

# Model checking based validation of cyber-physical systems

Partha Roop, Avinash Malik, Sidharta Andalam



BioRemediation™ Research Group

The University of Auckland

December 2016

[www.pretzel.ece.auckland.ac.nz/bio](http://www.pretzel.ece.auckland.ac.nz/bio)

# Outline

- 1 Acknowledgements
- 2 Introduction
- 3 Mathematical models
- 4 Specification/properties
  - Temporal Logics
  - Semantics
- 5 Model checking algorithm
- 6 Illustration
- 7 Timed Automata and Uppaal
- 8 The Cardiac Pacemaker Verification

## Some acknowledgements

- 1 Tony Zhao, Nathan Allen (Auckland University) for their joint work with me on pacemaker modelling and validation.
- 2 Uppaal model checking tools: <http://www.uppaal.org>.
- 3 For CTL model checking, I am using chapter 3 of a Springer book I co-authored [3].
- 4 The pacemaker case study is based on the TACAS 2012 paper from Upenn authors [2].
- 5 The Uppaal models used in this course is join work with my ME student Tony Zhao [4]. These are developed by adapting [2].

- 1 Students will appreciate the validation of a closed loop system comprising of timed automata models of the heart and pacemaker using model checking.
- 2 Students will understand the modelling of a pacemaker using timed automata and Uppaal.

## What is Verification?

The process of checking whether a system (software or hardware) conforms to or violates a pre-specified set of desired properties (often referred to as the requirements) is called *verification*. Methods used include:

- design review and code inspection
- (semi-)automated analysis of system.
- testing and automatic test case generation.
- model based testing.

Of these, testing is usually referred to as verification.

The techniques in the previous slide (usually based on testing) *try to find the presence of errors*. For safety critical systems, we need to prove the absence of certain errors. Formal verification techniques employ a method of proof over mathematical models of systems to *prove the absence of certain errors*.

## A simple illustration

**while 1 do**

    Mutexbegin::

    need[me] = true;

**while** need[other] **do**

**if** turn != me **then**

            need[me]=false;

**endif**

**endwhile**

    – critical section–

    –code of critical section here –

    Mutexend::

    need[me]=false;

    turn=other;

**endwhile**

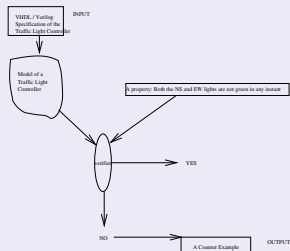
- Mutual exclusion - no two processes can enter the critical section simultaneously. [*safety property*]
- Deadlock freedom - processes are able to make progress. [*liveness property*]
- Eventual entry - every requesting process will be able to enter the critical section eventually. [*fairness property*]



Given:

- A Model of the Design,  $M$ .
- A property (expressed in a suitable *logic*).

Some method of proof is employed to check if the model *satisfies* the property. Two common methods of proof are *theorem proving* and *model checking*.



- ① Mathematical Model of the design.
- ② Language to express properties mathematically.
- ③ A Method of proof.

We will concentrate on *model checking* as a method of proof. Model checking is an algorithmic and fully automated method of proof for *finite* models of systems. Hence, it is an ideal tool for engineers.

## Mathematical Model

A mathematical model of concurrent systems (such a digital hardware) is a *Kripke structure* (a finite state machine variant ideal for formal proof).

### Definition (Kripke Structure)

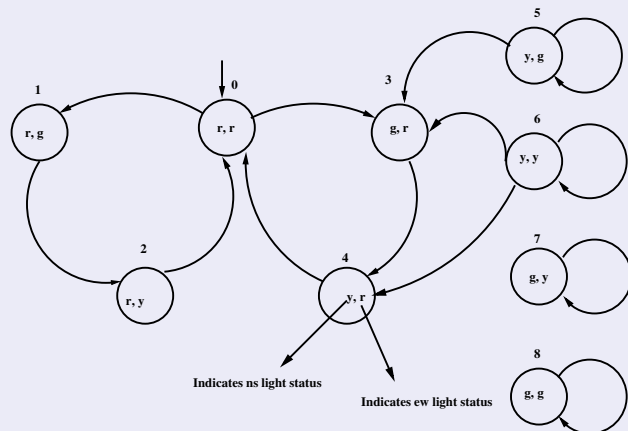
A Kripke Structure (KS) is defined as a tuple  $\langle AP, S, s_0, T, L \rangle$  where  $AP$  is the set of atomic propositions,  $S$  is the set of states with  $s_0$  being an unique start state,  $T \subseteq S \times S$  is the set of transition/edge relations between two states and  $L : S \rightarrow 2^{AP}$  is the labeling function mapping each state to a set of atomic propositions  $AP$  that holds at that state. We denote a transition  $(s_1, s_2) \in T$  as  $s_1 \rightarrow s_2$ .

