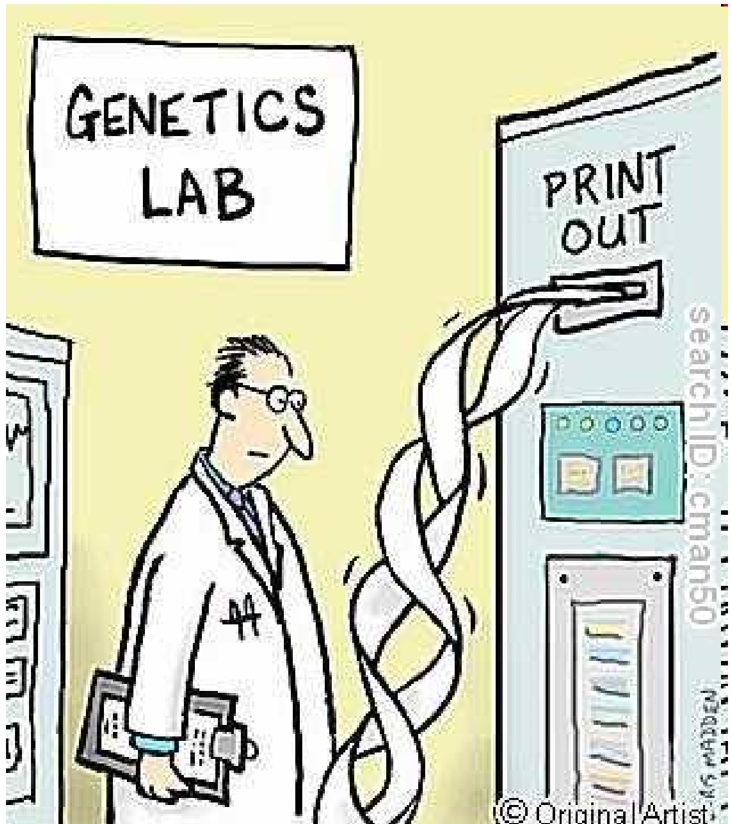
An Introduction to Model Checking

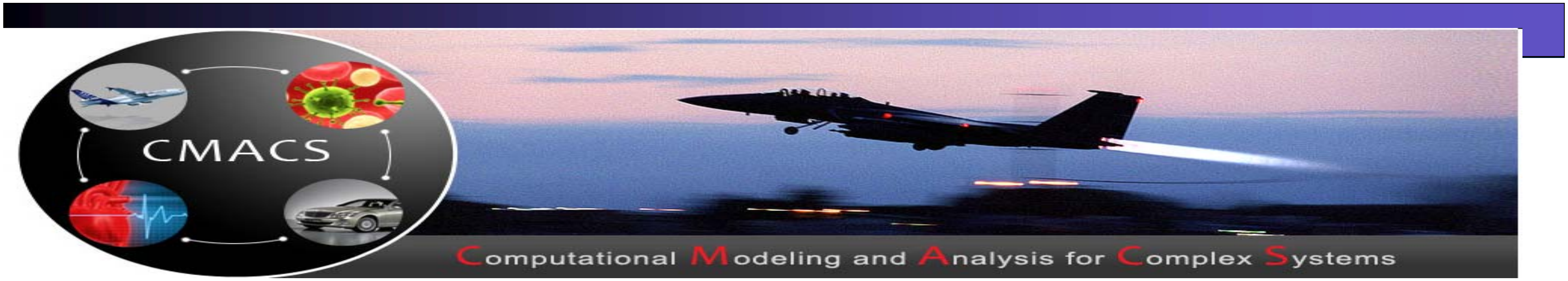
Scott A. Smolka
Stony Brook University



© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

Outline

- What is CMACS?
- What is Model Checking?
- Model Checking Example
- Applying MC to Heart Cells



- 5-Year, $10M NSF Expedition in Computing Award
- 7 academic institutions + NASA JPL
- 18 Principal Investigators, with Ed Clarke of CMU the Lead Investigator
- Seek to apply next-generation Model Checking and Abstract Interpretation techniques to Biological and Embedded Systems
- Challenge problems in ***Pancreatic Cancer***, ***Atrial Fibrillation***, and ***Automotive*** & ***Aerospace*** systems

Model Checking

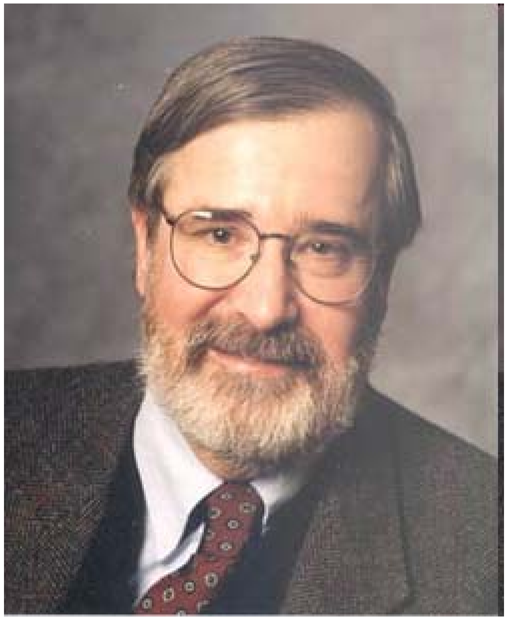
$$M \overset{?}{\models} \varphi$$

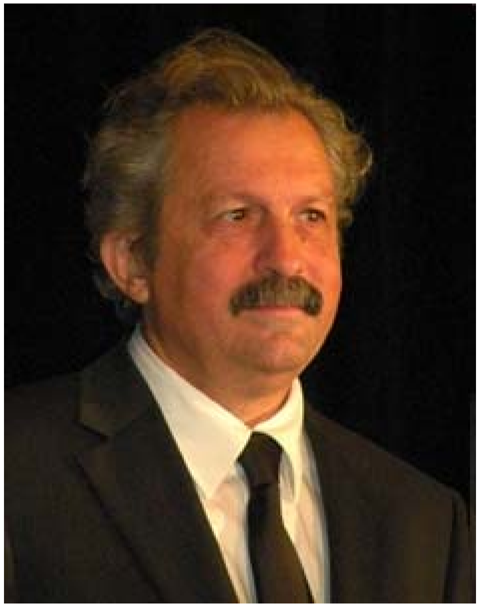
Does system model **M** satisfy system property **φ**?

