Sensing everywhere:

on quantitative verification for ubiquitous computing

Marta Kwiatkowska
University of Oxford

ECSS 2014, Wroclaw, 14th October 2014
Based on 2012 Milner Lecture, University of Edinburgh

Where are computers?

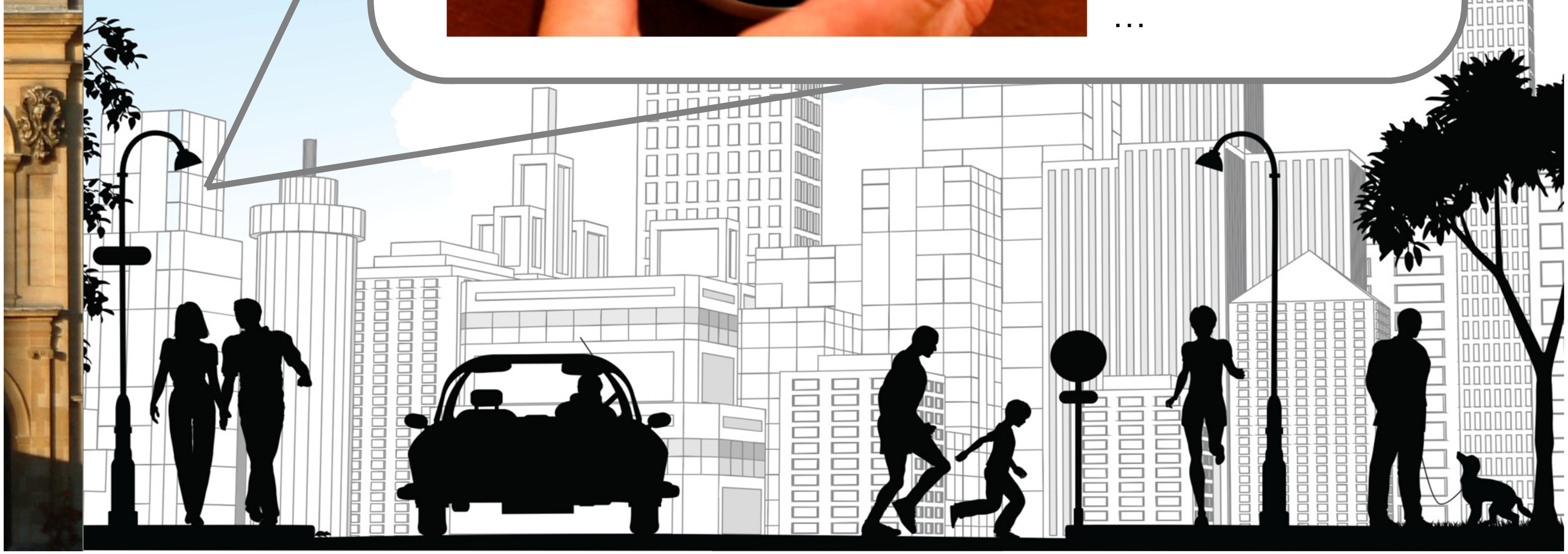
Once upon a time, back in the 1980s…

Smartphones, tablets, …

Smart homes

Internet of Things

-Home network
-Internet-enabled appliances
-Remote control
-Smart energy management
…



Smart cars

Intelligent vehicles

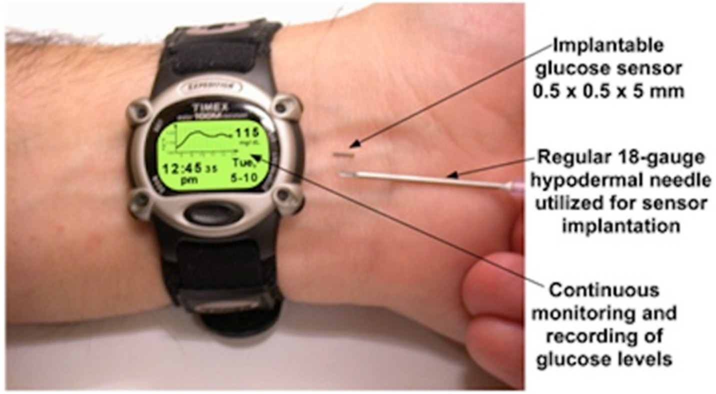
-Self-parking cars
-Driverless cars
-Search and rescue
-Unmanned missions
...

Smart wearables

Personalised health monitoring

-Heart rate
-Accelerometer
-Health tracking
-Fitness apps
…

Smart implantable medical devices…



Monitoring and treatment of diseases

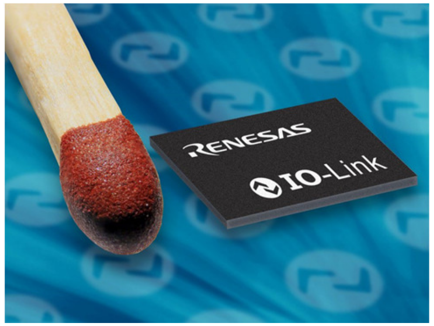
-Glucose level
-Heart rate
-Blood pressure
…

Ubiquitous computing

- Computing without computers

- Populations of sensor-enabled computing devices that are
  - embedded in the environment, or even in our body
  - sensors for interaction and control of the environment
  - software controlled, can communicate
  - operate autonomously, unattended
  - devices are mobile, handheld or wearable
  - miniature size, limited resources, bandwidth and memory
  - organised into communities

- Unstoppable technological progress
  - smaller and smaller devices, more and more complex scenarios, increasing take up…



Perspectives on ubiquitous computing

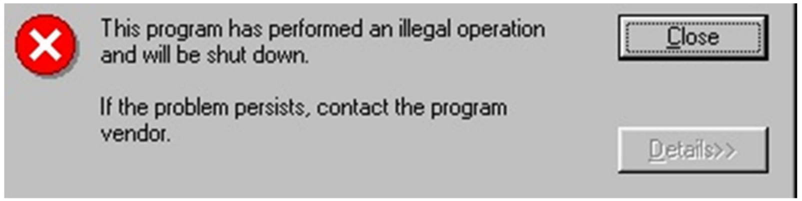
- Technological: calm technology [Weiser 1993]
  - "The most profound technologies are those that disappear. They weave themselves into everyday life until they are indistinguishable from it."

- Usability: 'everyware' [Greenfield 2008]
  - Hardware/software evolved into 'everyware': household appliances that do computing
- Scientific: "Ubicomp can empower us, if we can understand it" [Milner 2008]
  - "What concepts, theories and tools are needed to specify and describe ubiquitous systems, their subsystems and their interaction?"

- This lecture: from theory to practice, for Ubicomp
  - emphasis on practical, algorithmic techniques and industrially-relevant tools

Are we safe?

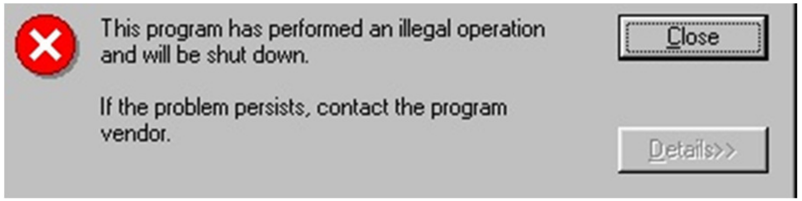
- Embedded software at the heart of the device



- What if…
  - self-parking car software crashes during the manouvre
  - health monitoring device fails to trigger alarm

Are we safe?

- Embedded software at the heart of the device



- What if…
  - self-parking car software crashes during the manouvre
  - health monitoring device fails to trigger alarm