### Definition (Path)

A path in a KS  $M$  starting from a state  $s$  is an infinite sequence of states  $\pi = s_0 s_1 s_2 \dots$  such that  $(s_i, s_{i+1}) \in T$  holds for all  $i \geq 0$ .

## Example: Model of a Traffic Light Controller

Let  $\mathcal{C} = \{r, y, g\}$ . Then, the state space of the controller is given by  $\mathcal{S} = \mathcal{C} \times \mathcal{C}$ .



## Requirements

- 1 Mathematical Model of the design.
- 2 Language to express properties mathematically.
- 3 A Method of proof.

Logic provides the the basis of verification methods. Like a programming language, a logic combines *syntax*, dictating how to write legal formulas, with *semantics*, which provide precise meaning to each formula.

Mathematical logic formalizes the notion of proof.

Logics may be classified as either:

- 1 Static Logics: The semantics of such logics remain fixed over time. Examples: propositional and predicate logic.
- 2 Dynamic Logics: Semantics of such logics evolve with time. Example: Modal logics such as temporal logics.

- A set of *propositions*.
- A set of *connectives*  $\{\wedge, \vee, \neg, \Rightarrow\}$ .

## Definition: Formula:

- Every proposition is a formula.
- If  $p$  is a formula then so is  $\neg p$ .
- If  $p, q$  are formulae then so are  $(p \wedge q)$ ,  $(p \vee q)$  and  $(p \Rightarrow q)$ .

Example:

$$((((p \Rightarrow q) \wedge (q \vee r)) \Rightarrow (p \vee r)) \Rightarrow \neg(q \vee s))$$

An interpretation assigns truth values to propositions appearing in the formula and then evaluates it.

P	Q	Implication
F	F	T
F	T	T
T	F	F
T	T	T

Table: Semantics of the “material implication”



A major limitation of propositional logic is poor expressibility due to the lack of variables and quantification.

- ① Today is a cloudy day. *Needs variables.*
- ② All cats are black. *Needs variables and quantification (universal quantifier).*
- ③ Some dogs are not black. *Needs variables and quantification (existential quantifier).*

Propositional and predicate logic are unsuitable to describe properties of dynamic systems (such as reactive systems). Temporal logic is a formalism for describing sequences of transition between states in a reactive system [1].

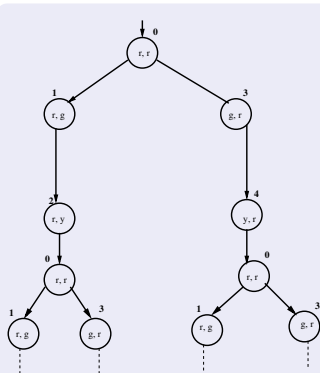
- 1 Linear Temporal Logics: Time is considered as a linearly ordered set, measuring it either with real or natural numbers. Example: LTL.
- 2 Branching Time Logics: The temporal order defines a tree which branches towards the future. Example: CTL, ACTL, CTL\*.

## Computation Tree

A computation tree is an infinite tree starting from a given state of a Kripke model. The tree is obtained by unwinding the model.

### Example:

A computation tree starting at node  $(r, r)$  of the traffic light example:



## Computation Tree Logic (CTL)

CTL formulas are defined over *paths* of a given infinite computation tree.

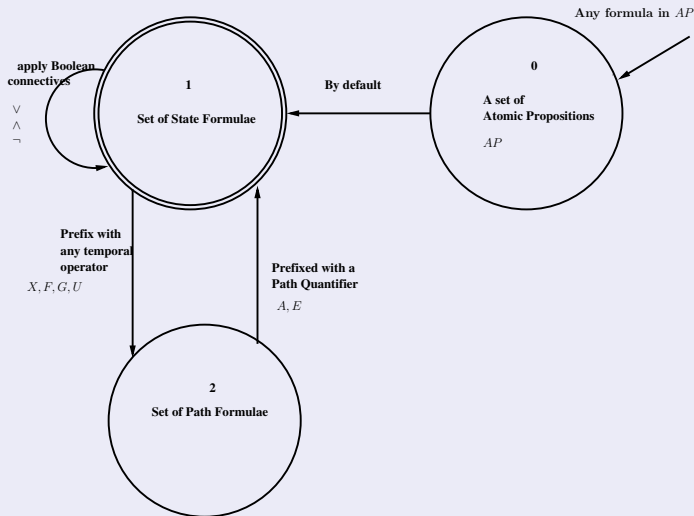
### Temporal Operators:

- 1  $X$ , the next time operator: this requires that a property holds in the second state of a given path.
- 2  $F$ , the eventually or in the future operator: this requires that a property hold in a future state of a given path.
- 3  $G$ , the always or the globally operator: this requires that a property hold along every state of a path.
- 4  $U$ , the until operator: it requires two properties ( $p_1 U p_2$ ) and requires that there is a state on the path where  $p_2$  holds and that  $p_1$  must hold along every preceding state.

