Sensing everywhere:

on quantitative verification for ubiquitous computing

Marta Kwiatkowska

University of Oxford

Milner Lecture, University of Edinburgh, 25 Sep 2012

Where are computers?

Once upon a time, back in the 1980s…

Smartphones, tablets…

Sensor apps

GPS/GPRS tracking
Accelerometer
Air quality

Access to services
Personalised monitoring

House appliances, networked…

Fridge that Tweets!

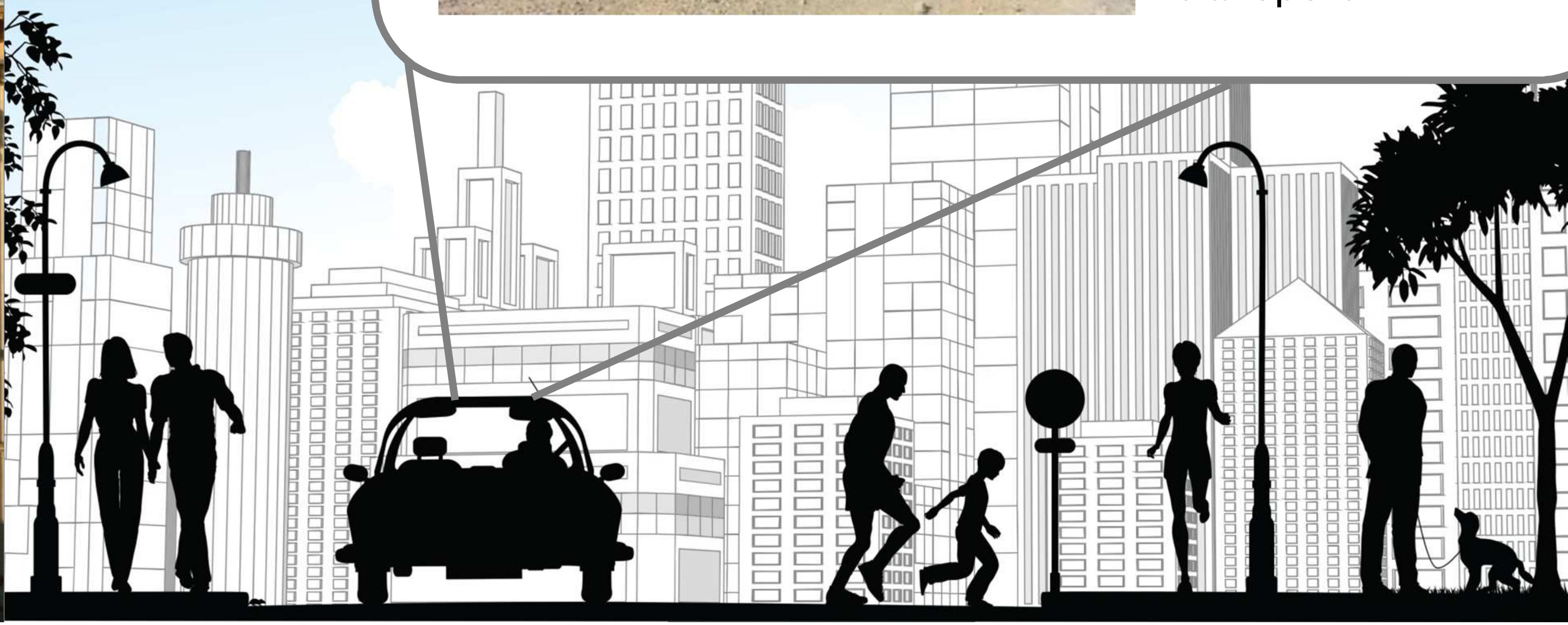
Home network
Internet-enabled
Remote control
Energy management

Intelligent transport…

Look, no hands!

Self-parking cars
Traffic jam assistance

Personalised transport



Medical devices…

Wearable or implantable health monitoring

Heart rate
Breathing
Movement
Glucose…

Ubiquitous computing

- Computing without computers
- (also known as Pervasive Computing or Internet of Things
  - enabled by wireless technology and cloud computing)
- 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…

Perspectives on ubiquitous computing

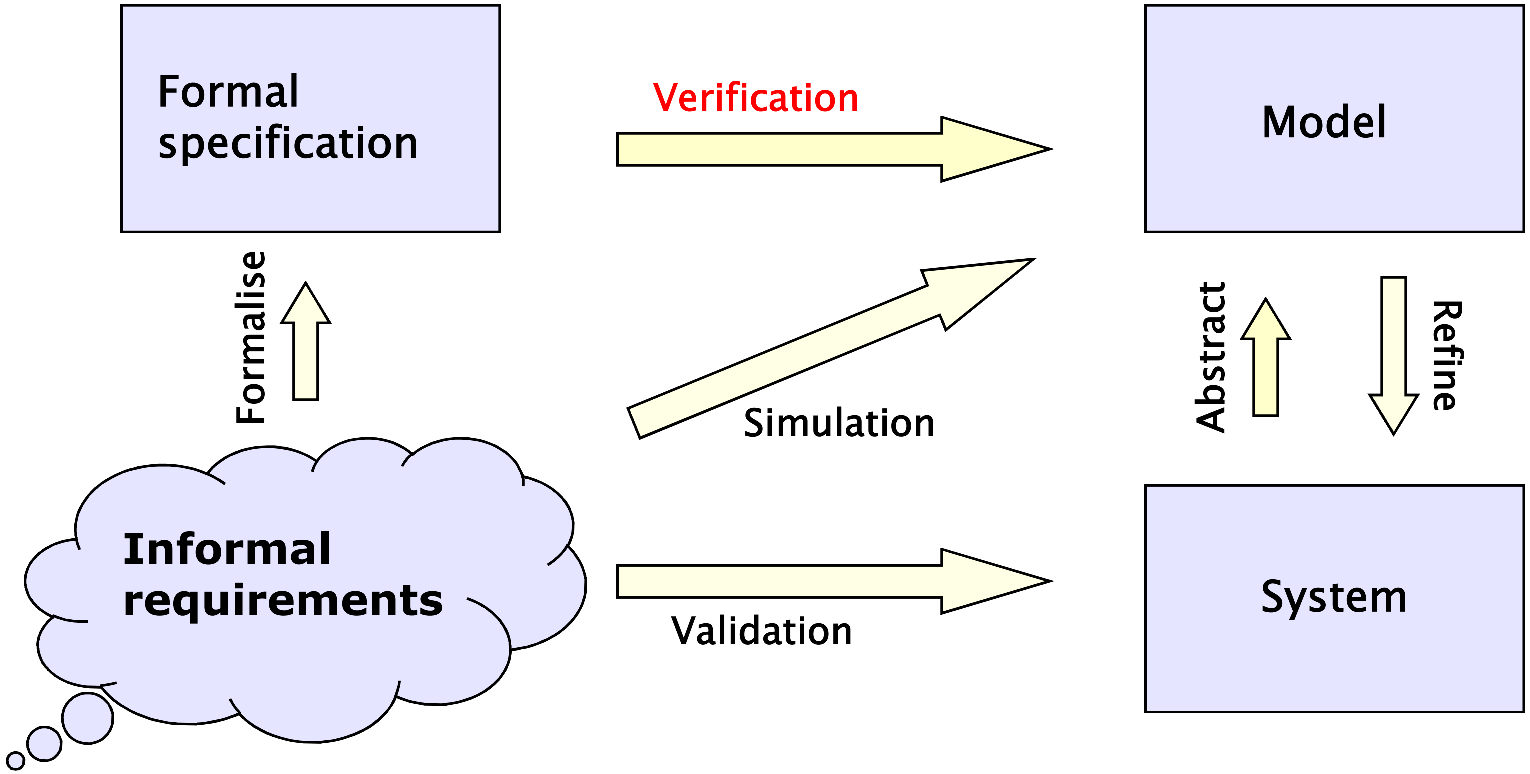
- Technological: calm technology [Weiser 1993]
  - "The most profound technologies are those that disappear. They weave themselves into everyday life until they are indistinguishable from it."
- Usability: 'everyware' [Greenfield 2008]
  - Hardware/software evolved into 'everyware': household appliances 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

Software quality assurance

- Software is a critical component
  - embedded software failure costly and life endangering
- Need quality assurance methodologies
  - model-based development
  - rigorous software engineering
  - software product lines
- Use formal techniques to produce guarantees for:
  - safety, reliability, performance, resource usage, trust, …
  - (safety) "probability of failure to raise alarm is tolerably low"
  - (reliability) "the smartphone will never execute the financial transaction twice"
- Focus on automated, tool-supported methodologies
  - automated verification via model checking
  - quantitative verification