M given as a ***state machine***.

φ usually specified in ***temporal logic***.

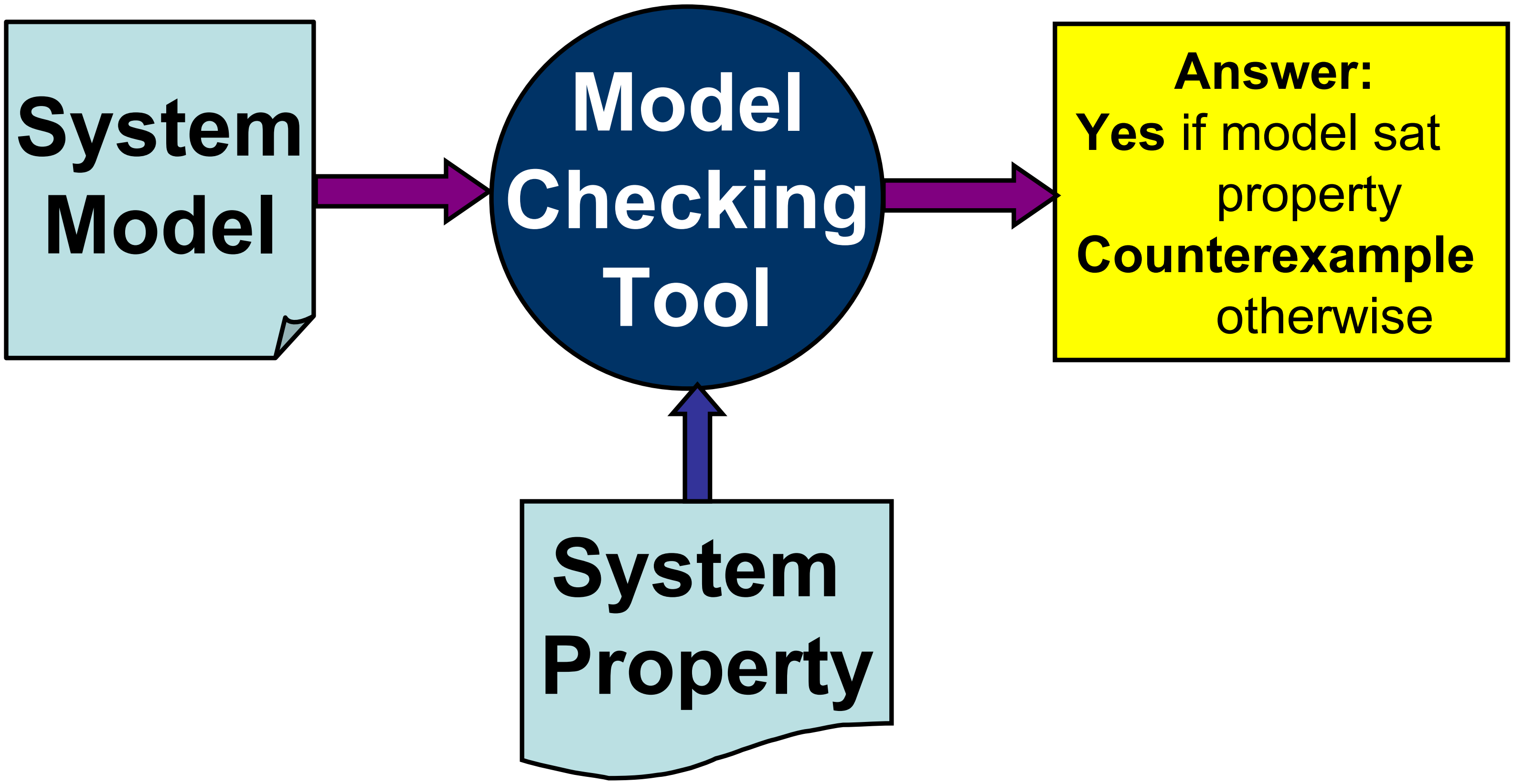
Clarke Emerson Sifakis Receive 2007 Turing Award





… they developed this fully automated approach [Model Checking] that is now the most widely used verification method in the hardware and software industries.

Model Checking



What is Model Checking?

[Clarke & Emerson 1981]:

"Model checking is an automated technique that, given a finite-state model of a system and a logical property, systematically checks whether this property holds for (a given initial state in) that model."