### Path Quantifiers:

- 1  $E$ , the existential path quantifier.
- 2  $A$ , the universal path quantifier.

# CTL Syntax



CTL formulas is the set of all *state formulas* generated by the following definition.

### Definition (State Formulae)

- If  $p \in AP$  then  $p$  is a state formula.
- If  $\varphi_1$  and  $\varphi_2$  are state formulae, then  $\neg\varphi_1, (\varphi_1 \vee \varphi_2), (\varphi_1 \wedge \varphi_2)$  are also state formulae.
- If  $\varphi$  is a *path formula* then  $E\varphi$  and  $A\varphi$  are also state formulas, where path formula are defined below.

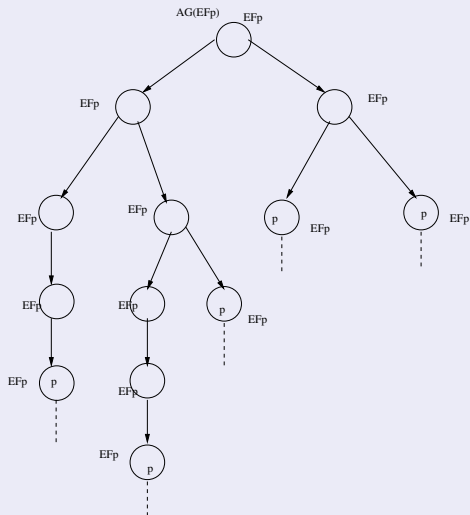
### Definition (Path Formulae)

If  $\varphi_1$  and  $\varphi_2$  are *state* formulae then  $X\varphi_1, F\varphi_1, G\varphi_1, (\varphi_1 U \varphi_2)$  are path formulae.

- ① Every request is eventually acknowledged:  
 $AG(req \Rightarrow AFack)$ .
- ② Along all computation paths in the future, GATE is high:  $AFgate$ .
- ③ Safety Property: Red light is shown to at least one direction all the time:  
 $AG([ns = r] \vee [ew = r])$ .
- ④ Liveness Property: Any direction will always eventually see the green light:  
 $AG(AF[ns = g]) \wedge AG(AF[ew = g])$ .

## Example CTL Property

$AG(EFp)$





**Semantics for CTL state property:**  $\varphi \rightarrow \text{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid E\psi \mid A\psi$

$$\llbracket \text{true} \rrbracket_M = S$$

$$\llbracket p \rrbracket_M = \{s \mid p \in L(s)\}$$

$$\llbracket \neg\varphi \rrbracket_M = S - \llbracket \varphi \rrbracket_M$$

$$\llbracket \varphi_1 \vee \varphi_2 \rrbracket_M = \llbracket \varphi_1 \rrbracket_M \cup \llbracket \varphi_2 \rrbracket_M$$

$$\llbracket E\psi \rrbracket_M = \{s \mid \exists \pi \in \text{PATH}(s) : \pi \in \llbracket \psi \rrbracket_M\}$$

$$\llbracket A\psi \rrbracket_M = \{s \mid \forall \pi \in \text{PATH}(s) : \pi \in \llbracket \psi \rrbracket_M\}$$

where

**Semantics of Path Property:**  $\psi \rightarrow X\varphi \mid F\varphi \mid G\varphi \mid \varphi \cup \varphi$

$$\llbracket X\varphi \rrbracket_M = \{\pi \mid \pi[1] \in \llbracket \varphi \rrbracket_M\}$$

$$\llbracket F\varphi \rrbracket_M = \{\pi \mid \exists i \geq 0 : \pi[i] \in \llbracket \varphi \rrbracket_M\}$$

$$\llbracket G\varphi \rrbracket_M = \{\pi \mid \forall i \geq 0 : \pi[i] \in \llbracket \varphi \rrbracket_M\}$$

$$\llbracket \varphi \cup \varphi_2 \rrbracket_M = \{s \mid \exists j \geq 0 : \pi[j] \in \llbracket \varphi_2 \rrbracket_M \wedge \forall i < j : \pi[i] \in \llbracket \varphi_1 \rrbracket_M\}$$

The 8 basic CTL operators

( $AX$ ,  $EX$ ,  $AF$ ,  $EF$ ,  $AG$ ,  $EG$ ,  $AU$ ,  $EU$ ) can all be expressed in terms of  $EX$ ,  $EG$ ,  $EU$ .

- $AXf = \neg EX(\neg f)$ .
- $AFf = \neg EG(\neg f)$ .
- $AGf = \neg EF(\neg f)$ .
- $A[fUg] = \neg E[\neg gU(\neg f \wedge \neg g)] \wedge \neg EG\neg g$ .
- $EFf = E[trueUf]$ .