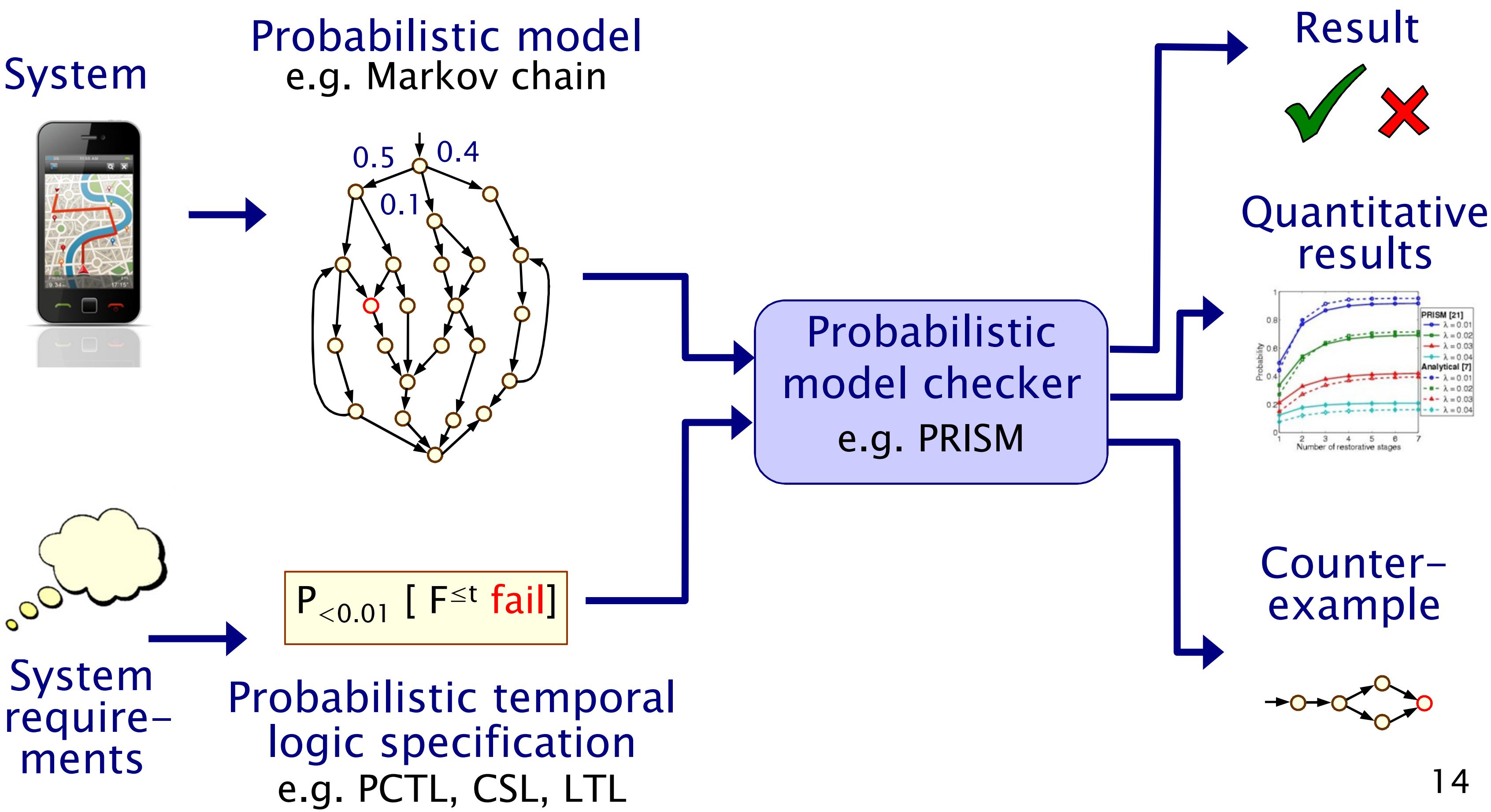
- Imagined or real?
  - February 2014: Toyota recalls 1.9 million Prius hybrids due to software problems
  - Jan-June 2010 “Killed by code”: FDA recalls 23 defective cardiac pacemaker devices because they can cause adverse health consequences or death, six likely caused by software defects

Software quality assurance

- Software is an integral component
  - performs critical, lifesaving functions and basic daily tasks
  - software failure costly and life endangering
- Need quality assurance methodologies
  - model-based development
  - rigorous software engineering
- Use formal techniques to produce guarantees for:
  - safety, reliability, performance, resource usage, trust, …
  - (safety) “heart rate never drops below 30 BPM”
  - (energy) “energy usage is below 2000 mA per minute”
- Focus on automated, tool-supported methodologies
  - automated verification via model checking
  - quantitative/probabilistic verification

Quantitative (probabilistic) verification then

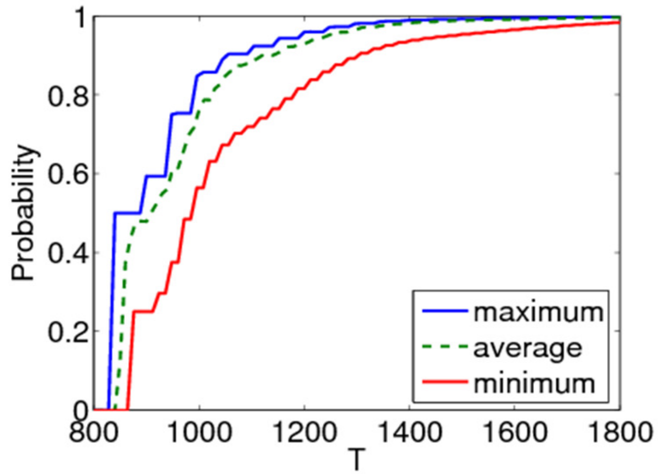
Automatic verification (aka model checking) of quantitative properties of probabilistic system models



Why quantitative verification?

- Real ubicomp software/systems are quantitative:
  - Real-time aspects
    - hard/soft time deadlines
  - Resource constraints
    - energy, buffer size, number of unsuccessful transmissions, etc
  - Randomisation, e.g. in distributed coordination algorithms
    - random delays/back-off in Bluetooth, Zigbee
  - Uncertainty, e.g. communication failures/delays
    - prevalence of wireless communication

- Analysis "quantitative" & "exhaustive"
  - strength of mathematical proof
  - best/worst-case scenarios, not possible with simulation
  - identifying trends and anomalies



Quantitative properties

- Simple properties
  - $P_{\leq 0.01}$ [F "fail"] – "the probability of a failure is at most 0.01"

- Analysing best and worst case scenarios
  - $P_{max=?}$ [$F^{\leq 10}$ "outage"] – "worst-case probability of an outage occurring within 10 seconds, for any possible scheduling of system components"
  - $P_{=?}$ [$G^{\leq 0.02}$!"deploy" {"crash"}{max}] – "the maximum probability of an airbag failing to deploy within 0.02s, from any possible crash scenario"

- Reward/cost-based properties
  - $R_{\{\text{“time”}\}=?}$ [F "end"] – "expected algorithm execution time"
  - $R_{\{\text{“energy”}\}max=?}$ [$C^{\leq 7200}$] – "worst-case expected energy consumption during the first 2 hours"

From verification to synthesis…

- Automated verification aims to establish if a property holds for a given model
- Can we find a model so that a property is satisfied?
  - difficult, especially for quantitative properties…
  - advantage: correct-by-construction
- We initially focus on simpler problems
  - strategy synthesis
  - parameter synthesis
  - template-based synthesis
- Many application domains
  - robotics (controller synthesis from LTL/PCTL)
  - security (generating attacks)
  - dynamic power management (optimal policy synthesis)

Historical perspective

- First algorithms proposed in 1980s
  - [Vardi, Courcoubetis, Yannakakis, …]
  - algorithms [Hansson, Jonsson, de Alfaro] & first implementations
- 2000: tools ETMCC (MRMC) & PRISM released
  - PRISM: efficient extensions of symbolic model checking [Kwiatkowska, Norman, Parker, …]
  - ETMCC (now MRMC): model checking for continuous-time Markov chains [Baier, Hermanns, Haverkort, Katoen, …]
- Now mature area, of industrial relevance
  - successfully used by non-experts for many application domains, but full automation and good tool support essential
    - distributed algorithms, communication protocols, security protocols, biological systems, quantum cryptography, planning…
  - genuine flaws found and corrected in real-world systems

Tool support: PRISM

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
  - continuously updated and extended
- Support for four probabilistic models:
  - models: DTMCs, CTMCs, MDPs, PTAs, …
  - properties: PCTL, CSL, LTL, PCTL*, costs/rewards …
- Features:
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
  - adopted and used across a multitude of application domains
  - 90+ case studies
- See: http://www.prismmodelchecker.org/

The challenge of ubiquitous computing

- Quantitative verification is not powerful enough!

- Necessary to <u>model communities and cooperation</u>
  - add self-interest and ability to form coalitions
- Need to <u>monitor and control physical processes</u>
  - extend models with continuous flows
- Important to <u>interface to biological systems</u>
  - consider computation at the molecular scale…

- In this lecture, focus on the above directions
  - each demonstrating transition from theory to practice
  - formulating novel verification algorithms
  - resulting in new software tools, beyond PRISM…

Focus on...