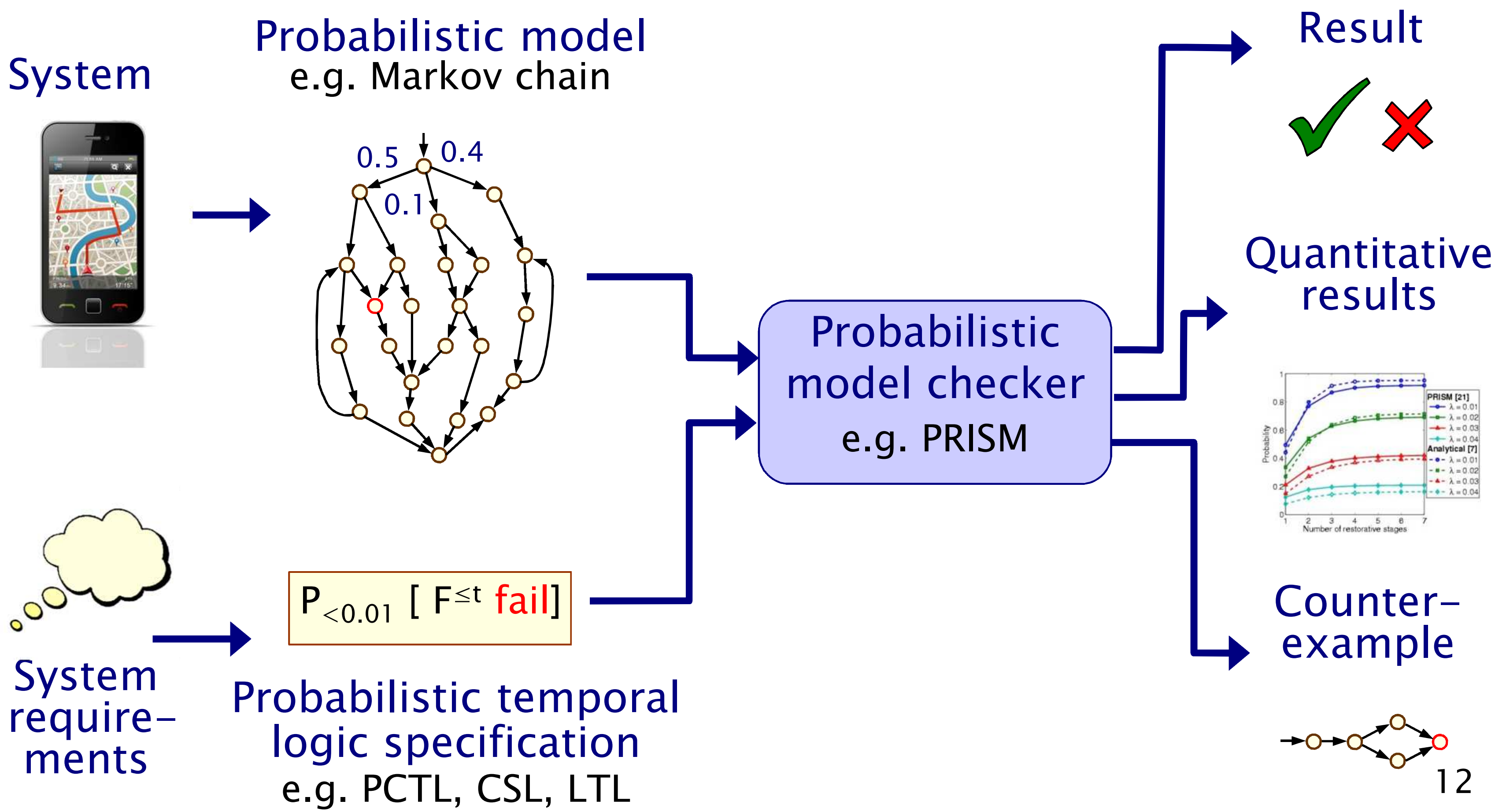
Rigorous software engineering

- Verification and validation
  - Derive model, or extract from software artefacts
  - Verify correctness, validate if fit for purpose



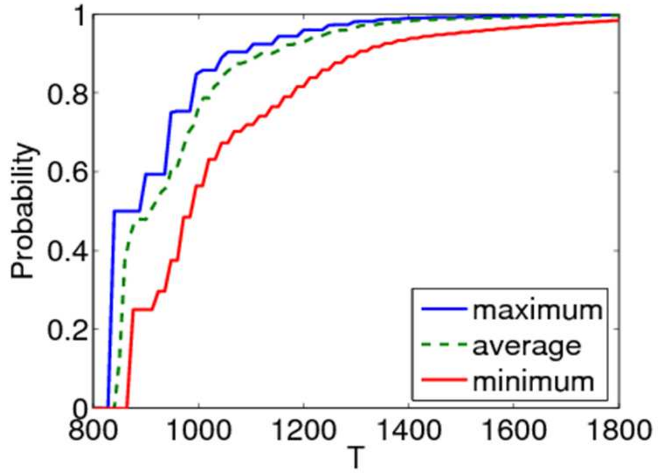
Quantitative (probabilistic) verification

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"

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

Quantitative probabilistic verification

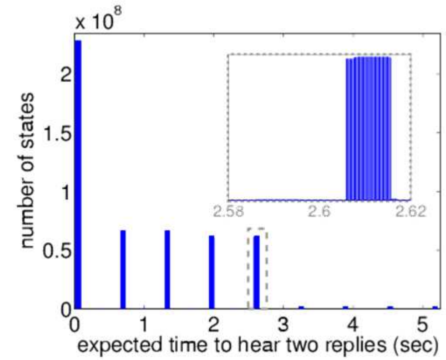
- What's involved
  - specifying, extracting and building of quantitative models
  - graph-based analysis: reachability + qualitative verification
  - numerical solution, e.g. linear equations/linear programming
  - typically computationally more expensive than the non-quantitative case
- The state of the art
  - fast/efficient techniques for a range of probabilistic models
  - feasible for models of up to 10^7 states (10^{10} with symbolic)
  - extension to probabilistic real-time systems
  - abstraction refinement (CEGAR) methods
  - probabilistic counterexample generation
  - assume-guarantee compositional verification
  - tool support exists and is widely used, e.g. PRISM, MRMC

Tool support: PRISM

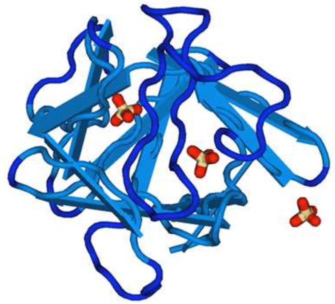
- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
- Support for:
  - models: DTMCs, CTMCs, MDPs, PTAs, …
  - properties: PCTL, CSL, LTL, PCTL*, costs/rewards, …
- Features:
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
- Many import/export options, tool connections
  - in: (Bio)PEPA, stochastic π-calculus, DSD, SBML, Petri nets, …
  - out: Matlab, MRMC, INFAMY, PARAM, …
- See: http://www.prismmodelchecker.org/

Quantitative verification in action

- Bluetooth device discovery protocol
  - frequency hopping, randomised delays
  - low-level model in PRISM, based on detailed Bluetooth reference documentation
  - numerical solution of 32 Markov chains, each approximately 3 billion states
  - identified worst-case time to hear one message



- Fibroblast Growth Factor (FGF) pathway
  - complex biological cell signalling pathway, key roles e.g. in healing, not yet fully understood
  - model checking (PRISM) & simulation (stochastic π-calculus), in collaboration with Biosciences at Birmingham
  - "in-silico" experiments: systematic removal of components
  - behavioural predictions later validated by lab experiments



The challenge of ubiquitous computing

- Quantitative verification is not powerful enough!

- Necessary to model communities and cooperation
  - add self-interest and ability to form coalitions
- Need to monitor and control physical processes
  - extend models with continuous flows
- In future important to interface to biological systems
  - 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

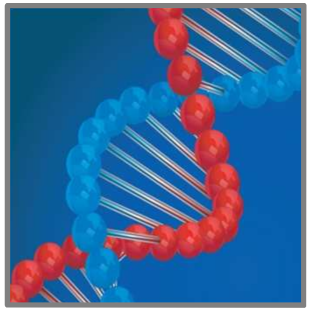
Focus on…

Cooperation

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control



Natural world

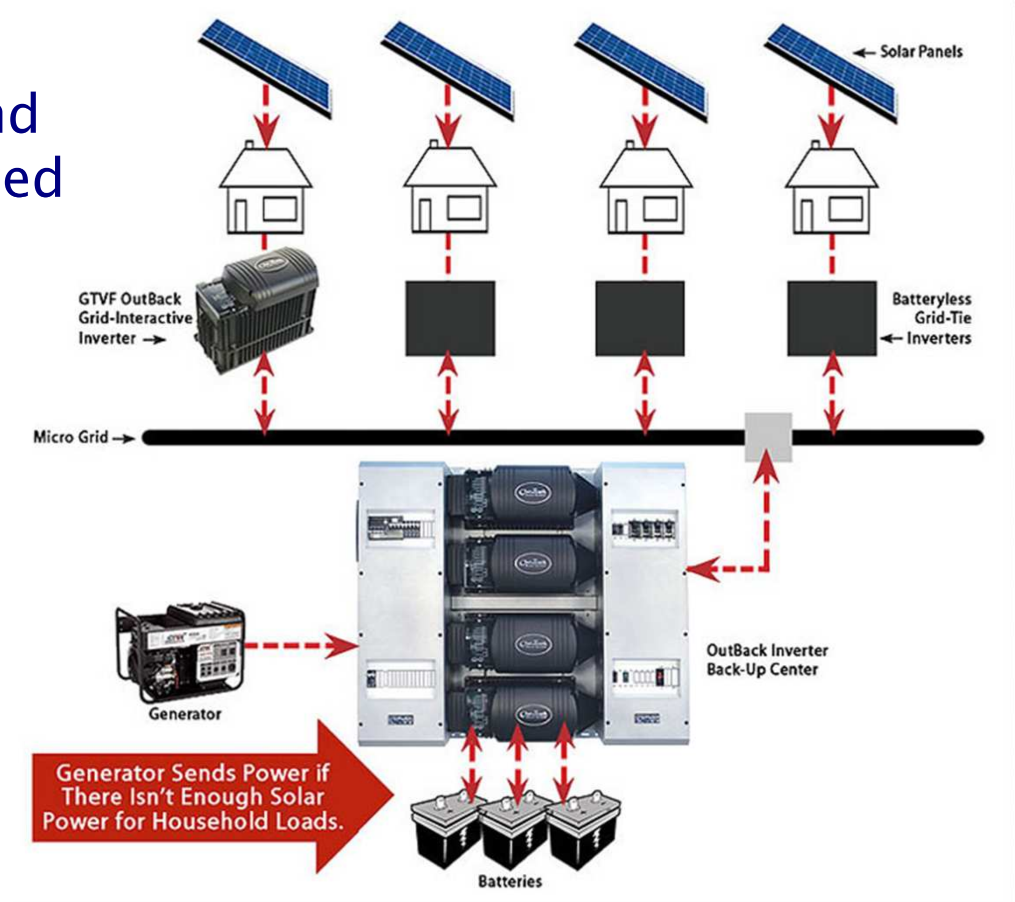
- Biosensing
- Molecular programming

Modelling cooperation

- 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, automated verification
- Research question: can we automatically verify cooperative and competitive behaviour?