## Requirements

- 1 Mathematical Model of the design.
- 2 Language to express properties mathematically.
- 3 A Method of proof.

### Definition (Model Checking)

Given a *KS* instance  $M$  and a CTL property  $\varphi$ , determine a  $S' \subseteq S$  such that  $S' = \{s \mid M, s \models \varphi\}$ . We say that  $M \models \varphi$  iff  $s_0 \in S'$ . In this case, we say that the model satisfies the property.

The algorithm works iteratively, labeling the states of  $M$  with sub-formulas in  $f$  that are true in a given state. Initially,  $label(s) = L(s)$ .

In the  $i$ th stage of the algorithm, sub-formulas with  $(i - 1)$  nested CTL operators are processed. When a given sub-formula is processed, it is added to the labeling of state  $s$  if it holds in  $s$ .

Once the algorithm terminates, we have  $M, s \models f$  if  $f \in label(s)$ .

### Example of nesting:

Let  $f = AF(EX[ew = g])$ . This can be rewritten as:  $\neg EG[\neg EX(ew = g)]$

The nesting of CTL operators in the above formula is:

- i=1:  $(ew = g)$
- i=2:  $EX(ew = g)$
- i=3:  $\neg EX(ew = g)$
- Let  $q = \neg EX(ew = g)$
- i=4:  $EGq$
- i=5:  $\neg EGq$

Since all CTL formulas can be expressed using  $\neg, \vee, \wedge, EX, EG, EU$  there are 6 cases to handle:

- ①  $g$ : an atomic proposition.
- ②  $\neg f_1$
- ③  $f_1 \vee f_2$
- ④  $EXf_1$
- ⑤  $E[f_1 U f_2]$
- ⑥  $EGf_1$

- For formulas of the form  $g = \neg f_1$ , we label all states that are not labelled by  $f_1$ .
- For formulas of the form  $g = f_1 \vee f_2$ , we label all states that are labelled by either  $f_1$  or  $f_2$ .
- For formulas of the form  $g = EXf_1$ , we label all states that have some successor labelled by  $f_1$ .
- For formulas of the form  $g = E[f_1 U f_2]$ , we apply algorithm  $checkEU(f_1, f_2)$ .
- For formulas of the form  $g = EGf_1$ , we apply algorithm  $checkEG(f_1)$ .

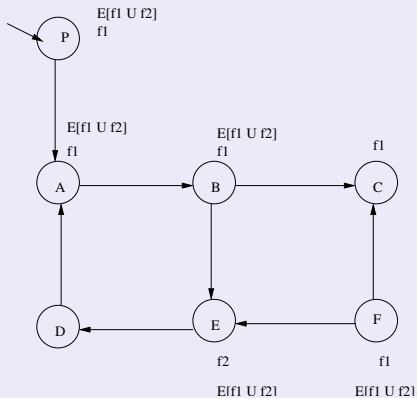
## checkEU()

```
procedure checkEU( $f_1, f_2$ )  
   $T := \{s \mid f_2 \in \text{label}(s)\}$   
  for each  $s \in T$  do  
     $\text{label}(s) := \text{label}(s) \cup \{E[f_1 U f_2]\};$   
  endfor  
  while  $T \neq \emptyset$  do  
    choose  $s \in T$ ;  
    for all  $t$  such that  $R(t, s)$  do  
      if  $E[f_1 U f_2] \notin \text{label}(t)$  and  $f_1 \in \text{label}(t)$  then  
         $\text{label}(t) := \text{label}(t) \cup \{E[f_1 U f_2]\};$   
         $T := T \cup \{t\};$   
      endif  
    endfor  
  endwhile  
end procedure
```



## Example

$p = E[f1 \cup f2]$



This algorithm is based on decomposition of  $M$ 's graph into nontrivial strongly connected components.

**Definition:**

A **strongly connected component (SCC)**  $C$  is a *maximal subgraph* such that every node in  $C$  is reachable from every other node in  $C$  along a directed path entirely contained within  $C$ .  $C$  is **nontrivial** iff it has more than one node or it contains one node with a self loop.

For the algorithm, we construct a structure  $M'(S', R', L')$  from  $M$  as follows:

- 1  $S' = \{s \in S \mid M, s \models f_1\}$  e.g., we delete all states that don't satisfy  $f_1$ .
- 2  $R' = R|_{S' \times S'}$
- 3  $L' = L|_{S'}$

The algorithm depends on the following lemma.