Cooperation & competition

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control

Natural world

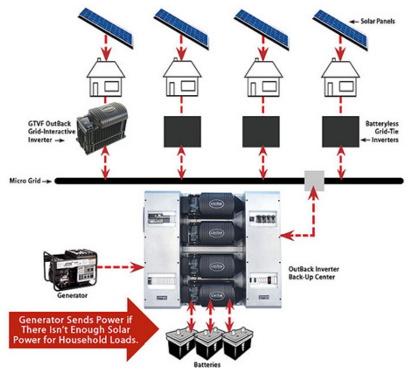
- Biosensing
- Molecular programming

Modelling cooperation & competition

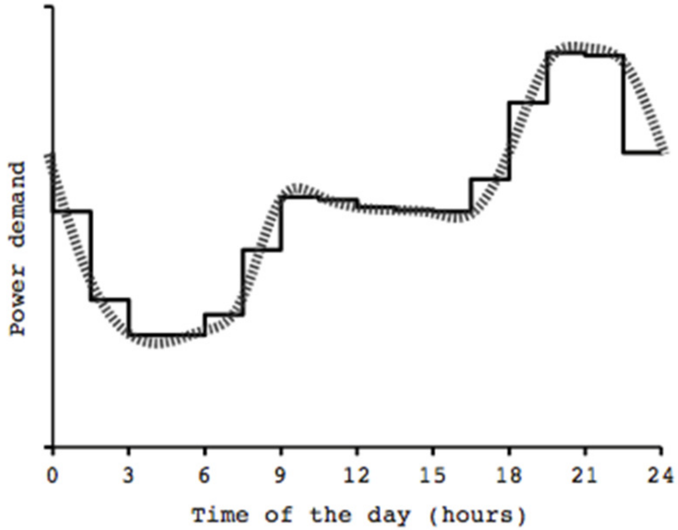
- Ubicomp systems are organised into communities
  - self-interested agents, goal driven
  - need to cooperate, e.g. in order to share bandwidth
  - possibly opposing goals, hence competititive behaviour
  - incentives to increase motivation and discourage selfishness
- Many typical scenarios
  - e.g. user-centric networks, energy management or sensor network co-ordination
- Natural to adopt a game-theoretic view
  - widely used in computer science, economics, …
  - here, distinctive focus on algorithms and temporal logic specification/goals
- Research question: can we automatically verify cooperative and competitive behaviour? synthesise winning strategies?

Case study: Energy management

- Energy management protocol for Microgrid
  - Microgrid: local energy management
  - randomised demand management protocol [Hildmann/Saffre'11]
  - probability: randomisation, demand model, …



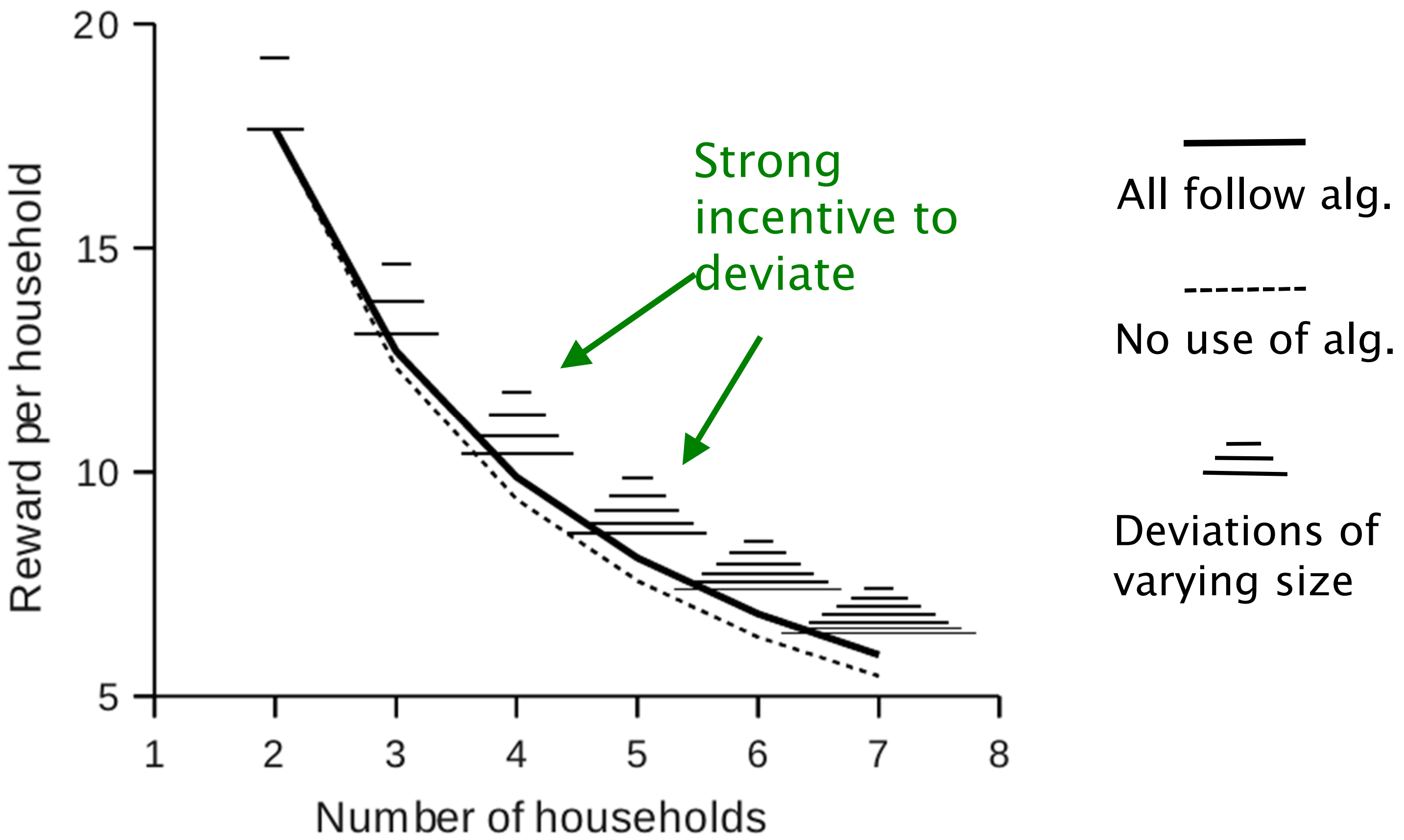
- Existing analysis
  - simulation-based,
  - assumes all clients are unselfish
- Our analysis
  - stochastic multi-player game
  - clients can cheat (and cooperate)
  - exposes protocol weakness
  - propose/verify simple fix



Verification of Competitive Stochastic Systems. Chen *et al*, *Formal Methods in System Design* 43(1): 61-92 (2013).