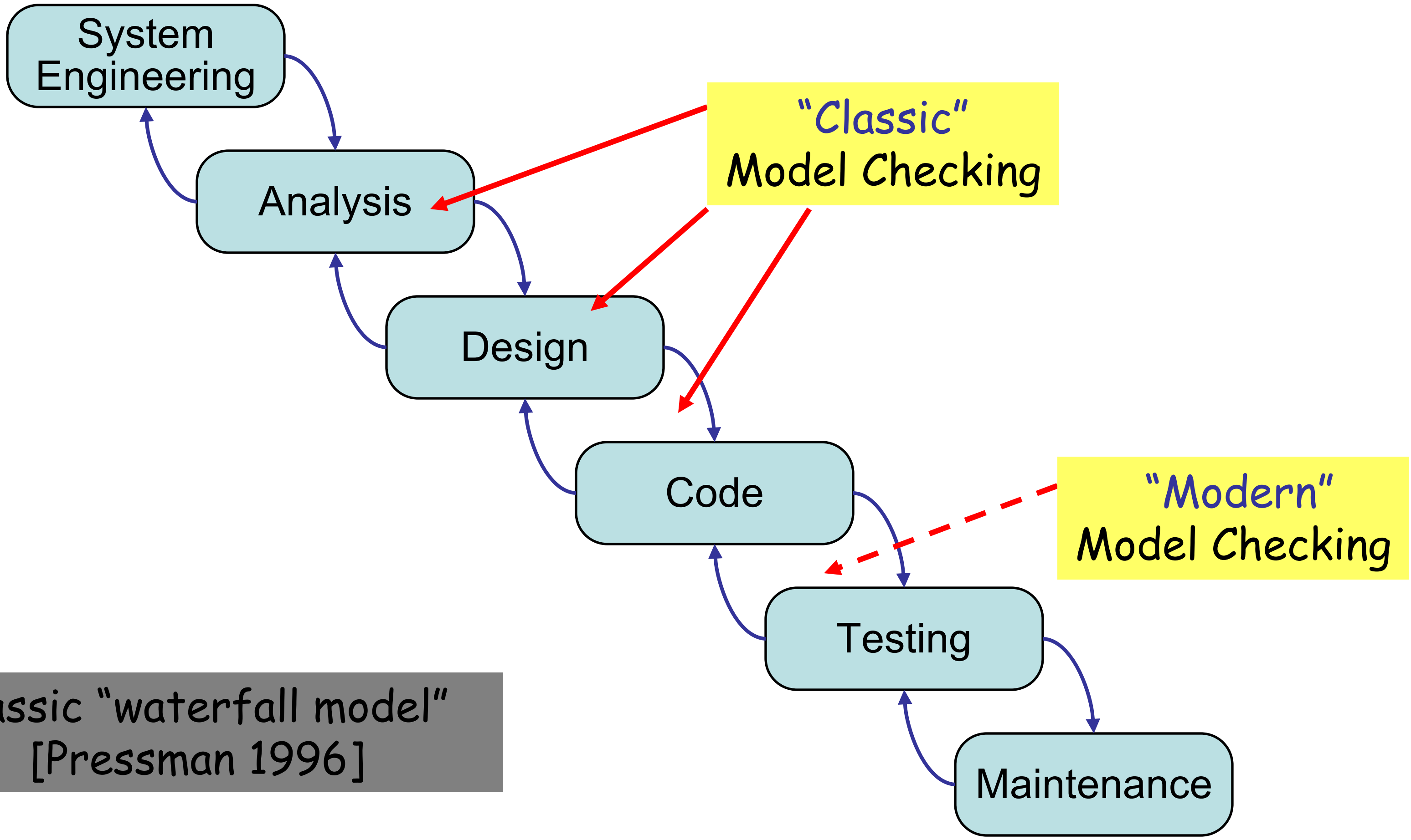
Model checking tools automatically verify whether $\mathbf{M} \models \boldsymbol{\varphi}$ holds, where *M* is a (finite-state) model of a system and property φ is stated in some formal notation.

Problem: state space explosion! Although finite-state, the model of a system typically grows exponentially.

Common Design Flaws in Concurrent Systems

- Deadlock
- Livelock, starvation
- Underspecification
  - unexpected reception of messages
- Overspecification
  - Dead code
- Violations of constraints
  - Buffer overruns
  - Array bounds violations
- Assumptions about speed
  - Logical correctness vs. real-time performance

System Development

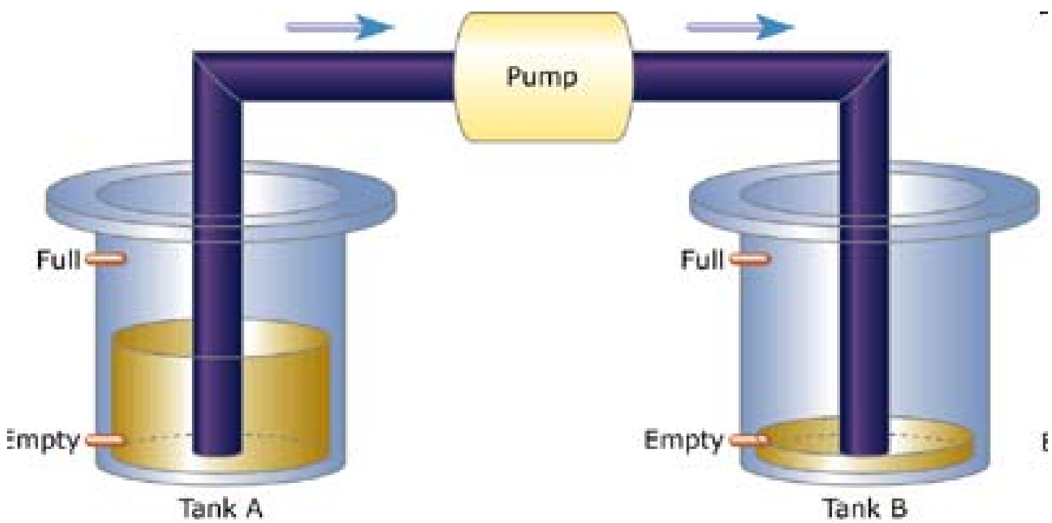


Classic "waterfall model" [Pressman 1996]

The SMV Model Checker

- Developed at CMU in the 1990s
- System model given as an FSA
- System properties given as CTL formulas
- SMV program has 3 parts:
  - (finite) variable declarations
  - (nondeterministic) variable assignments
  - property specification

A Simple Two-Tank Pumping System



Source Tank

Sink Tank

Pump System Specification

- Initially, both tanks are empty.
- **Pump** switched on as soon as water level in **tank A** reaches ok, provided **tank B** not full.
- **Pump** remains on as long as **tank A** not empty and **tank B** not full.
- **Pump** switched off as soon as **tank A** empty or **tank B** full.
- System should not attempt to switch the **pump** off (on) if it's already off (on).