**Lemma:**

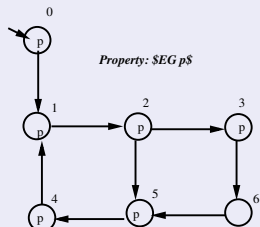
$M, s \models EGf_1$  iff the following conditions are satisfied:

- 1  $s \in S'$
- 2 There exists a path in  $M'$  that leads from  $s$  to some node  $t$  in a nontrivial SCC  $C$  of the graph  $(S', R')$ .

## checkEG()

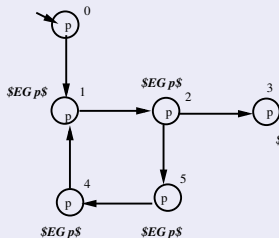
```
procedure checkEG( $f_1$ )
  //subgraph creation
   $S' := \{s \mid f_1 \in \text{label}(s)\}$ 
   $\text{SCC} = \{C \mid C \text{ is a nontrivial SCC of } S'\}$ 
  //group all nodes in any nontrivial SCC
  //which already satisfy  $f_1$ 
   $T := \bigcup_{C \in \text{SCC}} \{s \mid s \in C\}$ 
  for each  $s \in T$  do
     $\text{label}(s) := \text{label}(s) \cup \{EGf_1\}$ ;
  endfor
  while  $T \neq \emptyset$  do
    choose  $s \in T$ ;
     $T := T \setminus \{s\}$ ;
    //perform backward reachability from members of  $T$ 
    for all  $t$  such that  $t \in S'$  and  $R(t, s)$  do
      if  $EGf_1 \notin \text{label}(t)$  then
         $\text{label}(t) := \text{label}(t) \cup \{EGf_1\}$ ;
         $T := T \cup \{t\}$ ;
      endif
    endfor
  endwhile
end procedure
```

# Example

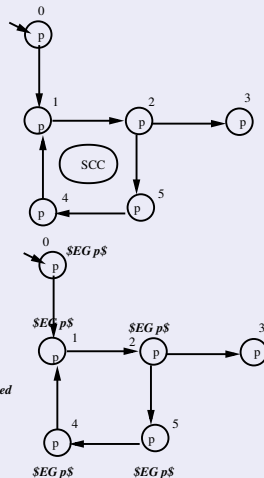


1. Remove states and transitions not satisfying  $\$p\$$

2. Label all SCC states with  $\$EG\ p\$$



3. Label any state from which a  $\$EG\ p\$$ -labelled state can be reached



## Example of Traffic Light Controller

Let  $f = AF(EX[ew = g])$ . This can be rewritten as:  $\neg EG[\neg EX(ew = g)]$

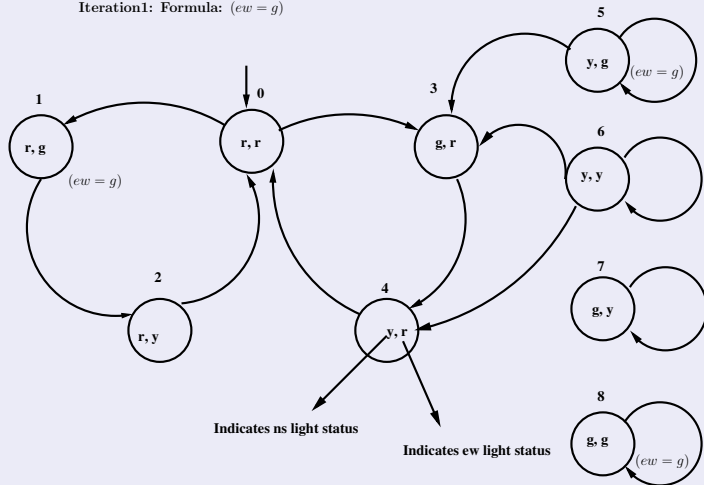
The nesting of CTL operators in the above formula is:

- i=1:  $(ew = g)$
- i=2:  $EX(ew = g)$
- i=3:  $\neg EX(ew = g)$
- Let  $q = \neg EX(ew = g)$
- i=4:  $EGq$
- i=5:  $\neg EGq$

## Example..

### step $i=1$

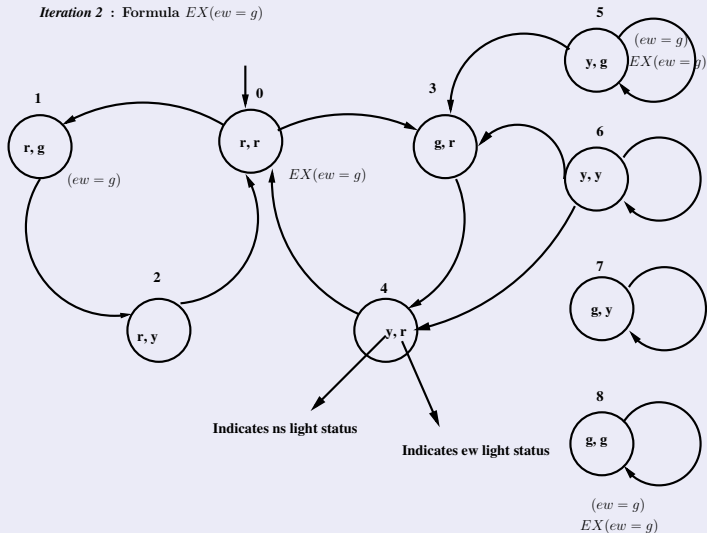
Iteration1: Formula:  $(ew = g)$



Example..

