't be verify in Uppaal.

[1] *A Tool for the Syntactic Detection of Zeno-timelocks in Timed Automata*, Howard Bowman, Rodolfo Gomez, and Li Su

2.3 Maximum delay between ventricular events

2.3.1 With an automaton and an error state

To verify that the delay between two ventricular events is never longer that LRI, we create an auxiliary automaton with a error state that is accessible if and only if it is the case. This automaton is called **LRI_1** (figure 7a).
Now, we can check that the state Error is never visited by **LRI_1** :

```
A[] not LRI_1.Error
```

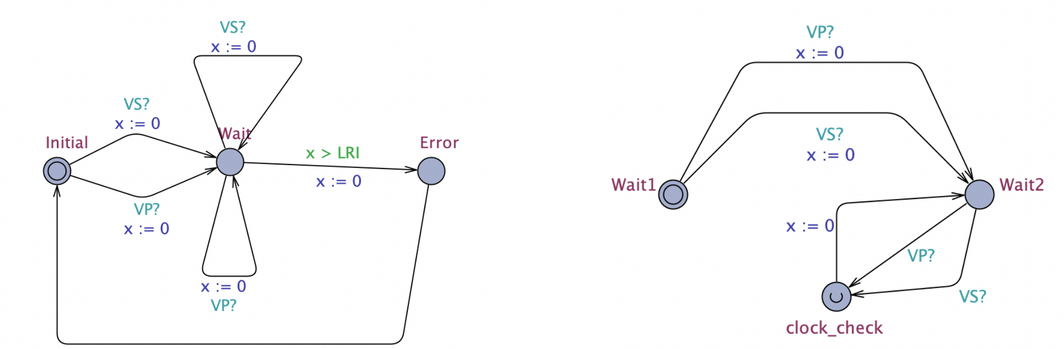
The property is satisfied.

2.3.2 Without error state

Alternatively, we can create another auxiliary automaton **LRI_2** (figure 7b).
Now, we can check that the local clock of this automaton never goes beyond LRI when a second ventricular signal is received :

```
A[] (LRI_2.clock_check imply LRI_2.x <= LRI)
```

The property is satisfied.



(a) **LRI_1** (b) **LRI_2**

Figure 7: Auxiliary automatons to check the maximum delay between ventricular events

2.4 Minimum delay between ventricular events

The pacemaker doesn't prevent from a heart beating too fast. Remember that the constant L_V has been set to 100. In theory, **HeartV** could send two *Vget* signals within 100 units of time; that doesn't respect the minimum delay between two ventricular events $URI = 400$.

2.4.1 Safe open-loop heart ($L_V \geq URI = 400$)

Let's set $L_V = 400$. To see if in this situation URI is respected, we use the previous automaton **LRI_2** and verify instead if the clock is always greater than URI :

```
A[] (LRI_2.clock_check imply LRI_2.x >= URI)
```

The property is satisfied and the delay between ventricular events is greater than URI when $L_V = 400$. It is quite clear that if we increase L_V, the minimum URI will remain respected.

2.4.2 Unsafe open-loop heart ($L_V < URI = 400$)

Let's set $L_V = 399$. The previous formula is not verified. However, it exists path that respects the minimum URI as the following property is true :

```
E[] (LRI_2.clock_check imply LRI_2.x >= URI)
```

By setting $L_V = 1$, we make the same observation. **To conclude, for $L_V < 400$, safe executions do exist.**

2.5 Bounded ventricular response to atrial event

Let's prove now that a atrial event is always followed by a ventricular response within an AVI units of time. For that, we create an auxiliary automaton called **ATV** (figure 8).

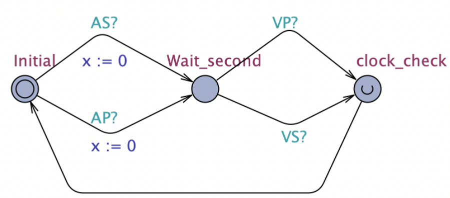


Figure 8: Auxiliary automaton **ATV**

As AVI may be extended, we will rather check if the delay is bounded by URI. Then, we verify the following formula :

```
A[] (ATV.clock_check imply ATV.x <= URI )
```

This property is satisfied.