Energy management for the future?

- Microgrid: proposed model for future energy markets
  - localised energy management
- Neighbourhoods use and store electricity generated from local sources
  - wind, solar, …
- Needs: demand-side management
  - active management of demand by users
  - to avoid peaks
  - autonomous operation



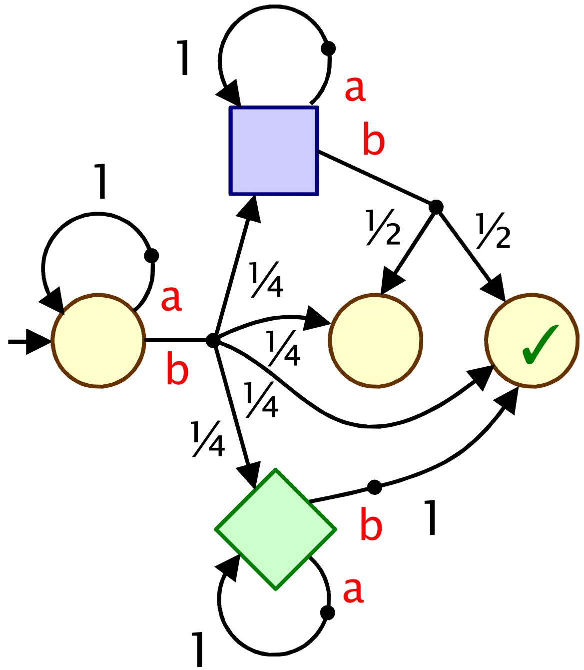
Microgrid demand-side management

- New protocols proposed, here consider demand-side management algorithm of [Hildmann/Saffre'11]
  - N households, connected to energy distribution supplier
  - households submit tasks requiring power
  - task submission probabilistic, realistic daily demand curve
  - aim to maximise value V per household, while minimising total energy cost
- Simple probabilistic algorithm:
  - upon task submission, if cost is below an agreed limit, execute it, otherwise only execute with probability P_{start}
- Analysis of [Hildmann/Saffre'11]
  - simulation-based analysis shows reduction in peak demand and total energy cost reduced, with good expected value V
  - (providing all households stick to algorithm)

Stochastic multi-player games

- Stochastic multi-player game (SMGs)
  - probability + nondeterminism + multiple players

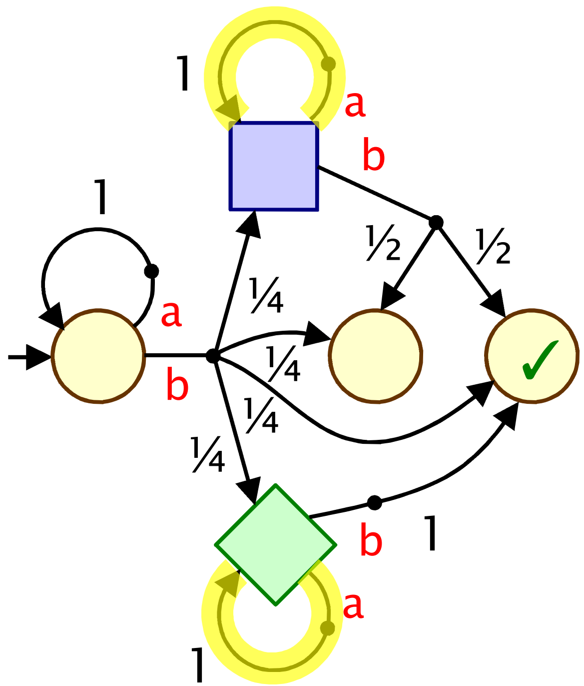
- A (turn-based) SMG is a tuple $(\Pi, S, \langle S_i \rangle_{i \in \Pi}, A, \Delta, L)$:
  - Π is a set of n players
  - S is a (finite) set of states
  - $\langle S_i \rangle_{i \in \Pi}$ is a partition of S
  - A is a set of action labels
  - $\Delta : S \times A \rightarrow Dist(S)$ is a (partial) transition probability function
  - $L : S \rightarrow 2^{AP}$ is a labelling with atomic propositions from AP



- NB
  - players ◇ □ can prevent player ○ from reaching ✓ with probability ≥⅓

Stochastic multi-player games

- Stochastic multi-player game (SMGs)
  - probability + nondeterminism + multiple players
- A (turn-based) SMG is a tuple $(\Pi, S, \langle S_i \rangle_{i \in \Pi}, A, \Delta, L)$:
  - Π is a set of n players
  - S is a (finite) set of states
  - $\langle S_i \rangle_{i \in \Pi}$ is a partition of S
  - A is a set of action labels
  - $\Delta : S \times A \rightarrow Dist(S)$ is a (partial) transition probability function
  - $L : S \rightarrow 2^{AP}$ is a labelling with atomic propositions from AP



- NB
  - players ◇ □ can prevent player ○ from reaching ✓ with probability $\geq$⅓

Property specification: rPATL

- New temporal logic rPATL
  - probabilistic & reward extension of alternating temporal logic
  - CTL, extended with:
    - coalition operator ⟨⟨C⟩⟩ of ATL
    - probabilistic and reward operators P, R of PCTL/PRISM

- Examples (simplifying the reachability operator F)
  - $\langle\langle\{1,2\}\rangle\rangle\ P_{<0.01}\ [\ F^{\leq 10}\ \text{“error”}\ ]$
  - “players 1 and 2 have a strategy to ensure that the probability of an error occurring within 10 steps is less than 0.1, regardless of the strategies of other players”
  - $\langle\langle C\rangle\rangle\ R_{=?}\ [\ F\ \text{“stable”}\ ]$
  - “the minimum expected energy that coalition C can conserve to reach a stable state, no matter what the other players do”

rPATL semantics

- SMGs have multiple (>2) players
- Fix coalition C for each analysis (assuming independence of strategies)
- Model check by reduction to a stochastic 2-player game

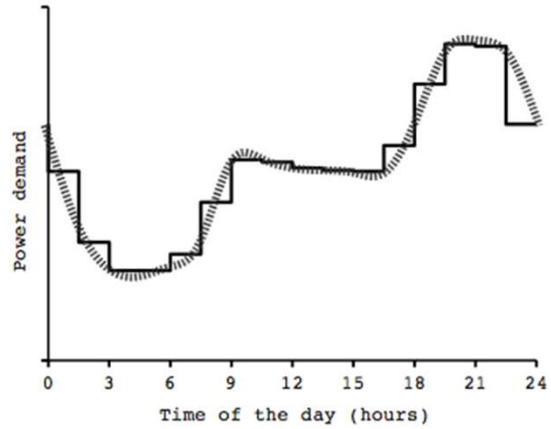
- Coalition game G_C for SMG G and coalition $C \subseteq \Pi$
  - 2-player SMG where C and $\Pi \backslash C$ collapse to players 1 and 2

- $\langle\langle C \rangle\rangle P_{<q}$[F "end"] is true in state s of G iff:
  - in coalition game G_C:
  - $\exists$ strategy $\sigma_1 \in \Sigma_1$ of player 1 such that $\forall$ strategies $\sigma_2 \in \Sigma_2$ of player 2 the probability of reaching 'end' is less than q

- Semantics for R operator defined similarly…

Microgrid demand-side management

- The model
  - SMG with N players (one per household)
  - analyse 3-day period, using piecewise approximation of daily demand curve
  - add rewards for value V



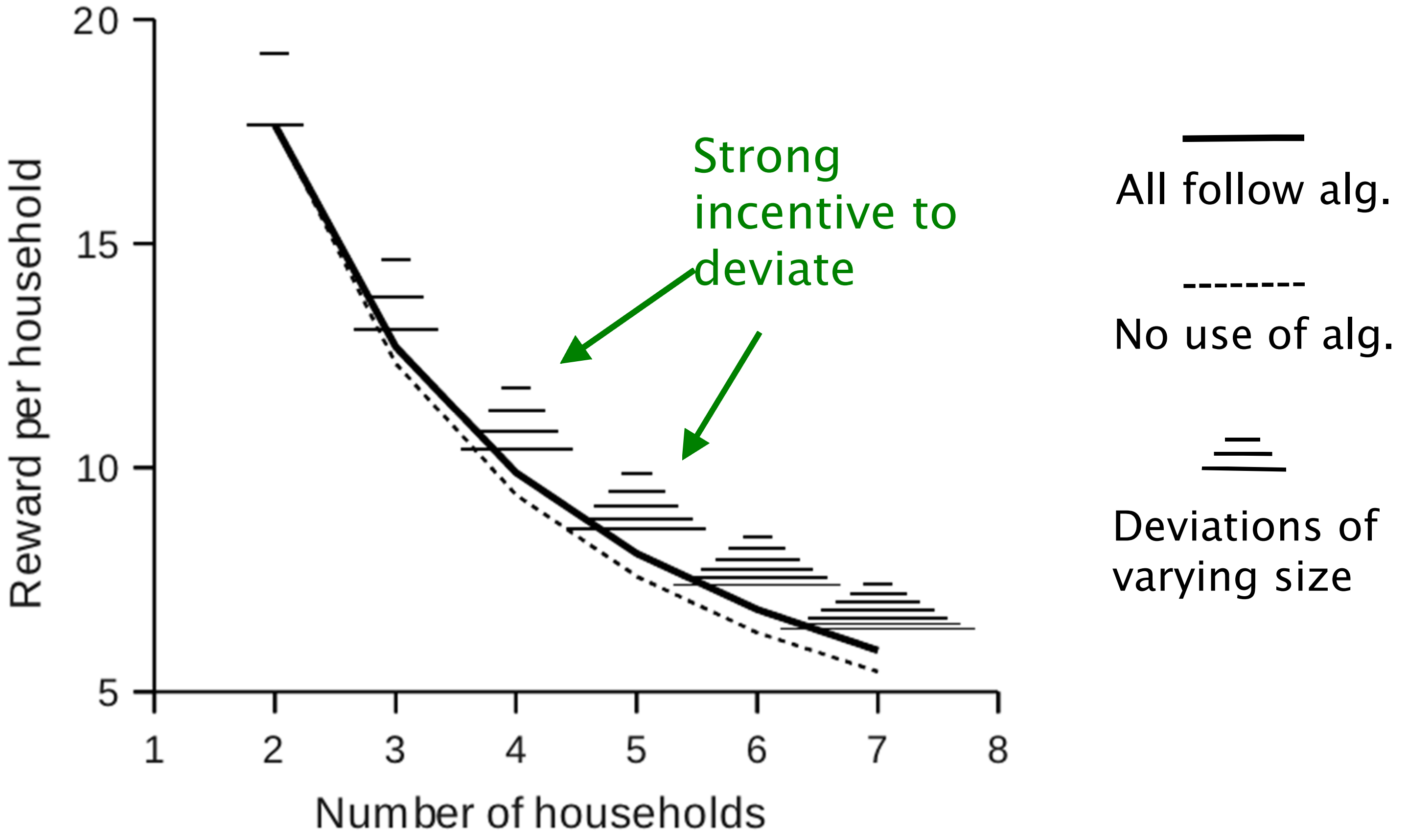
- Built/analysed models
  - for N=2,...,7 households