Pumping System Specification (Part I)

Comments

```
MODULE main

VAR
  level_a : {empty, ok, full}; -- source tank
  level_b : {empty, ok, full}; -- sink tank
  pump : {on, off};
```

Pumping System Specification (Part II)

```
ASSIGN
  next(level_a) := case
     level_a = empty : {empty, ok};
     level_a = ok & pump = off : {ok, full};
     level_a = ok & pump = on : {ok, empty, full};
     level_a = full & pump = off : full;
     level_a = full & pump = on : {ok, full};
     1 : {ok, empty, full};
  esac;
```

Pumping System Specification (Part III)

```
next(level_b) := case
    level_b = empty & pump = off : empty;
    level_b = empty & pump = on : {empty, ok};
    level_b = ok & pump = off : {ok, empty};
    level_b = ok & pump = on : {ok, empty, full};
    level_b = full & pump = off : {ok, full};
    level_b = full & pump = on : {ok, full};
    1 : {ok, empty, full};
  esac;
```

Pumping System Specification (Part IV)

```
next(pump) := case
      pump = off & (level_a = ok | level_a = full) &
      (level_b = empty | level_b = ok) : on;
      pump = on & (level_a = empty | level_b = full) : off;
      1 : pump; -- keep pump status as it is
  esac;

INIT
  (pump = off)
```

Pumping System Specification (Part V)

SPEC

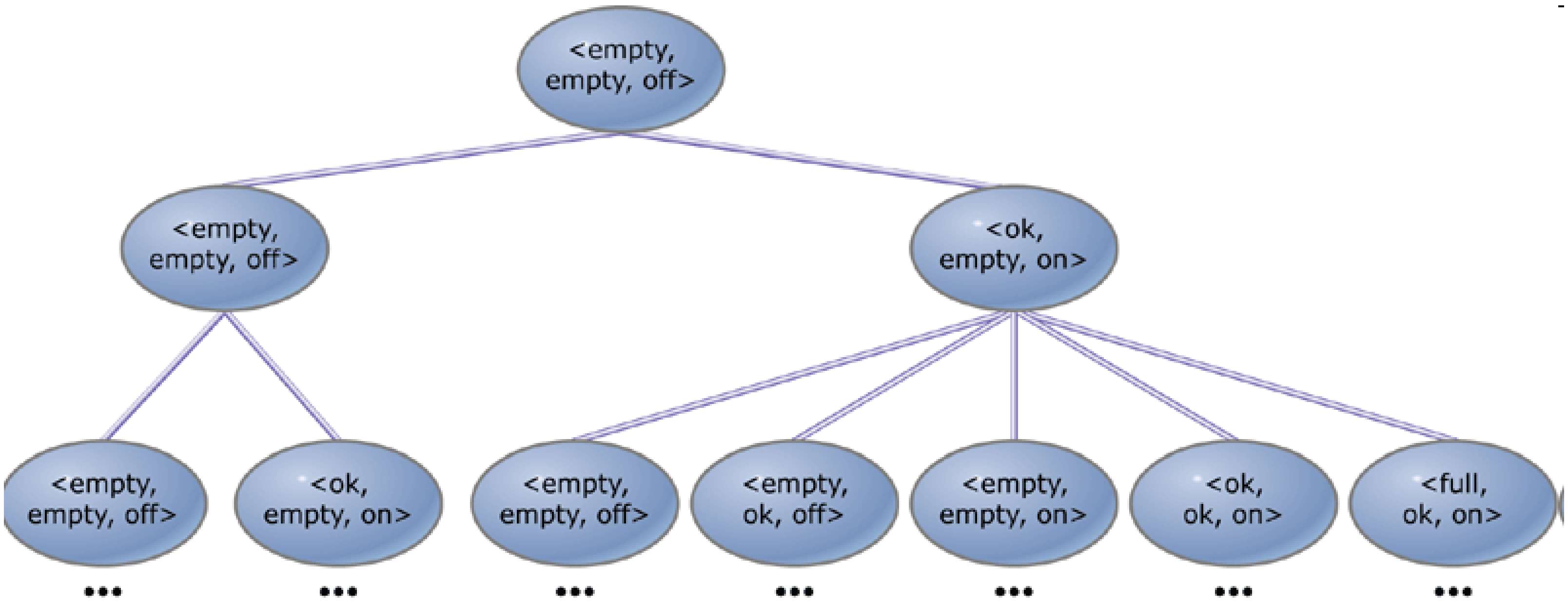
```
-- pump is always off if source tank is empty or sink tank is full
AG AF (pump = off -> (level_a = empty | level_b = full))

-- always possible to reach a state when the source tank is ok or full
AG EF (level_b = ok | level_b = full)
```

Model Executions

- Initially, system could be in any of nine states where no restrictions on water level in A or B but the pump is off
- Denote a state by an ordered tuple <A,B,P> where A and B are current water level in tank A and B, and P is current pump status
- Assume initial state to be <empty,empty,off>
- Next state could be <empty,empty,off> or <ok,empty,on>
- From <ok,empty,on>, next state could be <ok,empty,on>, <ok,ok,on>, <full,empty,on>, <full,ok,on>, <empty,empty,off>, or <empty,ok,off>.
- For each of these states, we could calculate next possible states

Initial Portion of Execution Tree



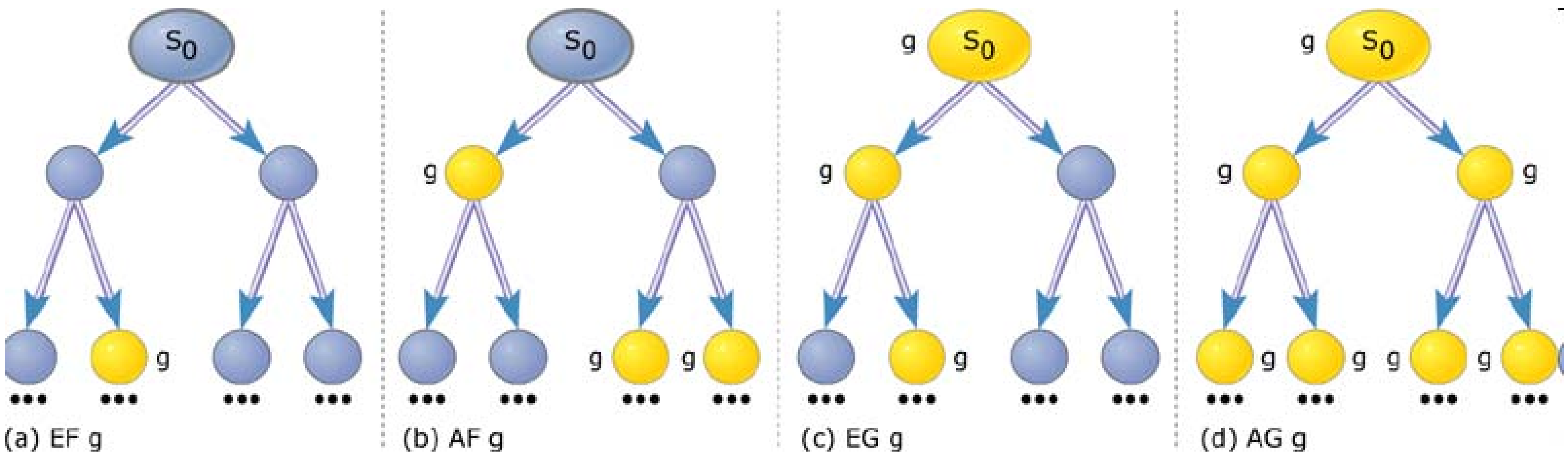
CTL Operators

The temporal logic CTL allows us to specify properties of paths (and states along paths) of an execution tree. It is an extension of Boolean propositional logic.