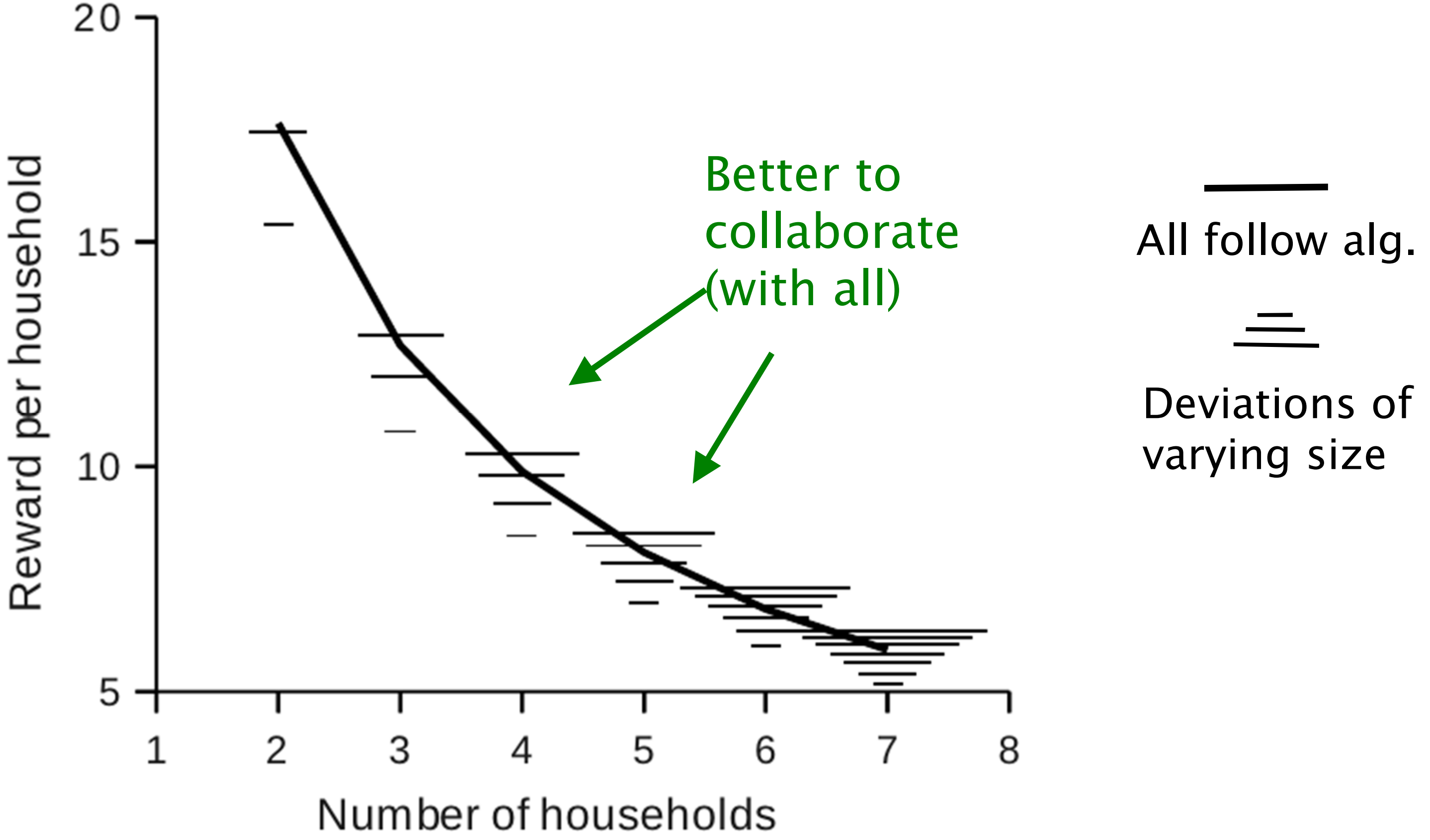
Results: Competitive behaviour

- The original algorithm does not discourage selfish behaviour…



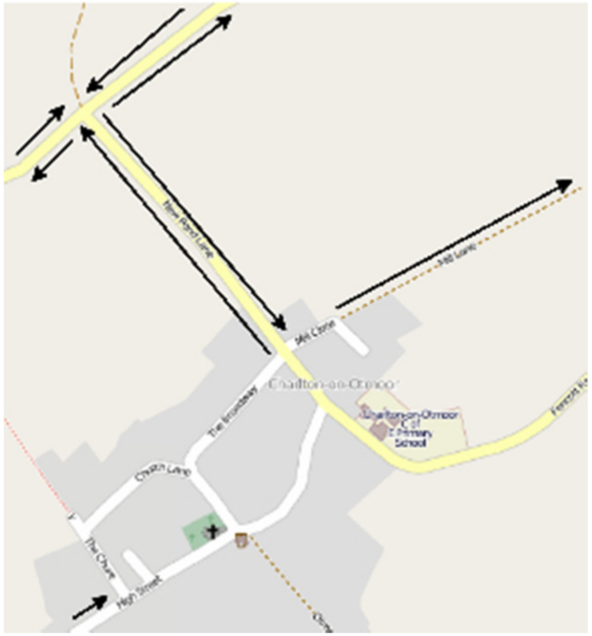
Results: Competitive behaviour

- Algorithm fix: simple punishment mechanism
  - distribution manager can cancel some tasks



Case study: Autonomous urban driving

- Inspired by DARPA challenge
  - represent map data as a stochastic game, with environment able to select hazards
  - express goals as conjunctions of probabilistic and reward properties
  - e.g. “maximise probability of avoiding hazards and minimise time to reach destination”



- Solution
  - synthesise a probabilistic strategy to achieve the multiobjective goal
  - enable the exploration of trade-offs between subgoals
- Applied to synthesise driving strategies for English villages
  - being developed as extension of PRISM

Synthesis for Multi-Objective Stochastic Games: An Application to Autonomous Urban Driving. Chen *et al*, In *Proc. QEST,* pages 322-337, IEEE CS Press. 2013.

Tool support: PRISM-games

- Prototype model checker for stochastic games
  - PRISM extended, adding games to the repertoire of models
  - property specification language based on ATL (Alternating Temporal Logic), incl. multiobjective
  - e.g. "coalition C has a strategy to ensure that the probability of success is above 0.9, regardless of strategies of other players"
  - verification and strategy synthesis
- Further case studies
  - collective decision making for sensor networks
  - user-centric networks
  - reputation-based protocols
- Available at:
  - http://www.prismmodelchecker.org/games/

Focus on...

Cooperation & competition

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control

Natural world

- Biosensing
- Molecular programming

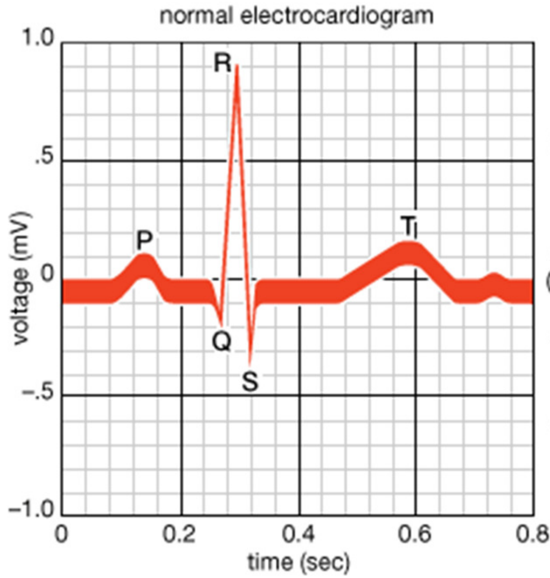
Monitoring physical processes

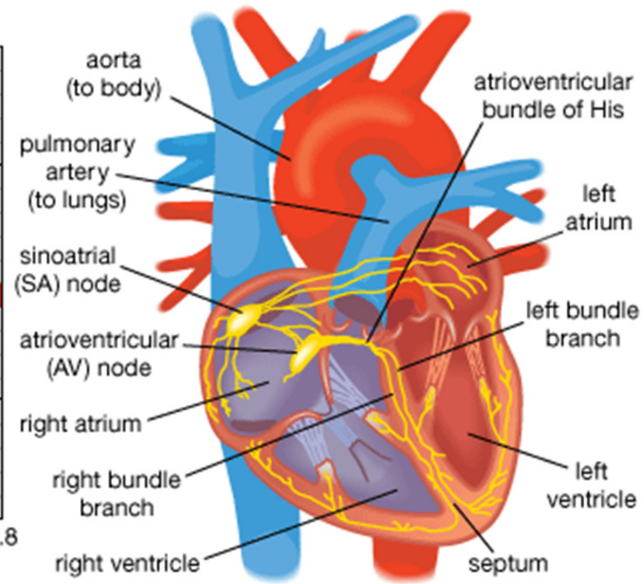
- Ubicomp systems monitor and control physical processes
  - electrical signal, velocity, distance, chemical concentration, …
  - often modelled by non-linear differential equations
  - necessary to extend models with continuous flows
- Many typical scenarios
  - e.g. smart energy meters, automotive control, closed loop medical devices
- Natural to adopt hybrid system models, which combine discrete mode switches and continuous variables
  - widely used in embedded systems, control engineering …
  - probabilistic extensions needed to model failure
- Research question: can we apply quantitative verification to establish correctness of implantable cardiac pacemakers? synthesise timing parameters?

Function of the heart

- Maintains blood circulation by contracting the atria and ventricles
  - spontaneously generates electrical signal (action potential)
  - conducted through cellular pathways into atrium, causing contraction of atria then ventricles
  - repeats, maintaining 60-100 beats per minute
  - a real-time system, and natural pacemaker

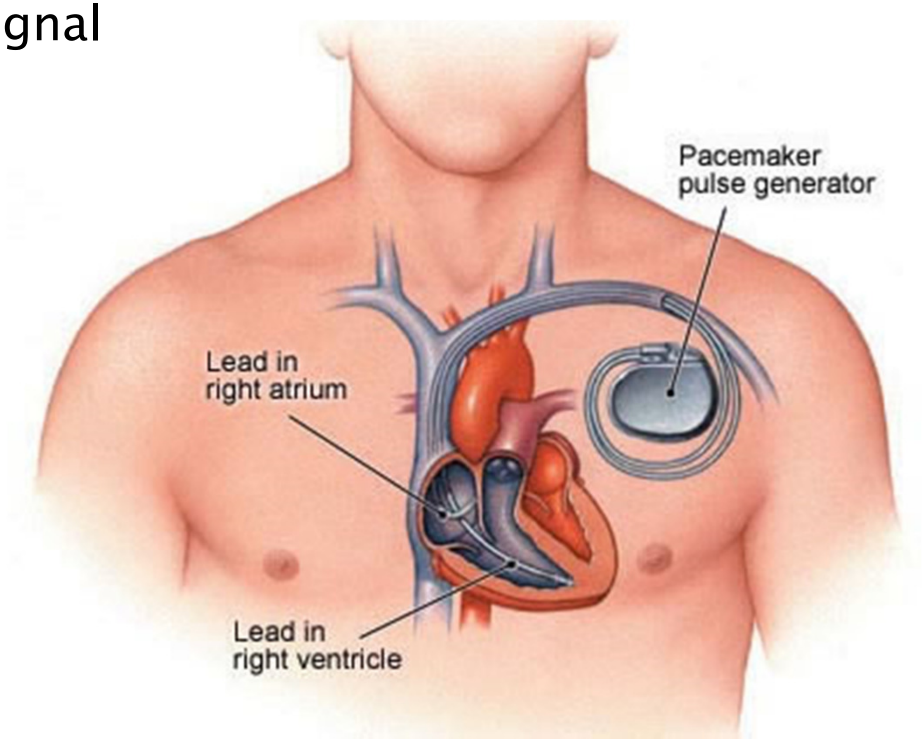
- Abnormalities in electrical conduction
  - missed/slow heart beat
  - can be corrected by implantable pacemakers