N	States	Transitions
5	743,904	2,145,120
6	2,384,369	7,260,756
7	6,241,312	19,678,246

- Step 1: assume all households follow algorithm of [HS'11] (MDP)
  - obtain optimal value for P_{start}
- Step 2: introduce competitive behaviour (SMG)
  - allow coalition C of households to deviate from algorithm

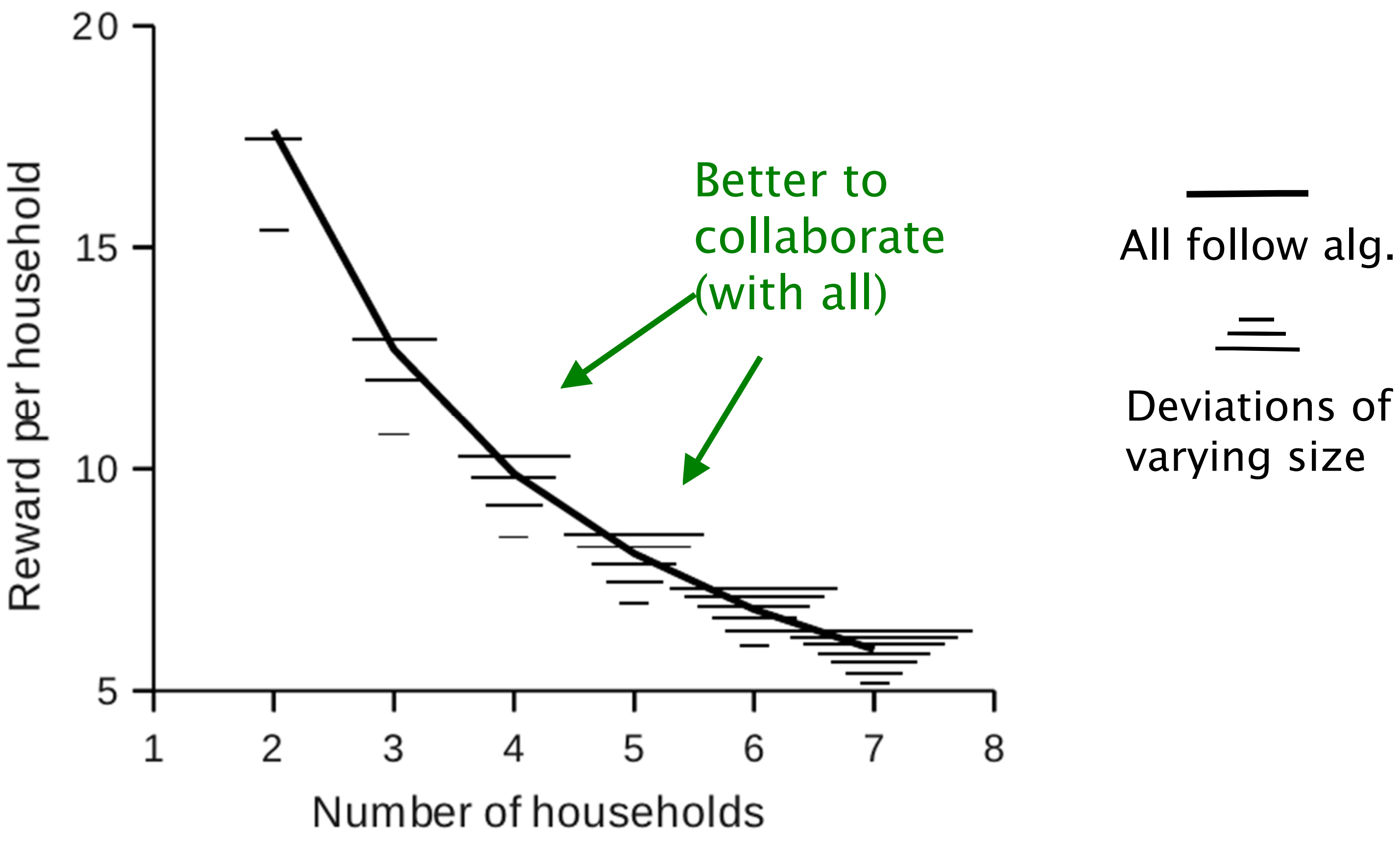
Results: Competitive behaviour

- The original algorithm does not discourage selfish behaviour…



Results: Competitive behaviour

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



Tool support: PRISM-games

- Model checking P and R operators for rPATL
  - complexity: NP ∩ coNP (except one case, else NEXP ∩ coNEXP)
  - compared to, e.g., P for Markov decision processes
  - proceeds by evaluation of numerical fixed points (similar to "value iteration")
- Prototype model checker for stochastic games
  - integrated into PRISM model checker
  - PRISM modelling and property specification languages extended, adding SMG to the repertoire of models
- Further case studies
  - e.g. team formation protocols, collective decision making for sensor networks
- Available now:
  - http://www.prismmodelchecker.org/games/

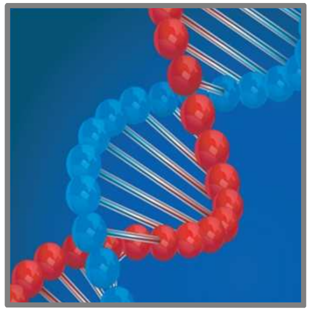
Focus on...

Cooperation

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control



Natural world

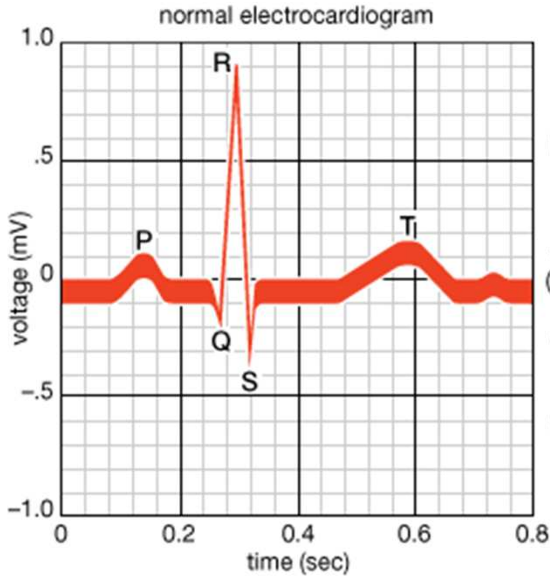
- Biosensing
- Molecular programming

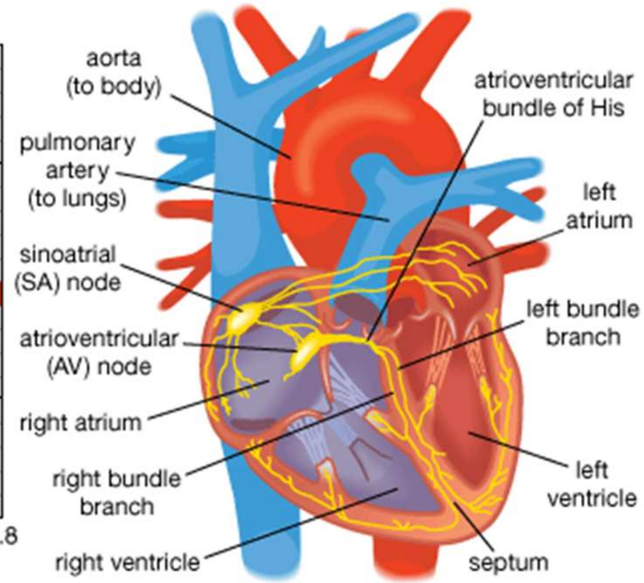
Monitoring physical processes

- 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?