EX φ	**true** in current state if formula φ is **true** in at least one of the next states
EF φ	**true** in current state if there exists some state in some path beginning in current state that satisfies the formula φ
EG φ	**true** in current state if every state in some path beginning in current state satisfies the formula φ
AX φ	**true** in current state if formula φ is **true** in every one of the next states
AF φ	**true** in current state if there exists some state in every path beginning in current state that satisfies the formula φ
AG φ	**true** in current state if every state in every path beginning in current state satisfies the formula φ

E (for some path) and **A** (for all paths) are ***path quantifiers*** for paths beginning from a state. **F** (for some state) and **G** (for all states) are ***state quantifiers*** for states along a path.

Intuition for CTL Operators



Simple CTL Properties of Pump System

- **AF (pump = on)**. For every path beginning at initial state, there's state in that path at which pump is on.
- **False**, since there's a path from initial state in which the pump always remains off (e.g., if tank A forever remains empty).
- SMV generates following **counterexample**. (Loop indicates infinite sequence of states beginning at initial state such that tank B is full in every state of path and hence pump is off.)

SMV Counterexample

-- specification AF pump = on is false

-- as demonstrated by following execution sequence

-- loop starts here

state 1.1:

level_a = full

level_b = full

pump = off

state 1.2:

Another Simple CTL Property

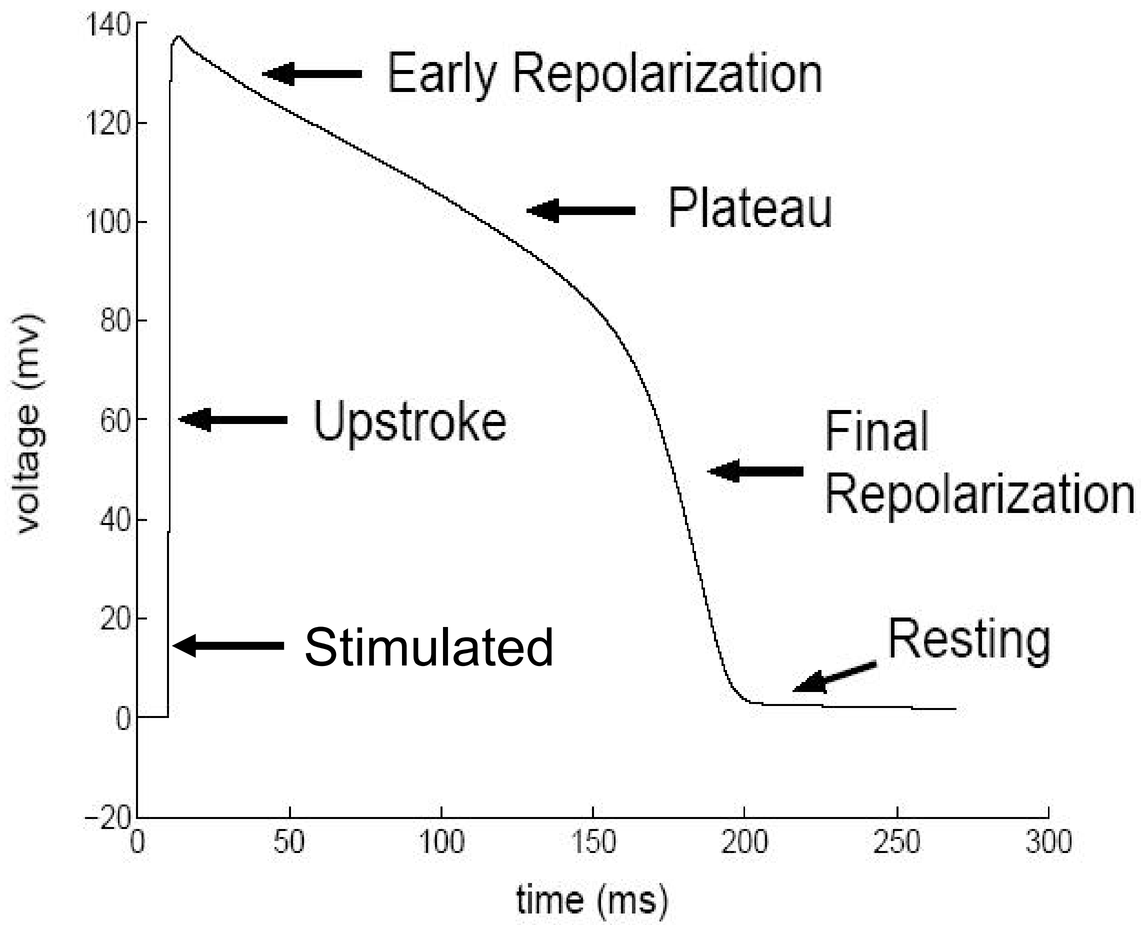
- Dual property **AF (pump = off)**. For every path beginning at initial state, there's a state in that path at which the pump is off.
- **Trivially true**, since in the initial state itself (which is included in all paths) **pump = off** is true.

What does MC have to do with Bio?