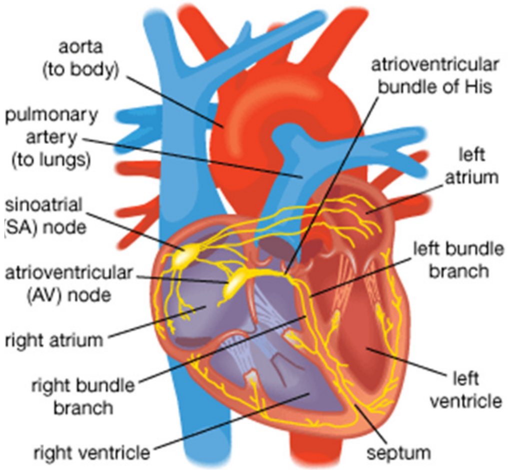
Implantable pacemaker

- How it works
  - reads electrical (action potential) signals through sensors placed in the right atrium and right ventricle
  - monitors the timing of heart beats and local electrical activity
  - generates artificial pacing signal as necessary
- Widely used, replaced every few years
- Core specification by Boston Scientific
- Basic pacemaker can be modelled as a network of timed automata [Ziang et al]

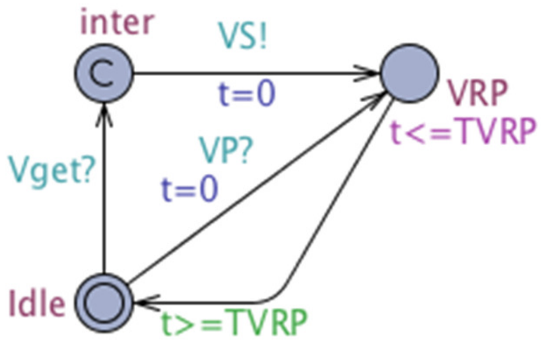


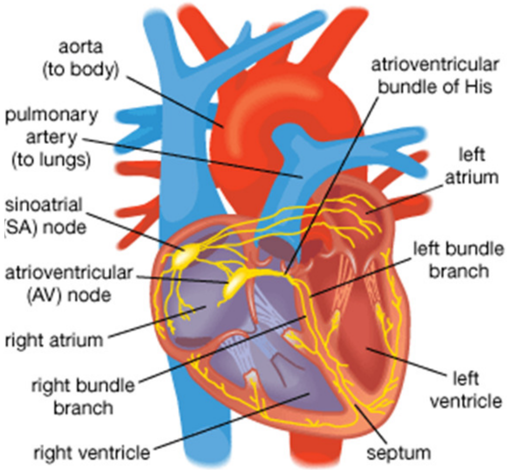
Quantitative verification for pacemakers

- Model the pacemaker and the heart, compose and verify



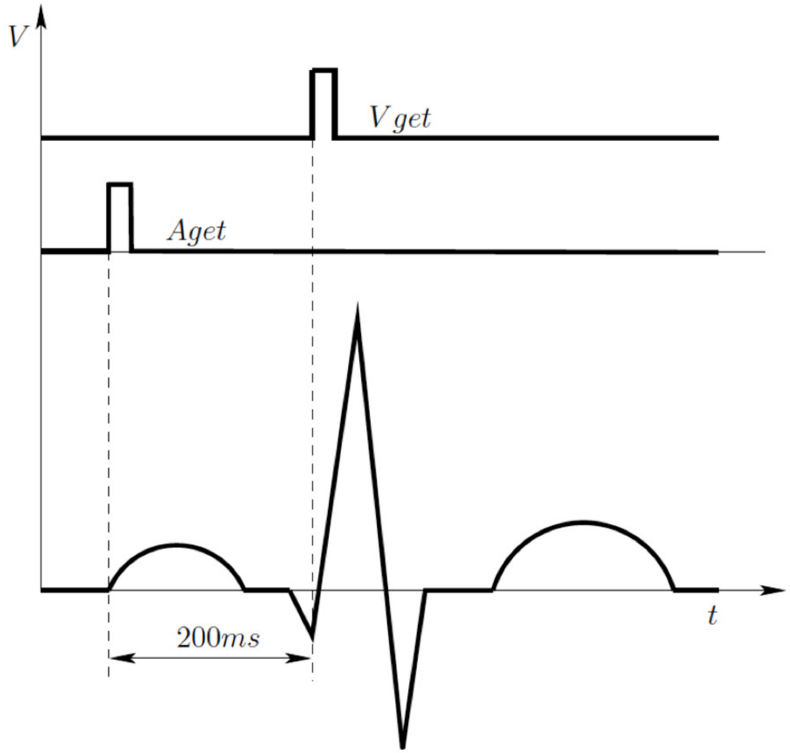
Quantitative verification for pacemakers

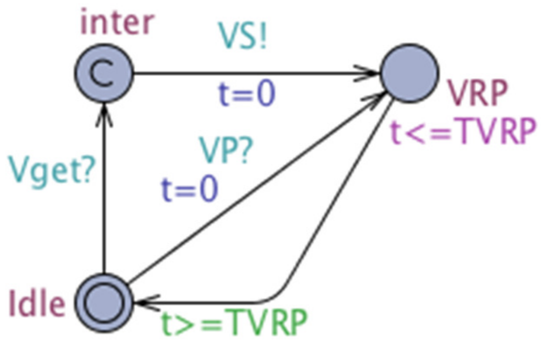





```
module VRP

s_vrp: [0..2] init 0;
t_vrp : clock;

// Invariants for clock t_vrp
  invariant
       (s_vrp = 2 => (t_vrp <= TVRP)) &
       (s_vrp = 1 => (t_vrp <= 0 ))
  endinvariant

[Vget] (s_vrp = 0) -> (s_vrp' = 1) & (t_vrp'=0);
[VP]   (s_vrp = 0) -> (s_vrp' = 2) & (t_vrp' = 0);
```

Quantitative verification for pacemakers



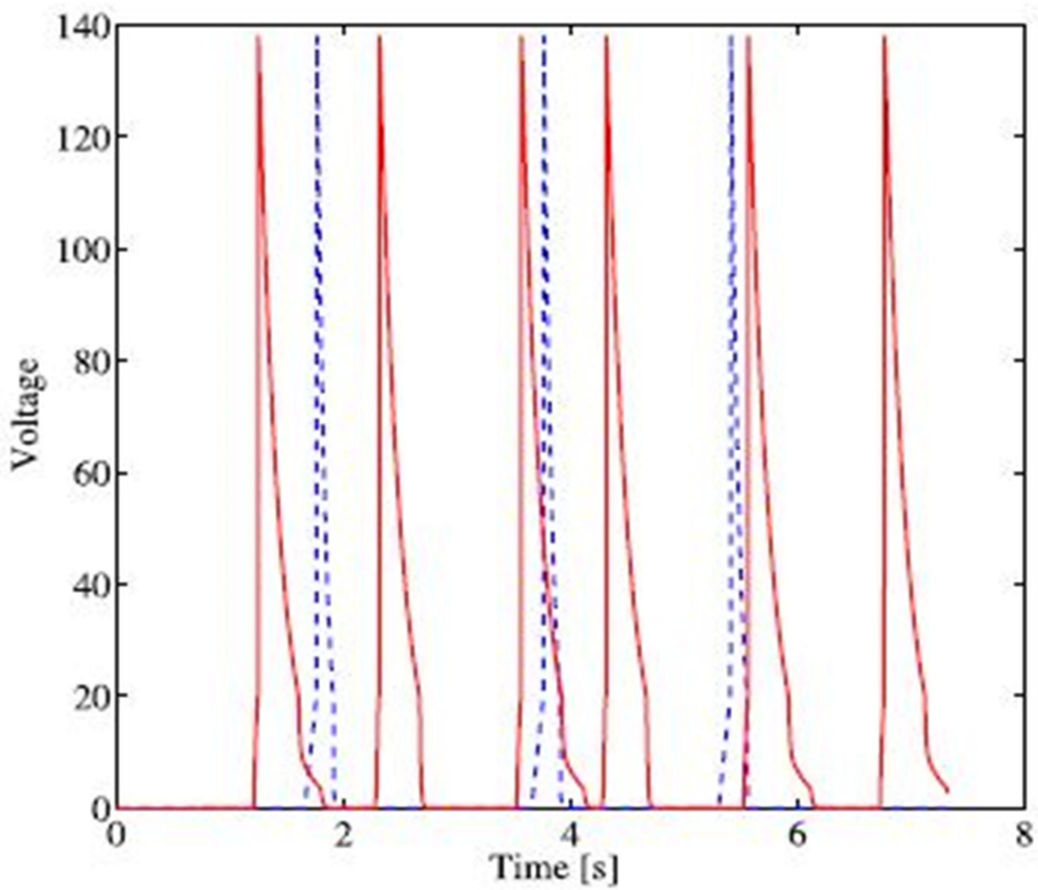



```
module VRP

s_vrp: [0..2] init 0;
t_vrp : clock;

// Invariants for clock t_vrp
  invariant
        (s_vrp = 2 => (t_vrp <= TVRP)) &
        (s_vrp = 1 => (t_vrp <= 0 ))
  endinvariant

[Vget] (s_vrp = 0) -> (s_vrp' = 1) & (t_vrp'=0);
[VP]  (s_vrp = 0) -> (s_vrp' = 2) & (t_vrp' = 0);
```