step  $i=2$

Iteration 2 : Formula  $EX(ew = g)$

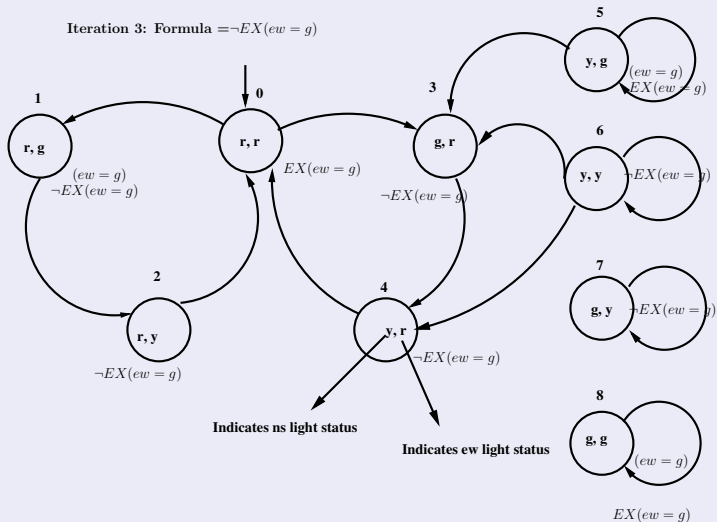




## Example..

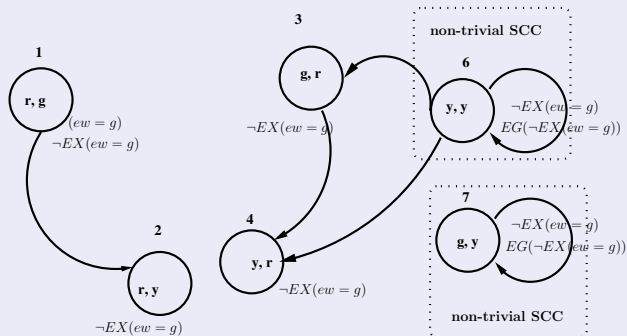
### step i=3

Iteration 3: Formula  $\neg EX(ew = g)$



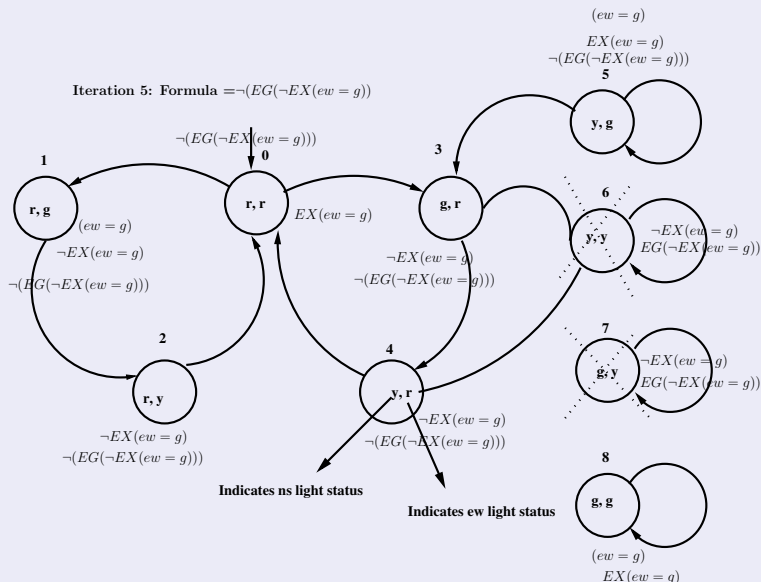
## step i=4

Iteration 4: Formula  $=EG(\neg EX(ew = g))$



## Example..

step i=5



- Real-time systems are modelled using timed automata (TA), which capture a sub-class of hybrid systems.
- TA: The progression of time is captured using a set of real-valued clocks.
- All clocks progress at the same rate i.e.  $\dot{t} = 1$  for any clock  $t$ .

## Definition (Clock constraints)

Let  $x, y, .. \in \mathcal{C}$  denote clocks and let  $a, b, .. \in \Sigma$  denote actions. Let  $\sim \in \{\leq, <, =, >, \geq\}$ ,  $n \in \mathbb{N}$ . Then, a clock constraint is of the form  $x \sim n$  or  $x - y \sim n$ .