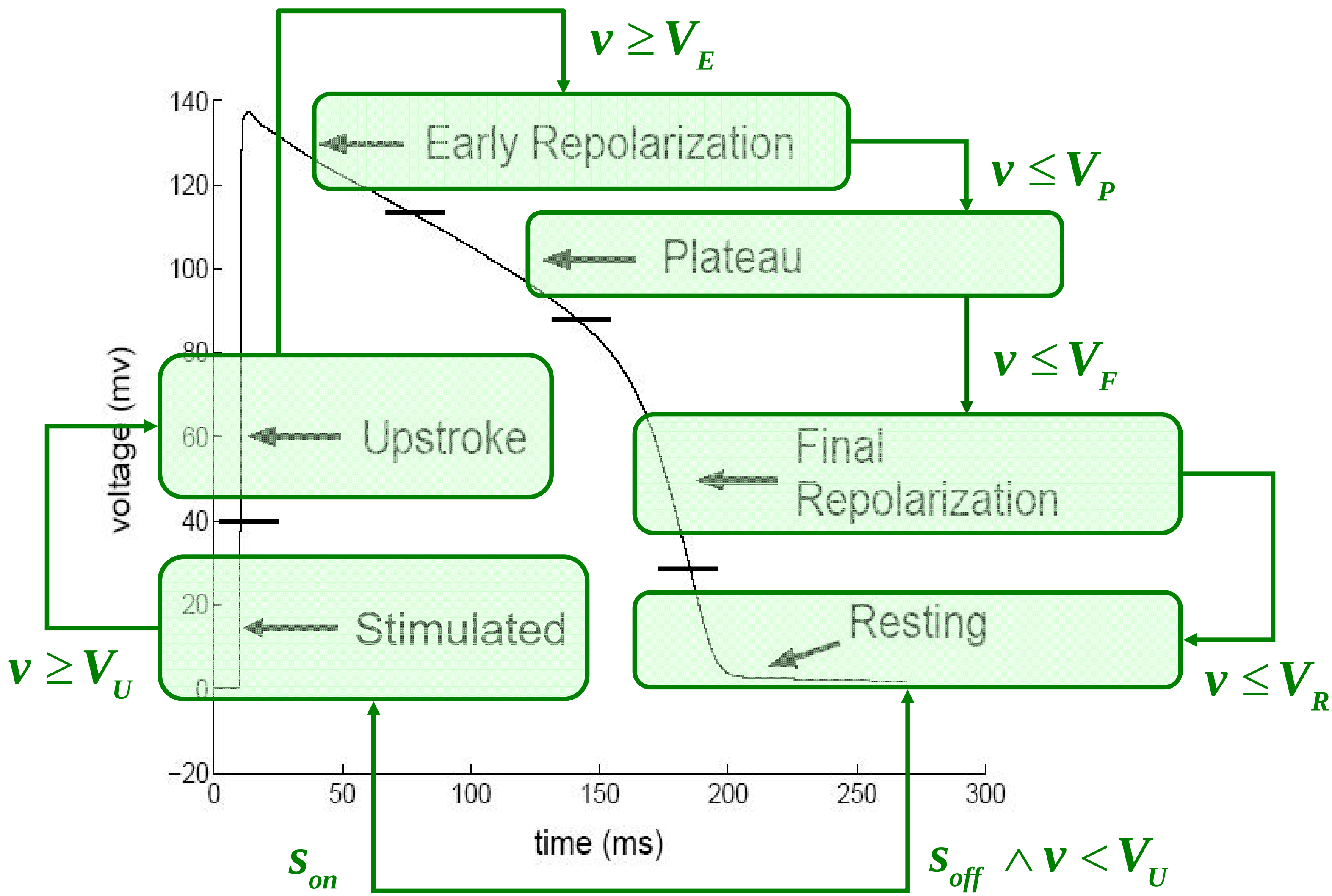
- And what does it have to do with **heart cells**, and **atrial fibrillation**, and … ?

- Can view Flavio’s minimal model as a special kind of state machine and try to apply MC to that!