Function of the heart

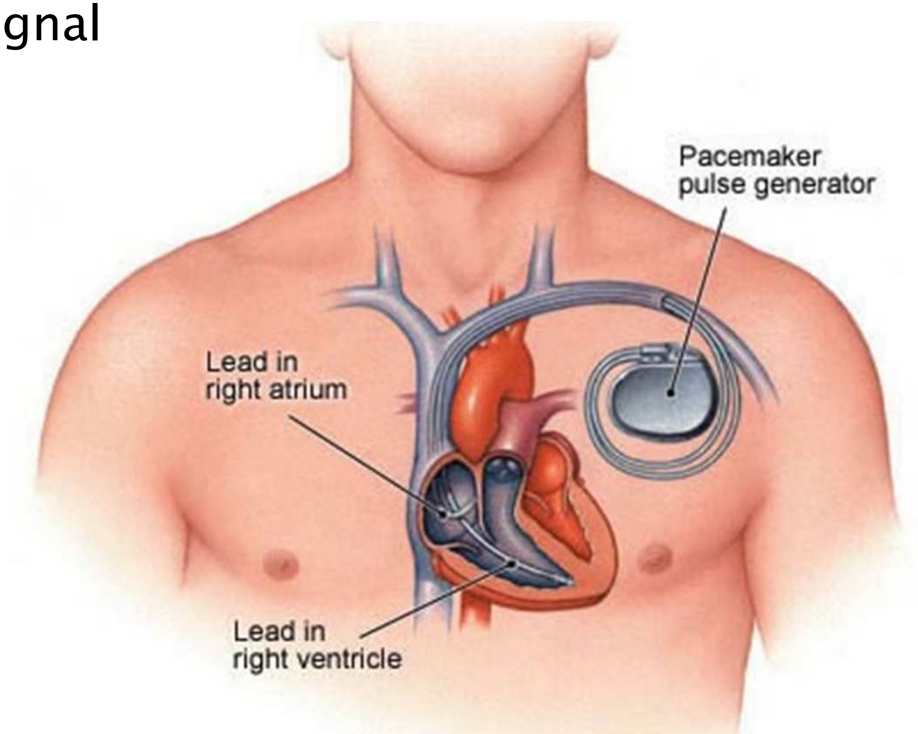
- Maintains blood circulation by contracting the atria and ventricles
  - spontaneously generates electrical signal (action potential)
  - conducted through cellular pathways into atrium, causing contraction of atria then ventricles
  - repeats, maintaining 60-100 beats per minute
  - a real-time system, and natural pacemaker
- Abnormalities in electrical conduction
  - missed/slow heart beat
  - can be corrected by implantable pacemakers



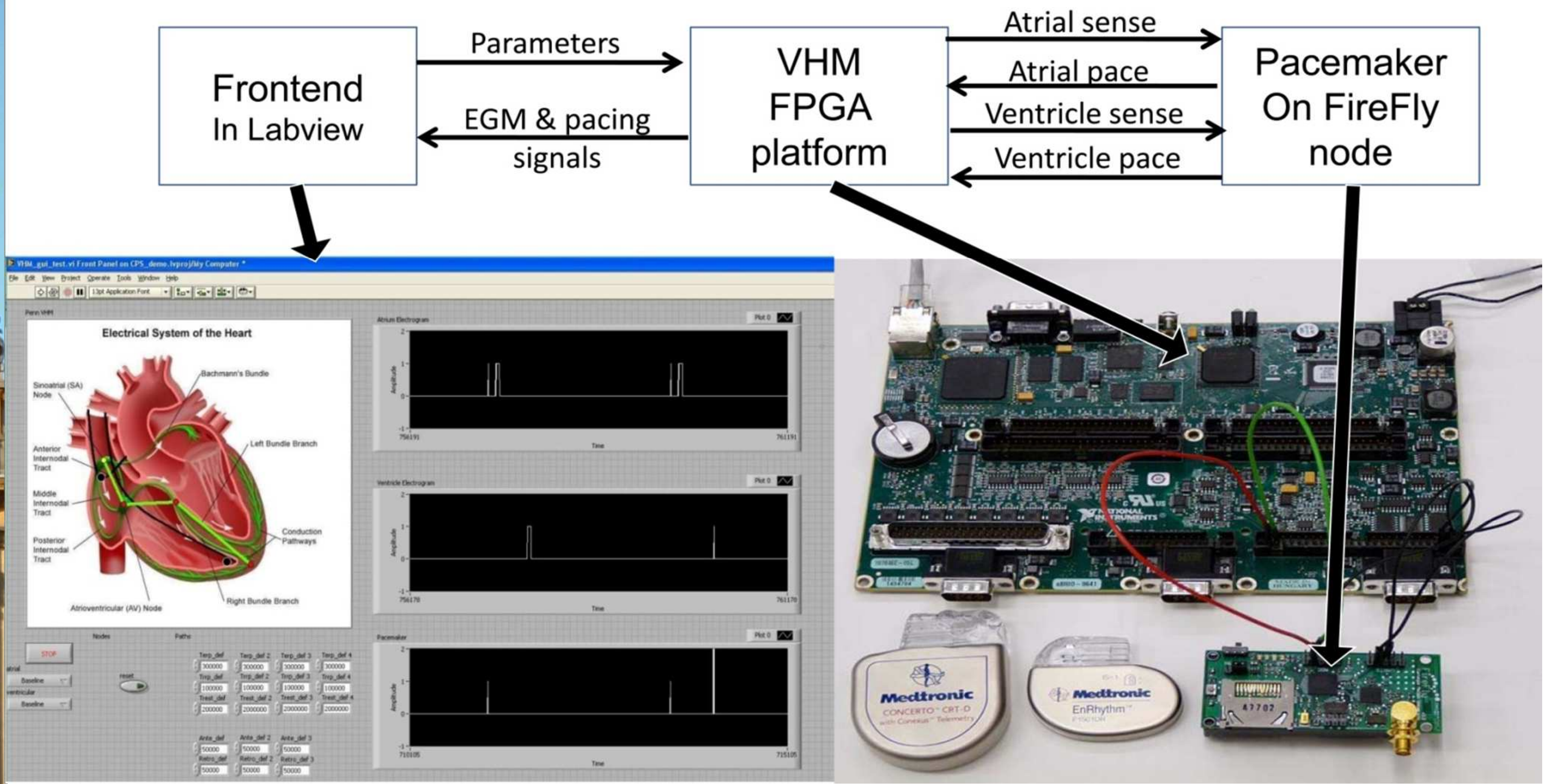


Implantable pacemaker

- 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
- Embedded software
- Widely used, replaced every few years
- Unfortunately…
  - 600,000 devices recalled during 1990-2000
  - 200,000 due to firmware problems



Closed-loop pacemaker testing



FPGA-based system developed at PRECISE Centre, Upenn [Jiang et al]

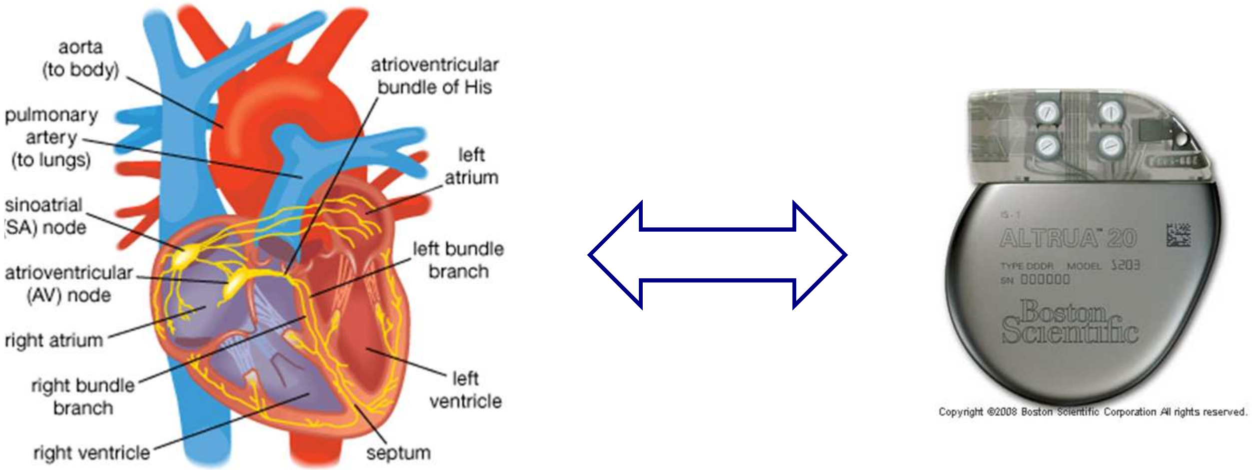
Real pacemaker devices, patient specific, but testing/validation only (various cardiac rhythms)

Quantitative verification for pacemakers?

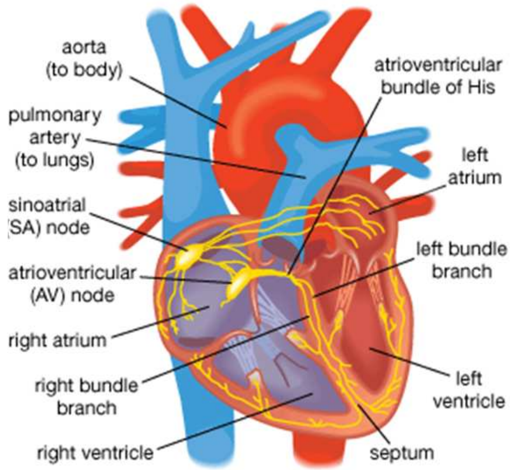
- Pacemaker model
  - various approaches exist, e.g. Simulink, Z and theorem proving, not suitable for quantitative verification
  - here, adopt the timed automata model of [Jiang et al]
- What does correctness mean?
  - the rhythm depends on the patient
  - faulty pacemaker may induce undesirable heart behaviour
- Seek realistic heart models for verification
  - adopt synthetic ECG model (non-linear ODE) [Clifford et al]
  - reflects chest surface measurements, map to action potential
  - probabilistic, can encode various diseases and can be learnt from patient data
- Properties
  - expressible as timed automata or MTL (Metric Temporal Logic)
  - more generally, reward properties for energy usage

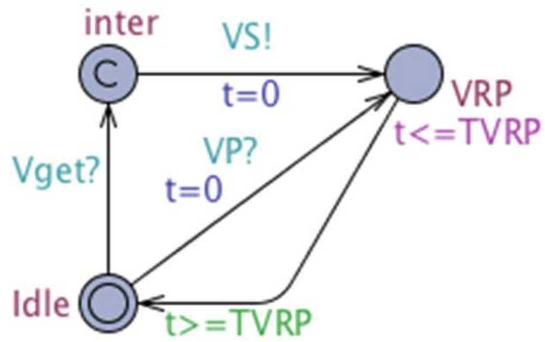
Quantitative verification for pacemakers