Quantitative verification for pacemakers

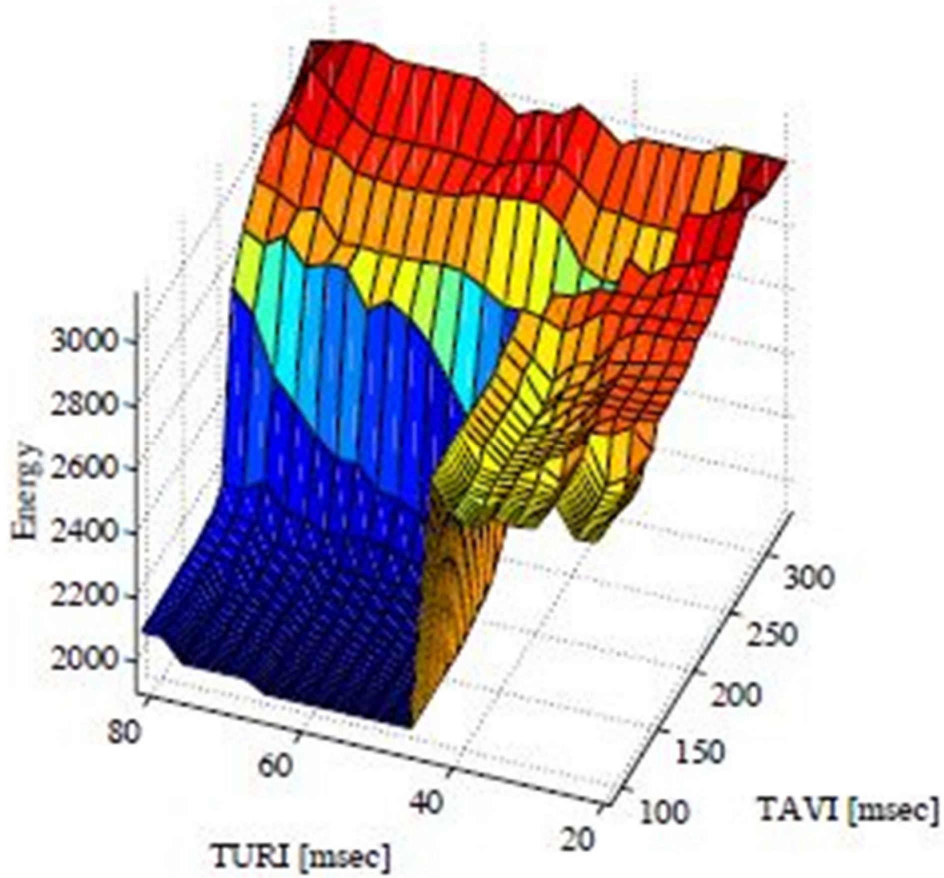
- Given a model of the pacemaker and a heart model, compose and verify against extended MTL (Metric Temporal Logic) properties (syntax omitted):
  - basic safety: “for any 1 minute window, the number of heart beats lies in the interval [60,100]”
  - energy: “for a given time point T, the energy consumed is less than the given energy level V”
- But models are multi-component, hybrid, nonlinear, and can contain stochasticity!
- Methodologies
  - rely on simulation and parameterise by simulation step
  - employ approximate verification based on finitely many simulation runs: estimate probability of satisfying property from Chernoff bound, for some confidence interval
  - overapproximate reach sets using annotations

Correction of Bradycardia



Blue lines original (slow) heart beat, red are induced (correcting)

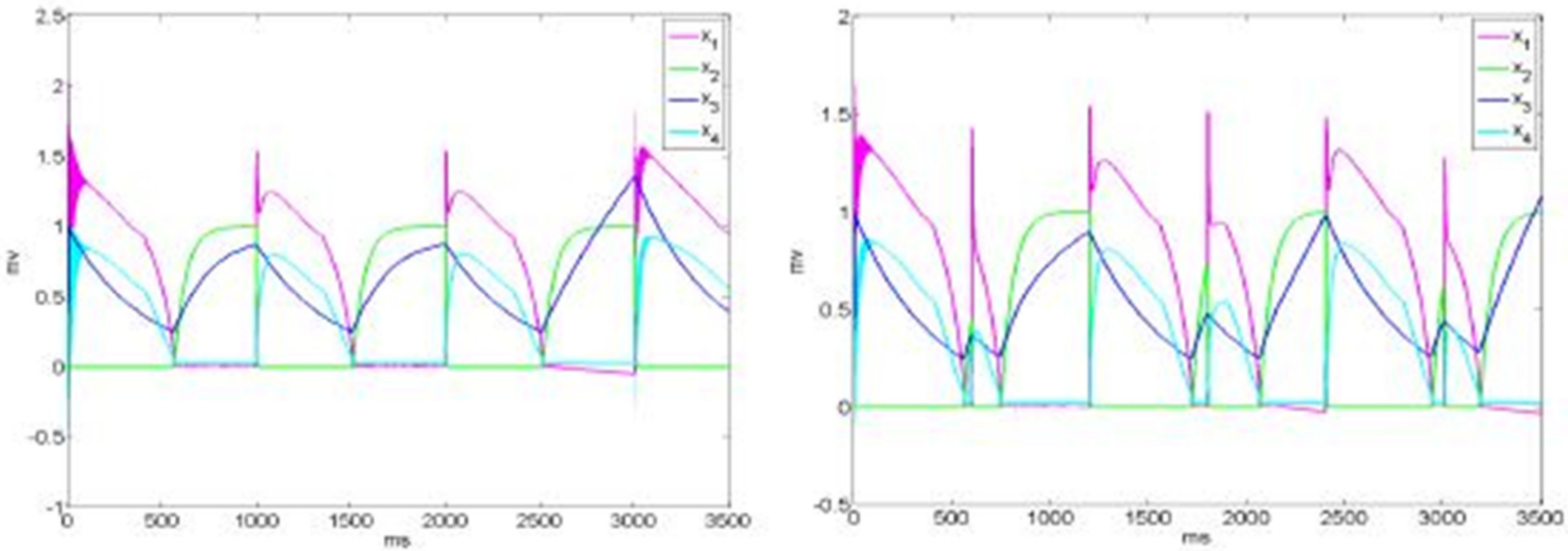
Energy consumption



Battery charge in 1 min under Bradycardia, varying timing parameters.

Quantitative Verification of Implantable Cardiac Pacemakers over Hybrid Heart Models. Chen *et al*, *Information and Computation*, 2014

Alternans in the heart



(a) (b)

We plot the reach set from a set of initial states with pacing rate of 1000 msec and observe that the AP durations do not change (a), whereas at a pacing rate of 600 msec (b) the AP durations alternate.

Invariant Verification of Nonlinear Hybrid Automata Networks of Cardiac Cells. Huang *et al*
In *CAV*, volume 8559 of LNCS, pages 373-390, Springer, 2014.

Tool support: MATLAB Simulink

- Develop a model-based framework
  - models are networks of timed or hybrid I/O automata, realised in Matlab Simulink
    - quantitative: energy usage, probabilistic switching
    - patient-specific parameterisation
- Functionality
  - plug-and-play composition of heart and pacemaker models
  - (approximate) quantitative verification against variants of MTL
    - to ensure property is satisfied
  - parametric analysis
    - for in silico evaluation, to reduce need for testing on patients
  - automated synthesis of optimal timing parameters
    - to determine delays between paces so that energy usage is optimised for a given patient
- See http://www.veriware.org/pacemaker.php

Focus on...