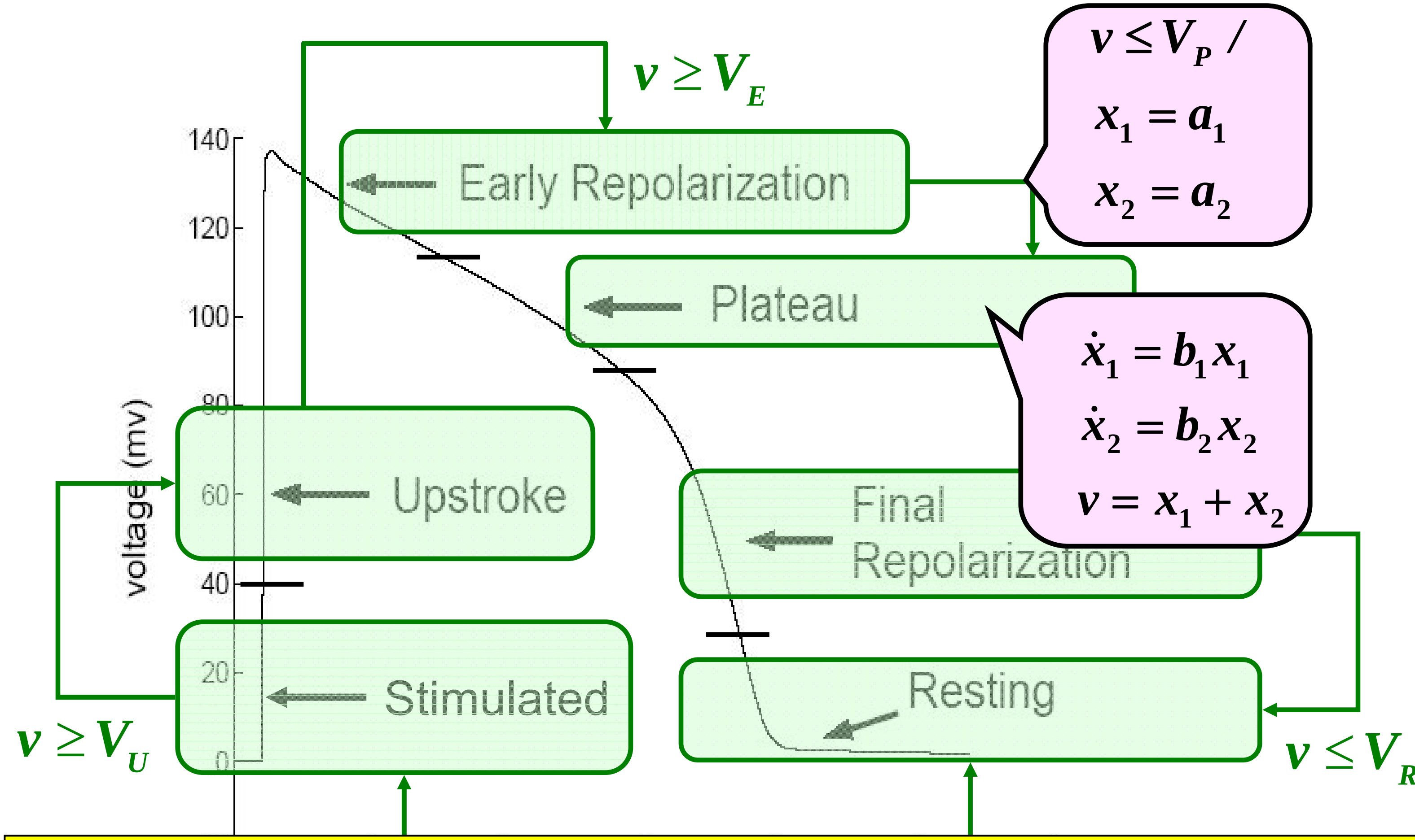
Hybrid Automaton Model: Cardiac Cell



Hybrid Automaton Model: Cardiac Cell

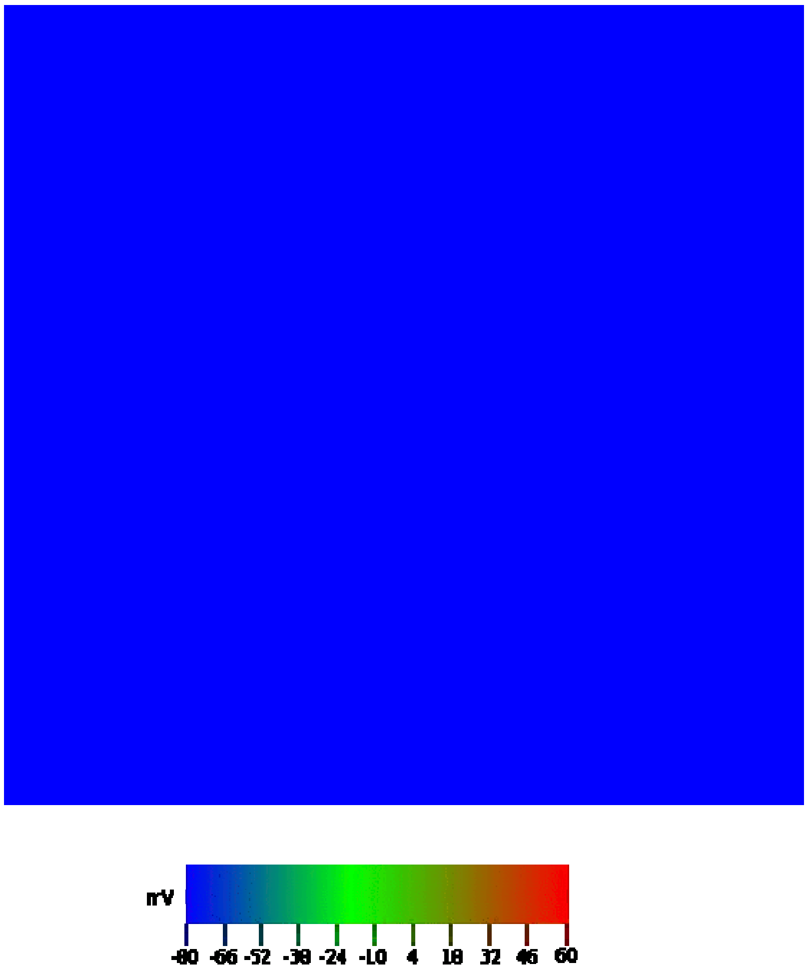


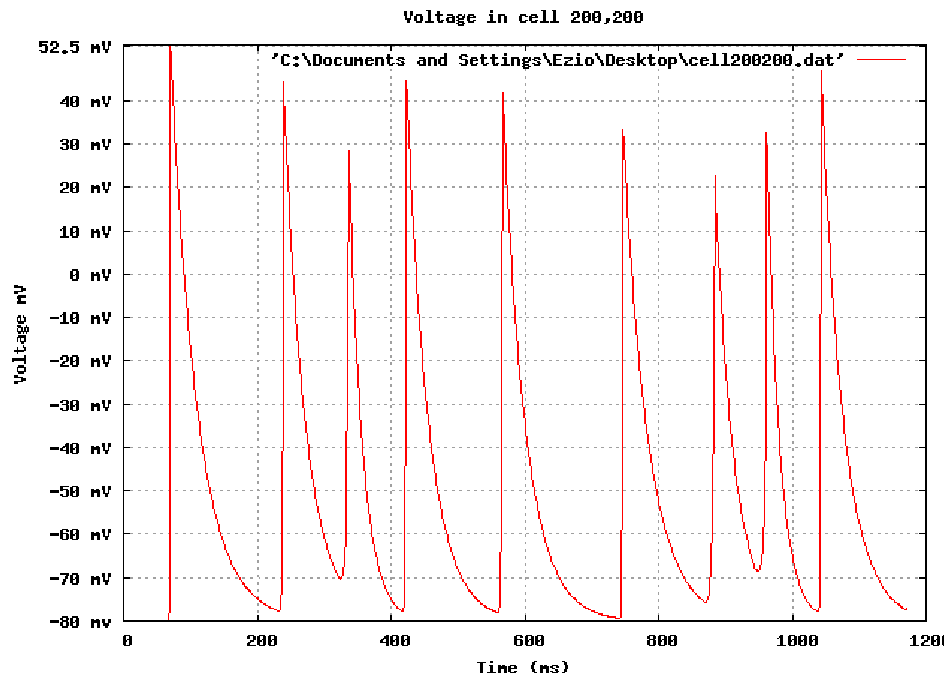
Hybrid Automaton Model: Cardiac Cell



P. Ye, E. Entcheva, S.A. Smolka and R. Grosu. **A Cycle-Linear Hybrid-Automata Model for Excitable Cells**. *IET Systems Biology, vol. 2(1), pp. 24-32, January, 2008.*