- Model the pacemaker and the heart, compose and verify



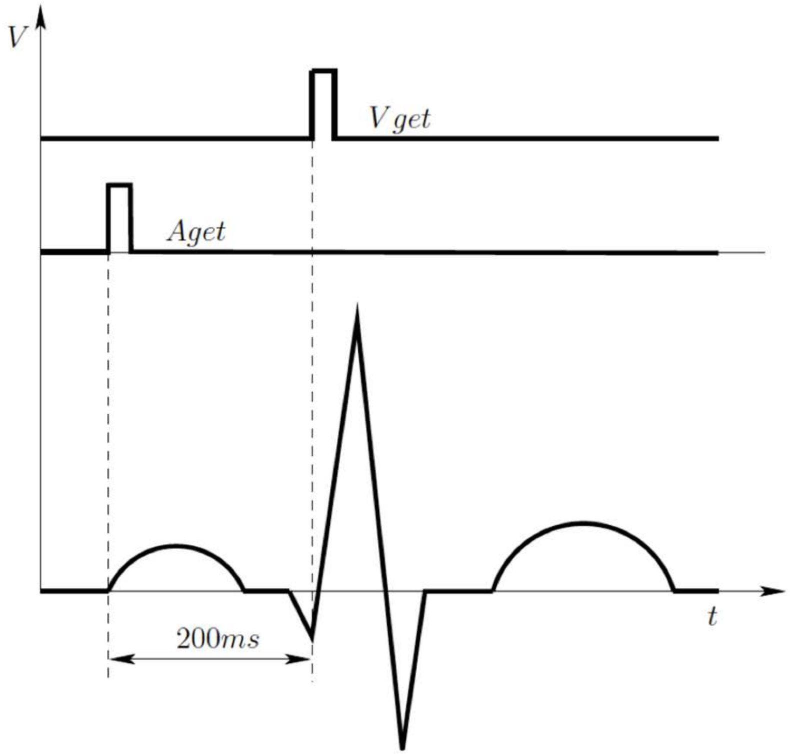
Quantitative verification for pacemakers

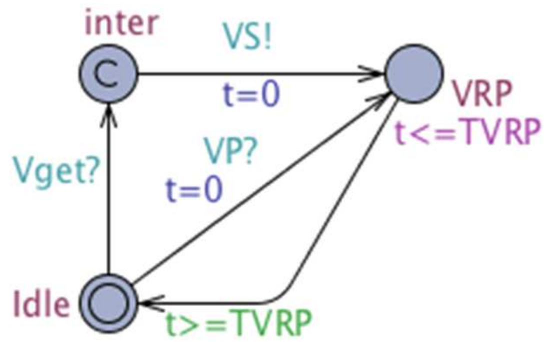





```
module VRP

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);
```

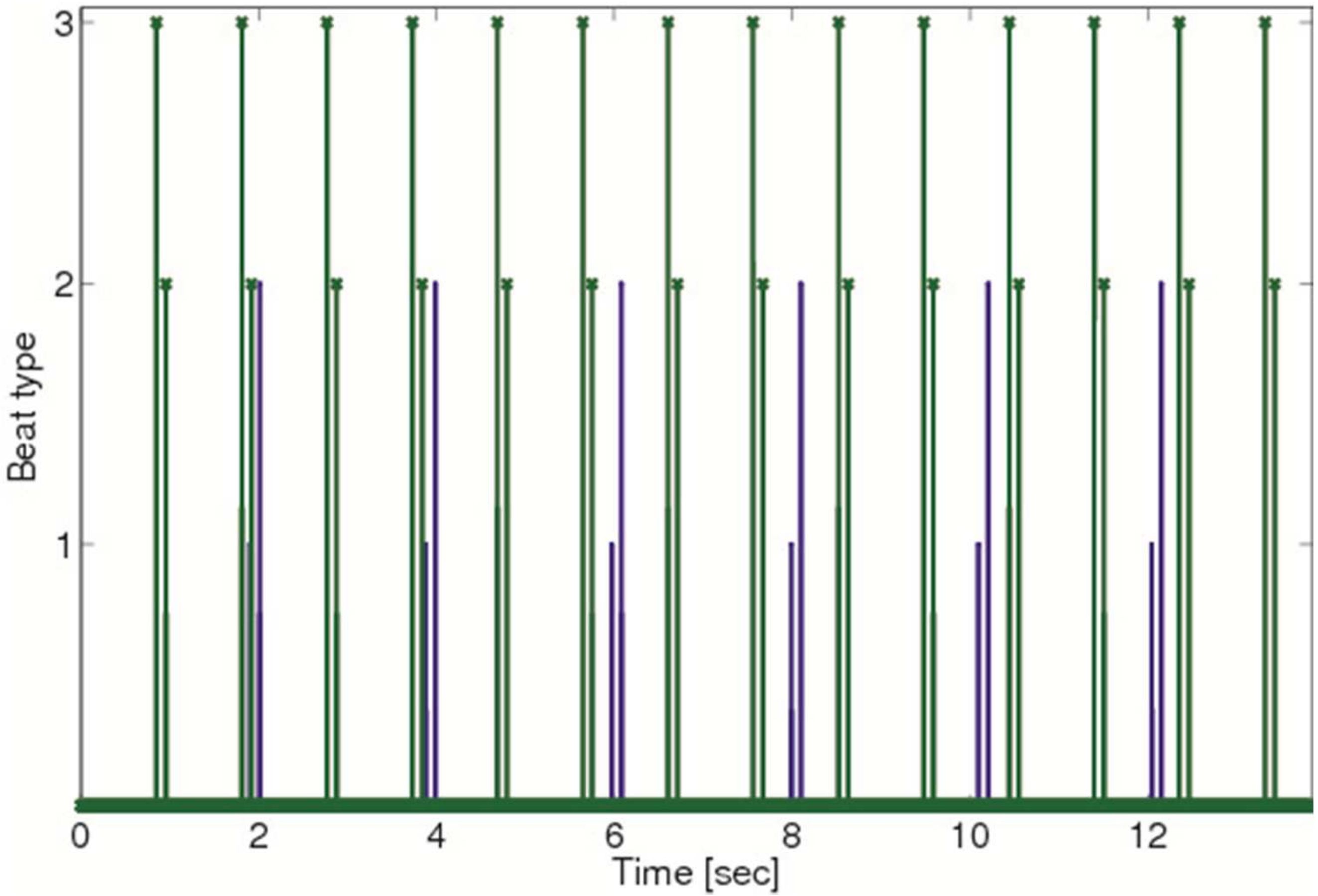
Quantitative verification for pacemakers






```
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);
```

Correction of Bradycardia



Purple lines original (slow) heart beat, green are induced (correcting)

Tool support: PRISM & MATLAB

- Developed and implemented a framework based on (I/O) synchronised composition of
  - discretised heart model (Runge-Kutta)
  - PRISM digital clock models of the pacemaker
- Support for probabilistic analysis
  - probabilistic switching between diseases, can be learnt from patient data
  - undersensing (faulty sensor leads)
  - expected energy usage
- Prototype toolset
  - implemented in MATLAB and PRISM
- Wireless glucose monitors present a greater challenge
- See

  http://www.prismmodelchecker.org/bibitem.php?key=CDKM12b

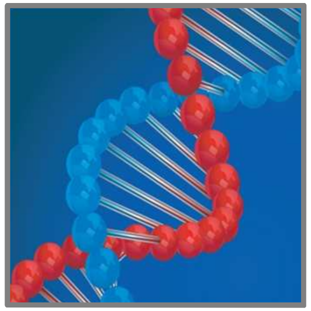
Focus on...

Cooperation

- Self-interest
- Autonomy

Physical processes

- Monitoring
- Control



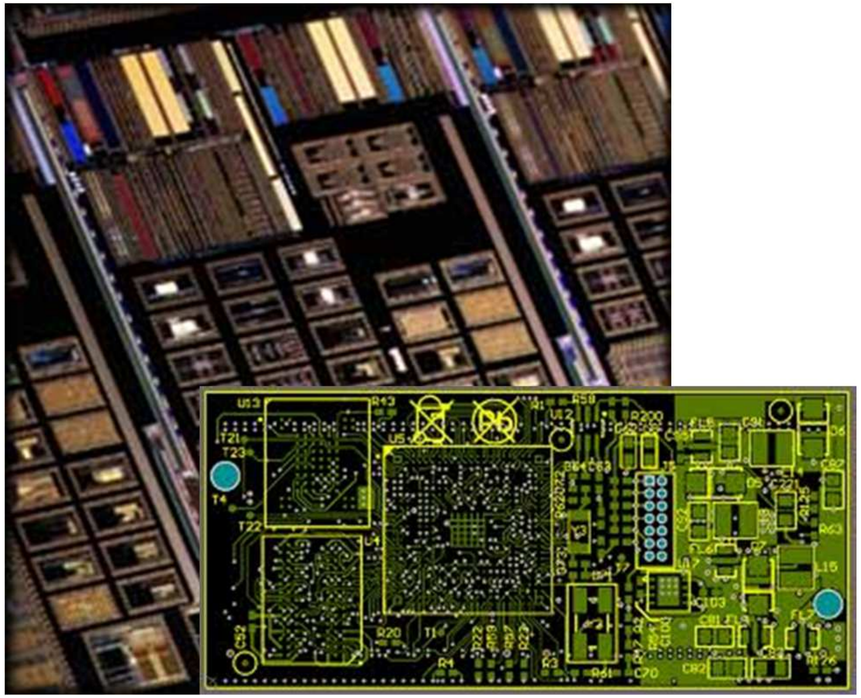
Natural world

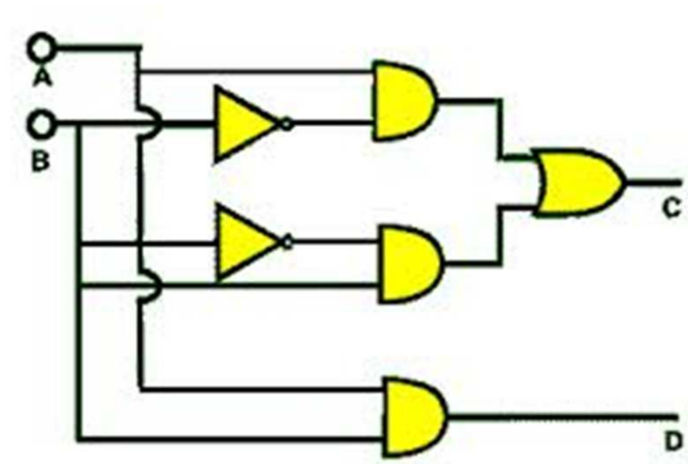
- Biosensing
- Molecular programming

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. 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 devise computing devices using DNA molecules
  - not synthetic biology, but shared techniques and tools
- Research question: can we apply (quantitative) verification to DNA programming?