Cooperation & competition

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control

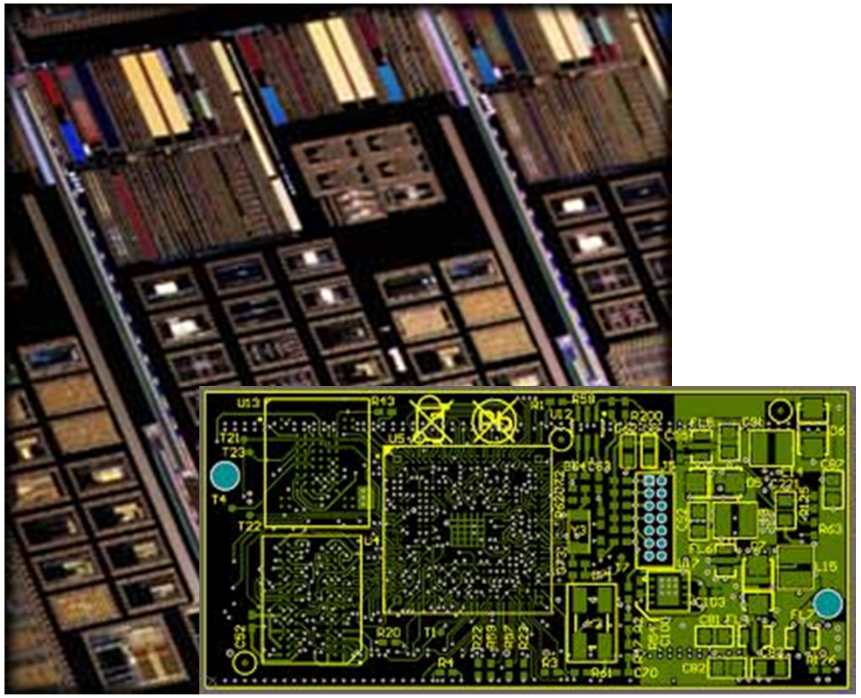
Natural world

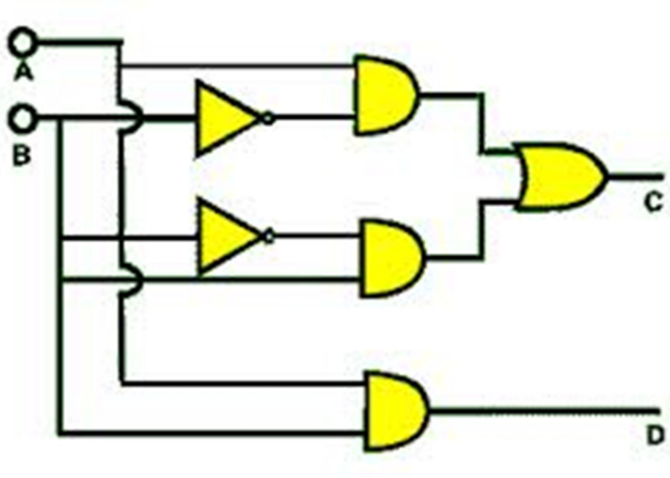
- Biosensing
- Molecular programming

Interacting with the natural world

- Ubicomp systems need to sense and control biological processes
  - programmable identification of substance, targeted delivery, movement
  - directly at the molecular level
- Many typical scenarios
  - e.g. smart therapeutics, drug delivery directly into the blood stream, implantable continuous monitoring devices
- Natural to adopt the molecular programming approach
  - here, focus on DNA computation, which aims to build computing devices using DNA molecules
  - shared techniques and tools with synthetic biology
- Research question: can we apply (quantitative) verification to DNA programs?

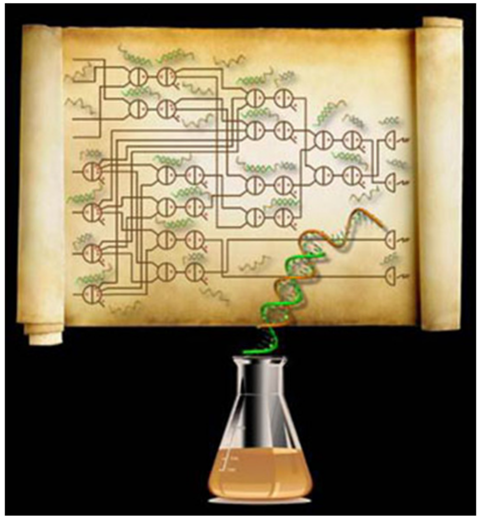
Digital circuits





- Logic gates realised in silicon
- 0s and 1s are represented as low and high voltage
- Hardware verification indispensable as design methodology

DNA circuits

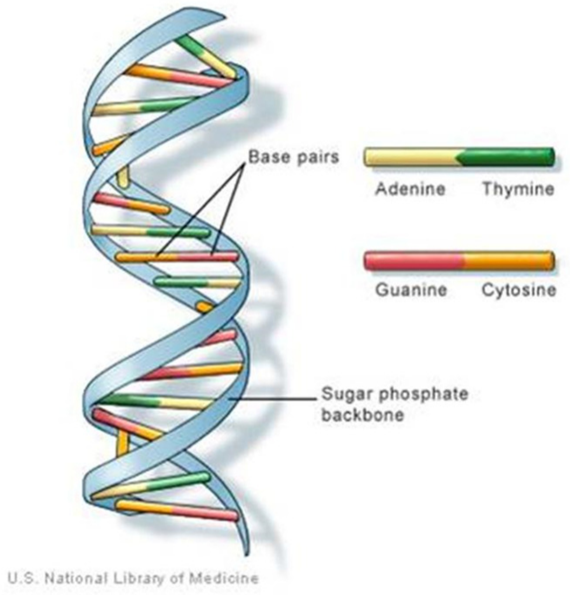


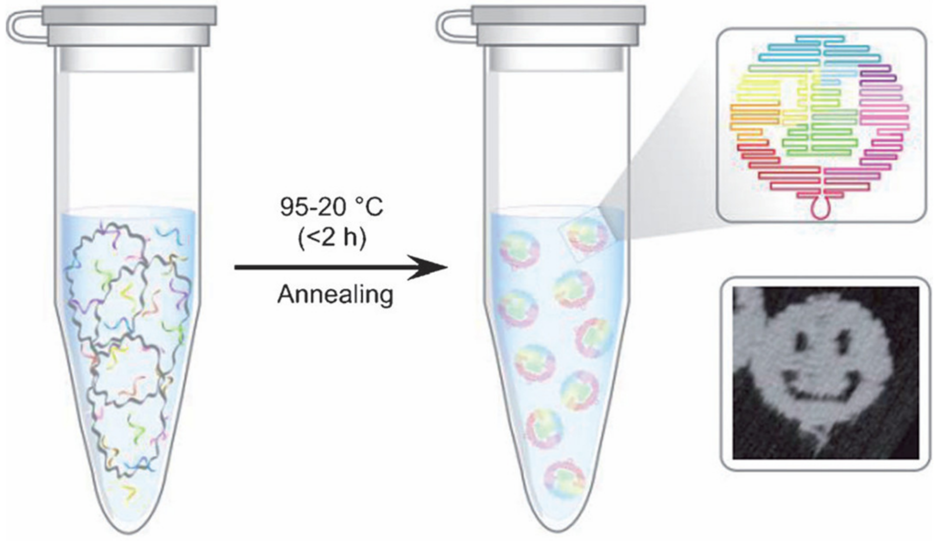
[Qian, Winfree, *Science* 2012]

Pop quiz, hotshot: what's the square root of 13?
Science Photo Library/Alamy

- "Computing with soup" (The Economist 2012)
- DNA strands are inputs and outputs
- Circuit of 130 strands computes square root of 4 bit number, rounded down
- 10 hours, but it's a first…

DNA structures





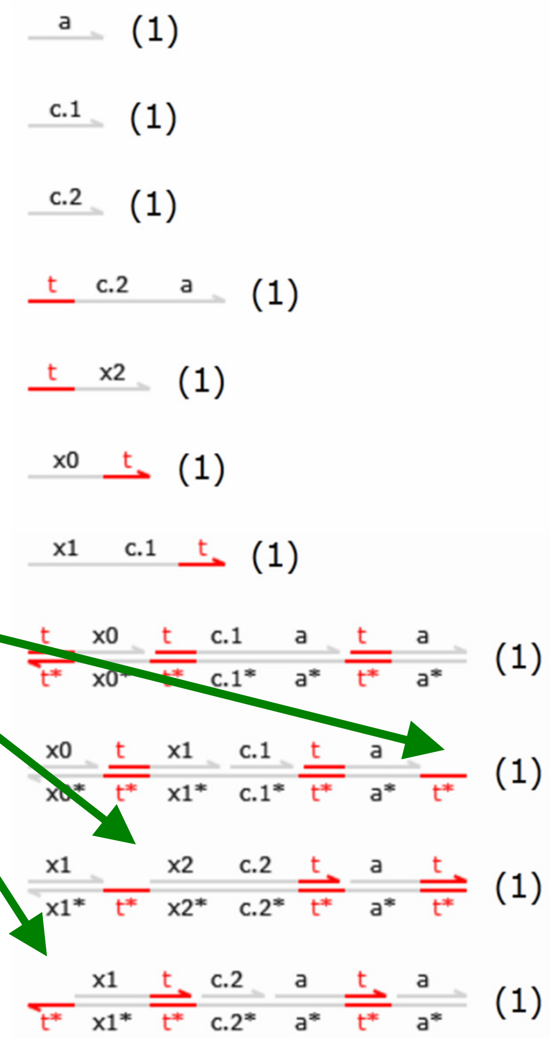
DNA origami

- DNA origami [Rothemund, Nature 2006]
  - DNA can self-assemble into structures – “molecular IKEA?”
  - Programmable self-assembly (can form tiles, nanotubes, boxes that can open, etc)
  - Simple manufacturing process (heating and cooling), not yet well understood

Logic gates made from DNA

Case study: DNA circuits

- DNA circuits: seemingly simple
- Design flaws possible!
- PRISM identifies a 5-step trace to the "bad" deadlock state
  - previously found manually [Cardelli'10]
  - detection now fully automated
- Bug is easily fixed
  - (and verified)