## Definition (Timed Automaton (TA))

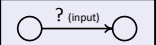
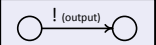
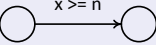
A timed automaton  $\mathcal{A}$  is defined as a tuple  $\langle N, l_0, E, I \rangle$  where:

- $N$  is a finite set of locations, also called nodes,
- $l_0 \in N$  is the initial location.
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$  is the set of edges and,
- $I : N \rightarrow \mathcal{B}(\mathcal{C})$  assigns invariants to locations.

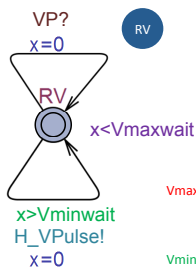
We denote a transition  $\langle l, g, a, r, l' \rangle \in E$  as  $l \xrightarrow{g, a, r} l'$ .

# Types of states and transitions in Uppaal

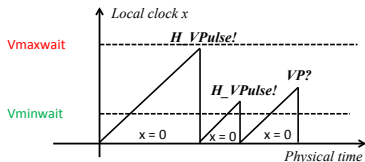
<i>Locations</i>	<i>Descriptions</i>
 Normal	<b>Normal</b> locations are used for describing commonly used states, in which time ticks unless instantaneous transitions take place.
 Initial	<b>Initial</b> locations. The start of an automaton is this location.
 Committed	<b>Committed</b> locations. When in the committed location, the only possible transition is always the one going out of the committed state. The committed location has to be left immediately.

<i>Transitions</i>	<i>Descriptions</i>
<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">  </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">  </div> <div>  </div>	<p><b>Broadcast Channels</b></p> <p><b>Broadcast channels</b> are used for synchronization in a one-producer-multiple consumer situation, and commonly seen in the automata network.</p> <p><b>Clock Guards</b></p> <p><b>Clock guards</b> are used for describing clock-driven transitions.</p>

# The Random Heart Model

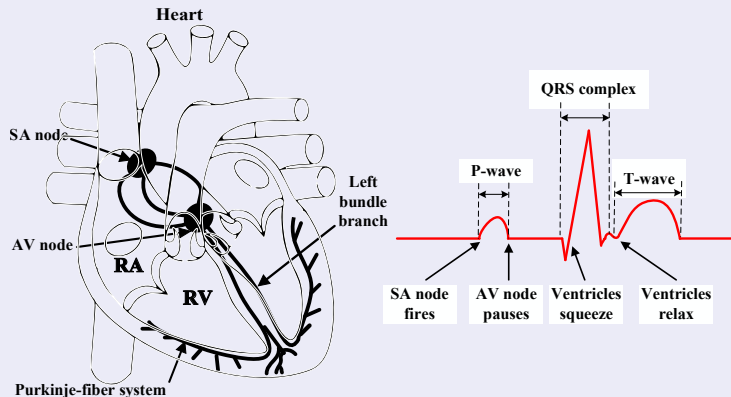


- Input:  $VP$
- Output:  $H\_VPulse$
- Local Clock:  $x$
- Clock Bound:  $x \in [Vminwait, Vmaxwait]$



- The random heart model generates  $H\_APulse$ ,  $H\_VPulse$  randomly in the interval  $[Minwait, Maxwait]$ .
- Progression of time modelled using real-valued clock variables  $Clock1$ ,  $Clock2$ .
- Invariants introduce *fairness* i.e. locations have to be exited before invariants become false.

## The human heart – a real-time system

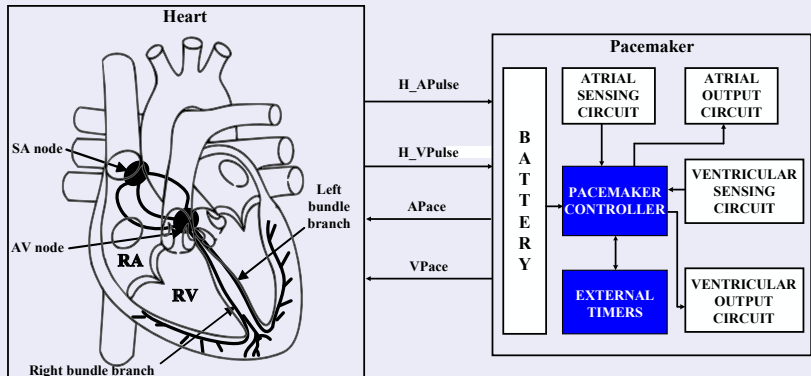


a

<sup>a</sup>Zhao and Roop, "Model Driven Design of Cardiac Pacemakers using IEC61499, CRC Press, 2015".