Digital circuits

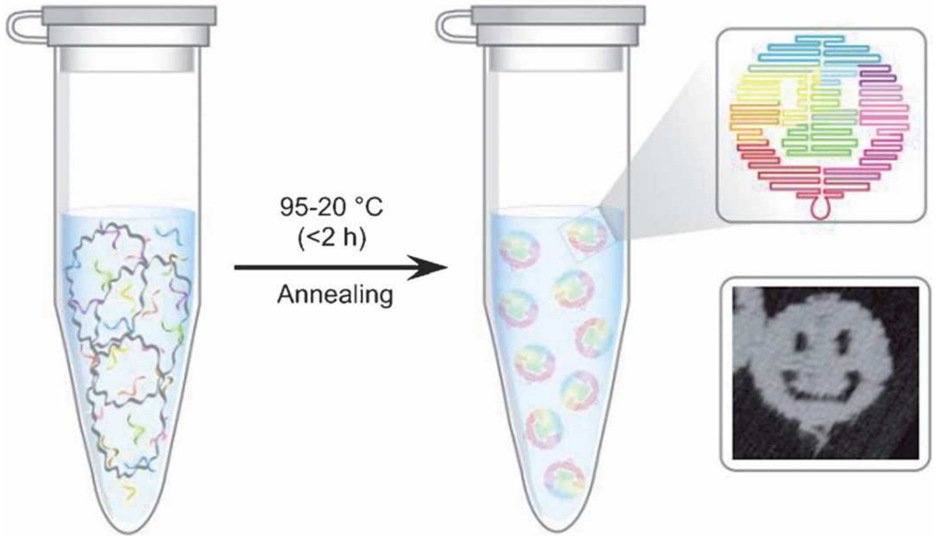




- Logic gates realised in silicon
- 0s and 1s are represented as low and high voltage
- Hardware verification indispensable as design methodology

DNA programming

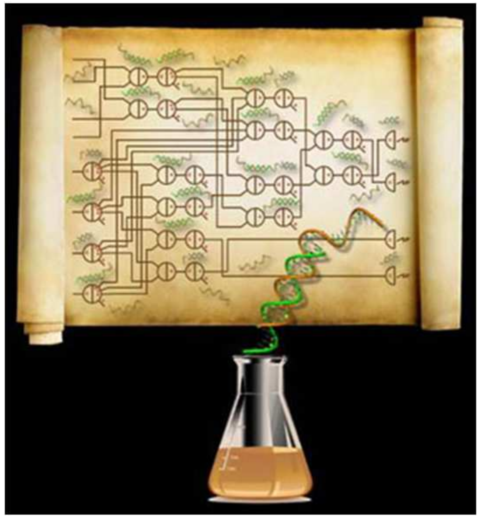
2nm



DNA origami

- "Computing with soup" (The Economist 2012)
  - DNA strands are mixed together in a test tube
  - single strands are inputs and outputs
  - computation proceeds autonomously
- Can we transfer verification to this new application domain?
  - stochasticity essential!

DNA circuits



[Qian, Winfree, *Science* 2012]

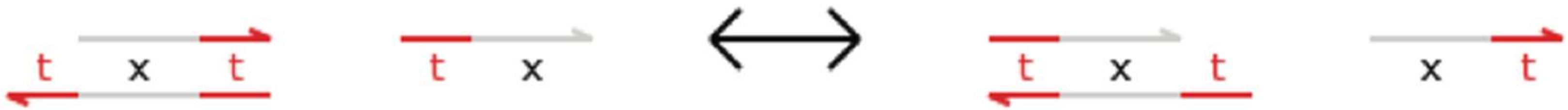
- Techniques exist for designing DNA circuits
- (DNA Strand Displacement)
- Circuit of 130 strands computes square root of 4 bit number, rounded down
- 10 hours, but it's a first…

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

DNA Strand Displacement

- Design (simplified) logic gates in DNA
  - double strands with nicks (interruptions) in the top strand

  - and single strands consisting of one (short) toehold domain t and one recognition domain x

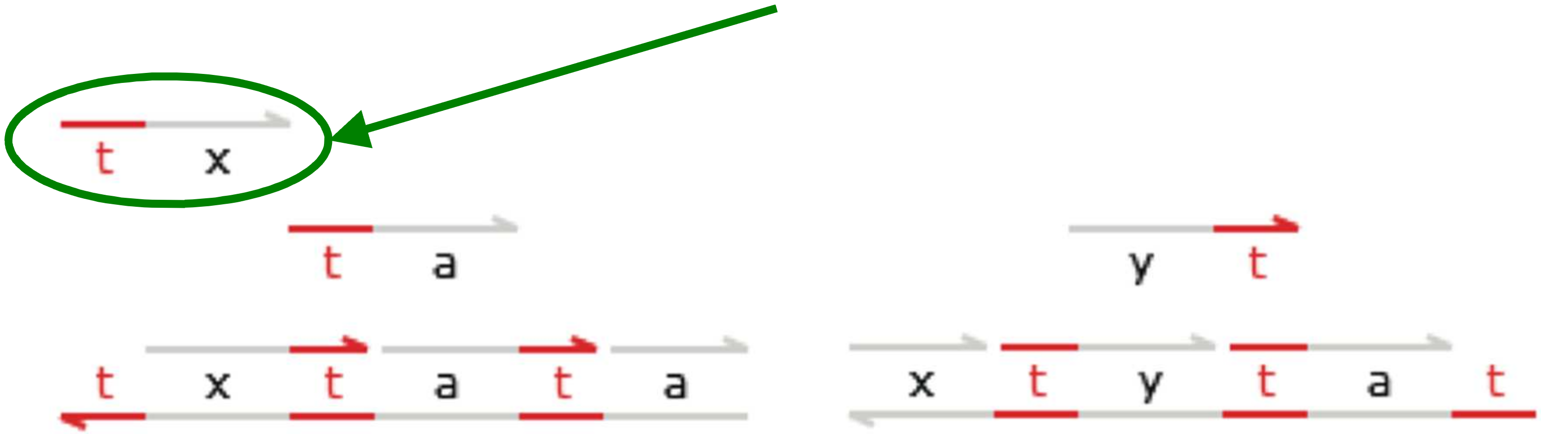


  - “toehold exchange”: branch migration of strand <t^ x> leading to displacement of strand <x t^>
- DSD process algebra semantics due to Cardelli
- DSD programming environment due to Phillips (Microsoft)