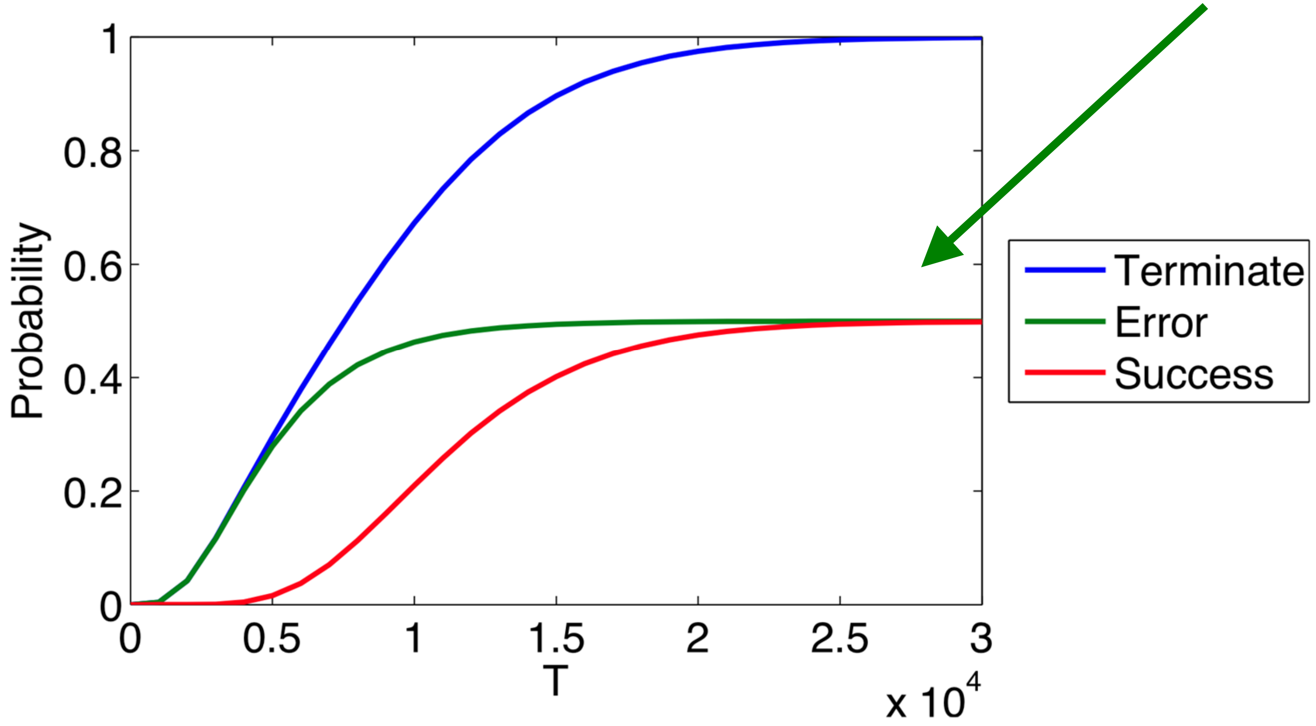
Counterexample:
(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,1,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,1,1,0,0,0,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,0,1,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0)

Design and Analysis of DNA Strand Displacement Devices using Probabilistic Model Checking, Lakin *et al*, Journal of the Royal Society Interface, 9(72), 1470-1485, 2012

Transducers: Quantitative properties

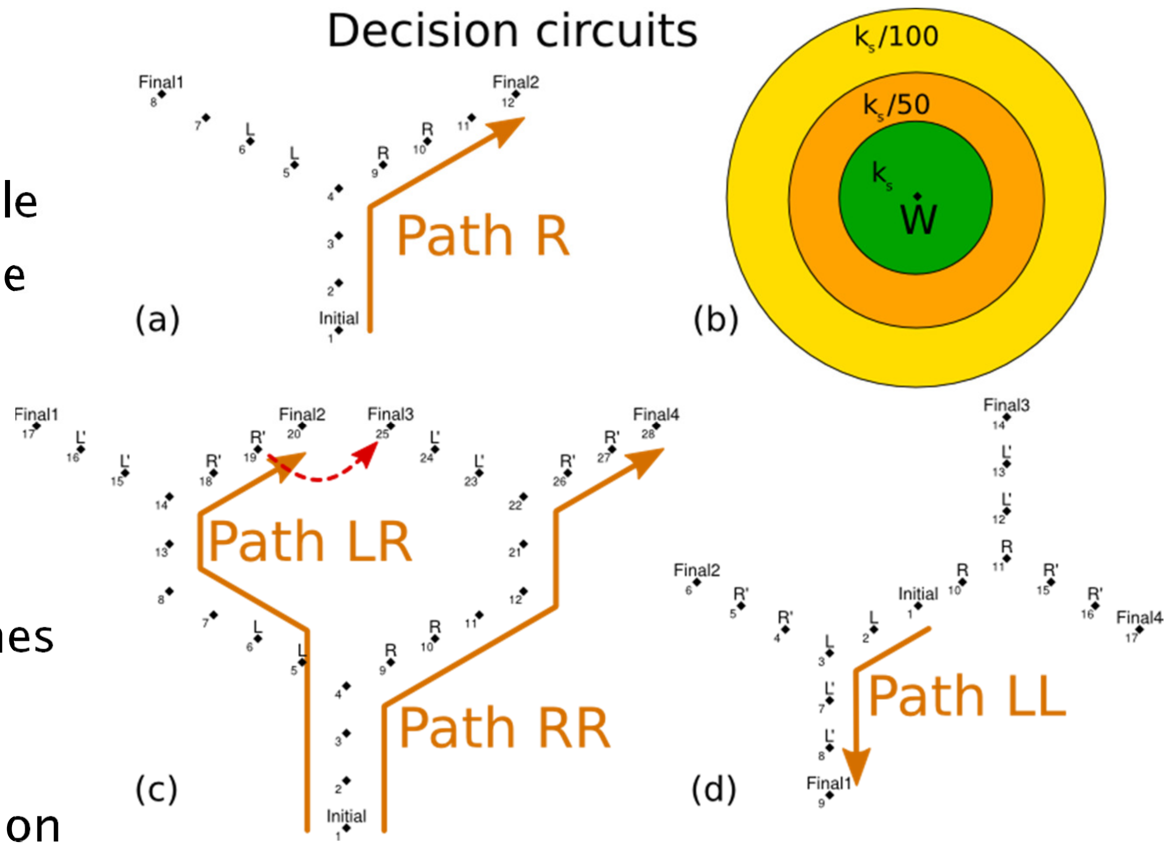
- We can also use PRISM to study the kinetics of the pair of (faulty) transducers:
  - $P_{=?}$ [$F^{[T,T]}$ "deadlock"]
  - $P_{=?}$ [$F^{[T,T]}$ "deadlock" & !"all_done"]
  - $P_{=?}$ [$F^{[T,T]}$ "deadlock" & "all_done"]

success/error
equally likely



Case study: DNA walkers

- How it works…
  - tracks laid out on DNA origami tile
  - can make molecule ‘walk’ by attaching/detaching from anchor
  - starts at ‘initial’, detect when reaches ‘final’
  - can control ‘left’/’right’ decision



- Biosensors for diagnosis, targeted drug delivery
  - safety/reliability paramount: devise a model, analyse with PRISM

DNA walker circuits: Computational potential, design, and verification, Dannenberg *et al*, Natural Computing, To appear, 2014

Tool support: DSD & PRISM

- Developed a framework incorporating DSD and PRISM
  - DSD designs automatically translated to PRISM via SBML
- Model checking as for molecular signalling networks
  - reduction to CTMC model
  - reuse existing PRISM algorithms
- Achievements
  - first ever (quantitative) verification of a DNA circuit
  - demonstrated bugs can be found automatically
  - but scalability major challenge
- Further case studies
  - approximated majority, molecular walkers
- Available now:

  http://www.veriware.org/dna.php

Summing up…

- An exciting future ahead!
  - Smartphones, smart devices, smart homes
- Brief overview of progress in quantitative verification
  - demonstrating first successes and usefulness of quantitative verification and synthesis methodology
  - and resulting in new techniques and tools
- Many technological and scientifi challenges remain
  - huge models!
  - compositional methods
  - integration of discrete, continuous and stochastic dynamics
  - scalability of quantitative verification and synthesis
  - accuracy of approximate verification
  - efficiency of parameter synthesis
  - model synthesis from quantitative requirements

Acknowledgements

- My group and collaborators in this work
- Collaborators who contributed to theoretical and practical PRISM development
- External users of, and contributors to, PRISM
- Project funding
  - ERC, EPSRC, Microsoft Research Cambridge
  - Oxford Martin School, Institute for the Future of Computing

- See also
  - VERIWARE www.veriware.org
  - PRISM www.prismmodelchecker.org