## The closed-loop pacing system

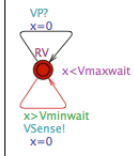


<sup>a</sup>

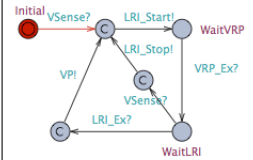
<sup>a</sup>Zhao and Roop, "Model Driven Design of Cardiac Pacemakers using IEC61499, CRC Press, 2015".

# Overall Uppaal Model – VVI Mode

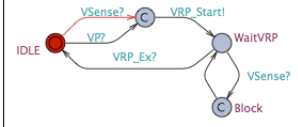
Random\_Ventricle



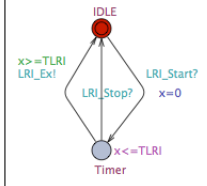
Con\_LRI



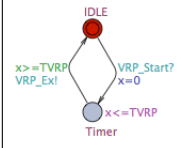
Con\_VRP



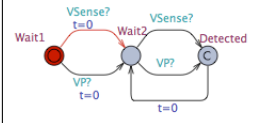
Timer\_LRI



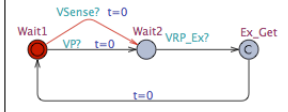
Timer\_VRP



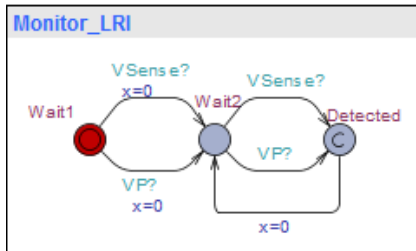
Monitor\_LRI



Monitor\_VRP

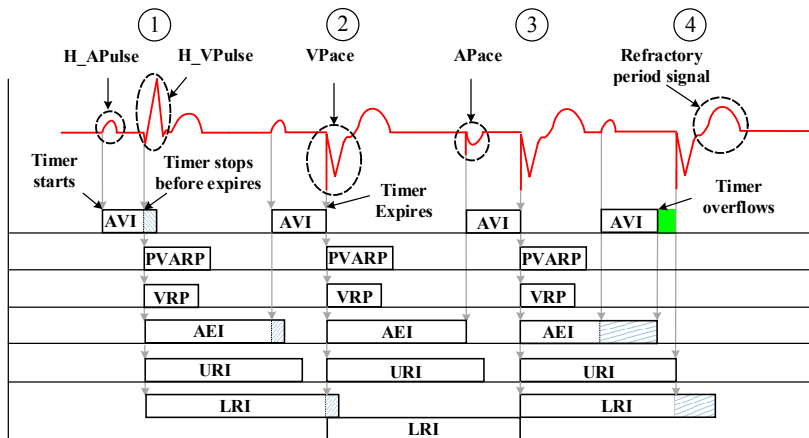


## Monitors and TCTL properties



- A monitor is a timed automaton that observe the system execution to reach a specific “location”. A clock is used to measure the time lapse from some initial event to a final event when this location is reached.
- For example, in the LRI monitor, we measure the time lapse between any two consecutive ventricular events using a clock called  $t$ . In this example, we reach a specific “committed location”, called Detected.
- Then we verify the TCTL property:  
 $A[] (Monitor\_LRI.Detected \text{ imply } Monitor\_LRI.t \leq TLRI)$

## DDD mode – timing diagram



<sup>a</sup>Zhao and Roop, “Model Driven Design of Cardiac Pacemakers using IEC61499, CRC Press, 2015”.

## Next: Uppaal Demo

- 1 I will demo the Uppaal tool.
- 2 We will discuss the VVI and DDD modes of the pacemaker.

- ➊ In this lecture we learnt about temporal logic, CTL.
- ➋ We learnt about explicit state model checking.
- ➌ We learnt about how to extend the approach for the verification of timed systems.
- ➍ We learnt about TCTL model checking using Uppaal.
- ➎ We present the verification of VVI and DDD modes using Uppaal.

- ➊ From timed to hybrid system modelling using hybrid automata.
- ➋ Cardiac cell modelling and associated detailed and abstract models.
- ➌ Modelling of the cardiac conduction system.
- ➍ Forward and backward conduction.
- ➎ Demonstration using Simulink.



E. M. Clarke, Orna Grumberg, and Doron A. Peled.

*Model Checking.*

The MIT Press, 1999.



Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam.

Cyber–physical modeling of implantable cardiac medical devices.

*Proceedings of the IEEE*, 100(1):122–137, 2012.



Roopak Sinha, Parthasarathi Roop, and Samik Basu.

*Correct-by-construction approaches for SoC design.*

Springer, 2014.



Yu Zhao and Partha S Roop.

Model-driven design of cardiac pacemaker using IEC 61499 function blocks.

*Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499*, 9:335, 2016.