[Cardelli’10] Two-Domain DNA Strand Displacement. DCM’10

Example: Transducer

- Transducer: converts input <t^ x> into output <t^ y>

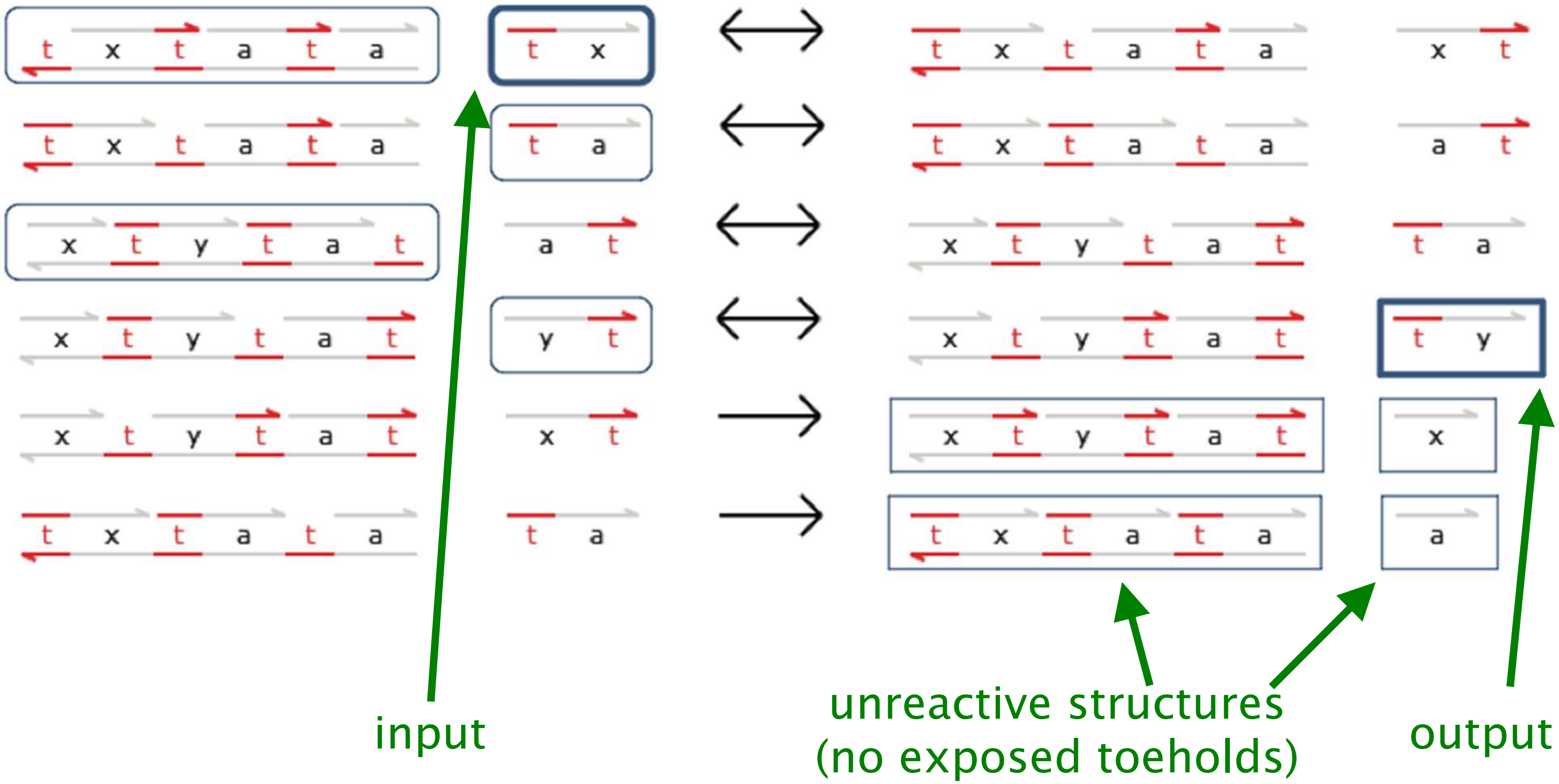


Computation in DNA

http://lucacardelli.name/

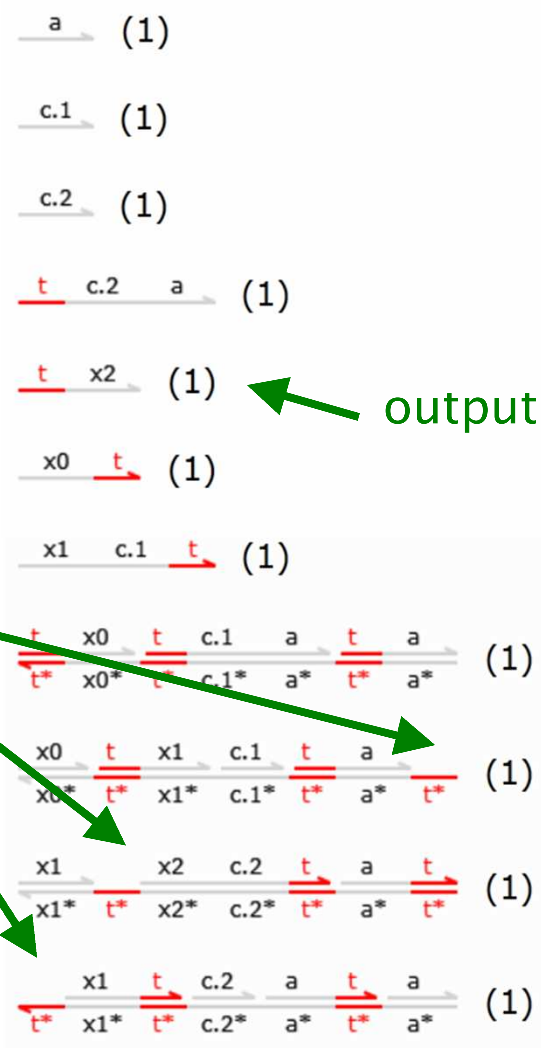
Example: Transducer

- Transducer: full reaction list



Transducer flaw

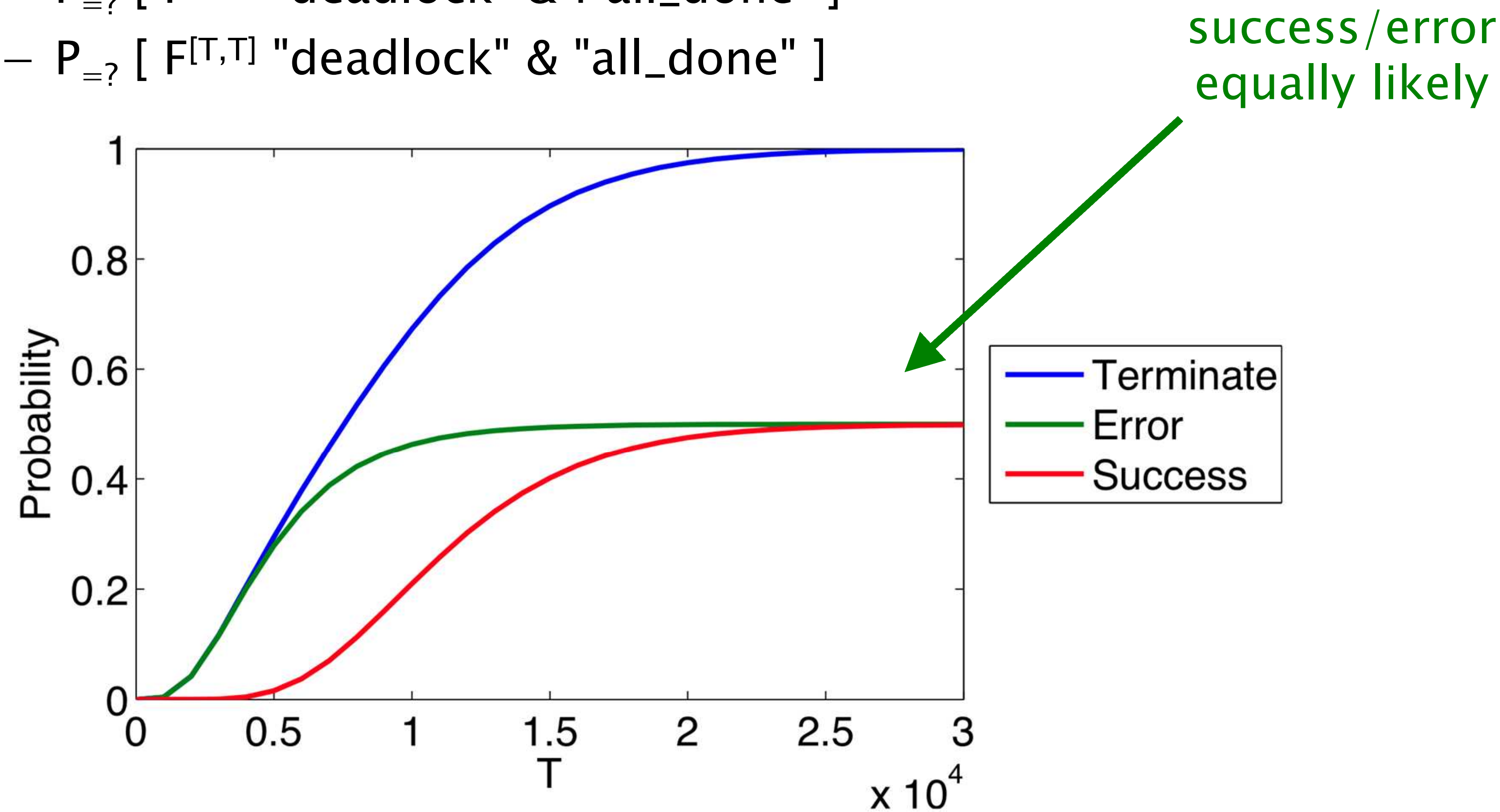
- Unwanted deadlock!
  - OK for one, fails for two copies of the gates
- PRISM identifies a 5-step trace
  - problem caused by "crosstalk" (interference) between DSD species
  - previously found manually [Cardelli'10]
  - detection now fully automated
- Bug is easily fixed
  - (and verified)



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)
(0,1,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,1,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,1,1,0,0,0,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0)
(0,0,1,0,1,1,0,1,0,0,1,0,1,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0)

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"]



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, can only deal with small designs
- Further case studies
  - Approximate Majority population protocol
- Available now:

  http://research.microsoft.com/en-us/projects/dna/

Summing up…

- Brief overview of three directions of particular importance to ubiquitous computing
  - demonstrating first successes and usefulness of quantitative verification methodology
  - and resulting in new techniques and tools
- Many challenges remain
  - for cooperation, addressing more general quantitative goals
  - incorporation of quantitative verification in pacemaker development environments, and
  - scalability of verification for molecular programming models
- More challenges not covered in this lecture
  - controller synthesis, code generation, runtime verification, approximate methods, more expressive models and logics, new application domains, …

References

- Cooperation
  - T. Chen, V. Forejt, M. Kwiatkowska, D. Parker and A. Simaitis. Automatic Verification of Competitive Stochastic Systems. TACAS 2012: 315-330.
- Pacemaker
  - T. Chen, M. Diciolla, M. Kwiatkowska and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. RTSS 2012.
  - See also Jiang et al: Modeling and Verification of a Dual Chamber Implantable Pacemaker. TACAS 2012: 188-203.
- DNA programming
  - M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska and A. Phillips. Design and Analysis of DNA Strand Displacement Devices using Probabilistic Model Checking. J R Soc Interface, 9(72), 1470-1485, 2012.
- See also
  - M. Kwiatkowska, G. Norman and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. CAV 2011: 585-591.

Acknowledgements

- My group and collaborators in this work
  - Luca Cardelli, Taolue Chen, Marco Diciolla, Vojtech Forejt, Matthew Lakin, Alexandru Mereacre, Gethin Norman, Dave Parker, Andrew Phillips, Aistis Simajtis
- Collaborators who contributed to theoretical and practical PRISM development
- External users of and contributors to PRISM
- Project funding
  - ERC, EPSRC LSCITS
  - Oxford Martin School, Institute for the Future of Computing
- See also
  - VERIWARE www.veriware.org
  - PRISM www.prismmodelchecker.org