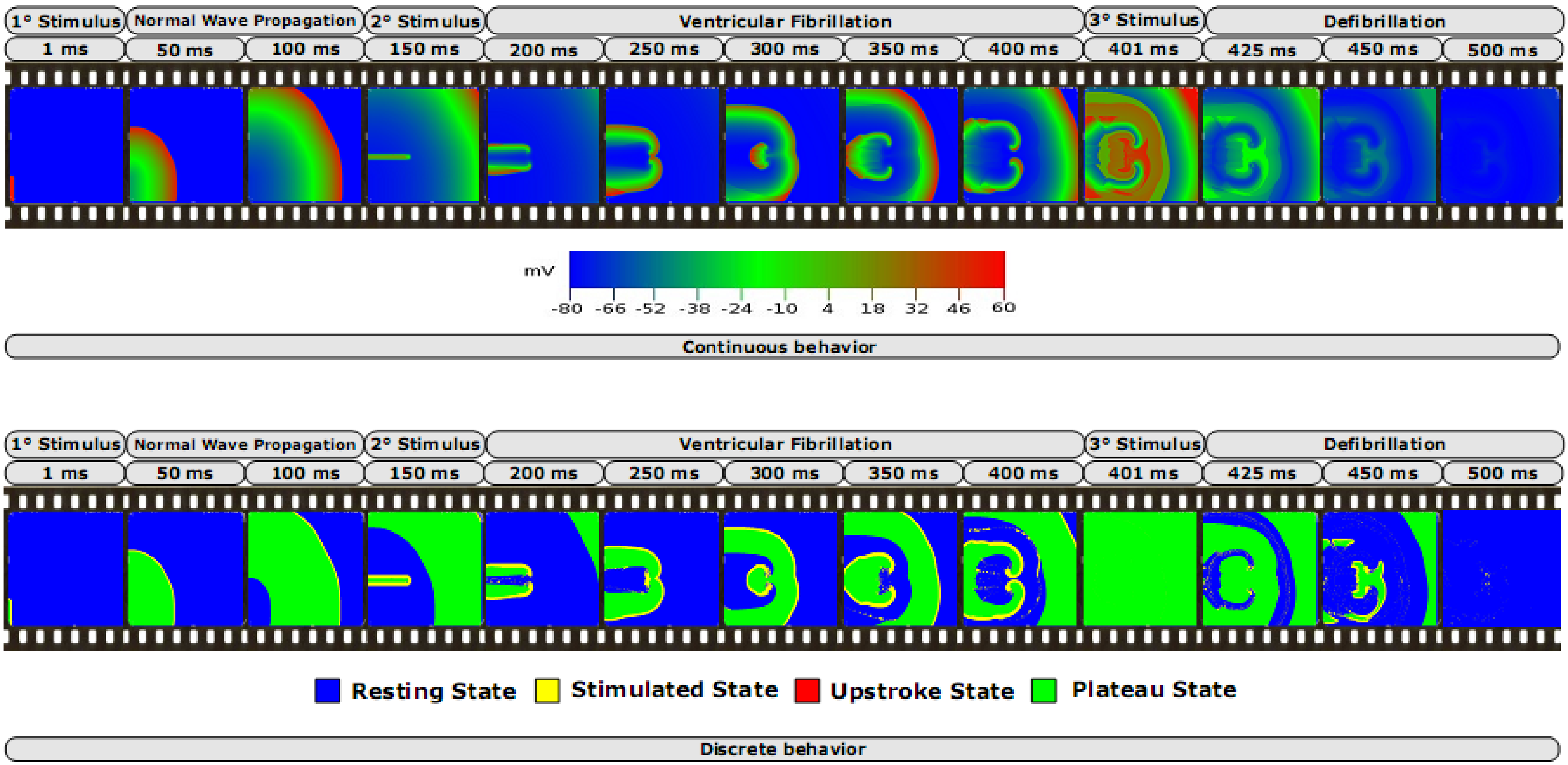
HA Network (Spatial) Simulation





- Fibrillation/Defibrillation protocol
- 400 x 400 HA cell array

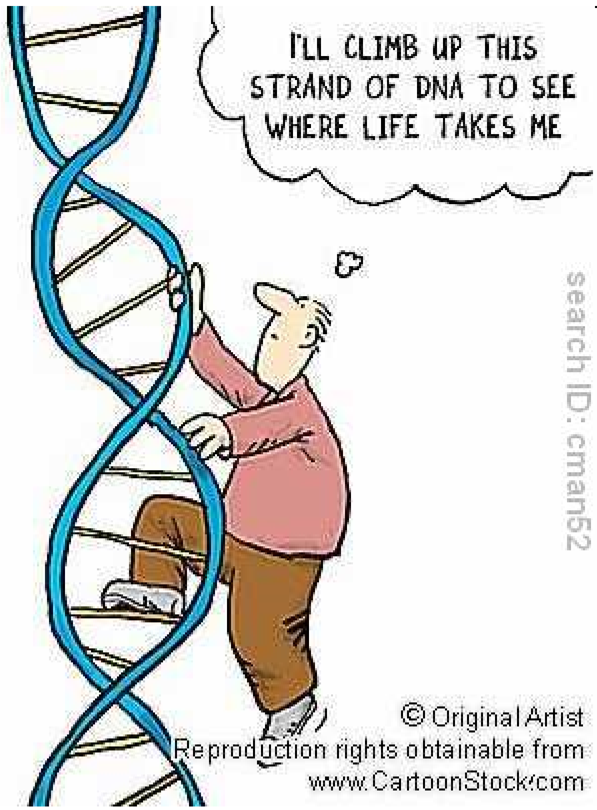
(Finite) Mode Abstraction



- Preserves spatial properties ($4^{160,000}$ images)

CMACS Wants You!

- NSF REUs
- Summer Internships
- RAs in CMACS graduate programs


I'LL CLIMB UP THIS STRAND OF DNA TO SEE WHERE LIFE TAKES ME
search ID: cman52
© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

Emergent Behavior in Cardiac Tissue