Probabilistic model checking with PRISM

Marta Kwiatkowska

Department of Computer Science, University of Oxford

4th SSFT, Menlo College, May 2014

What is probabilistic model checking?

- Probabilistic model checking…
  - is a formal verification technique
    for modelling and analysing systems
    that exhibit probabilistic behaviour

- Formal verification…
  - is the application of rigorous,
    mathematics-based techniques
    to establish the correctness
    of computerised systems

Why formal verification?

- Errors in computerised systems can be costly…

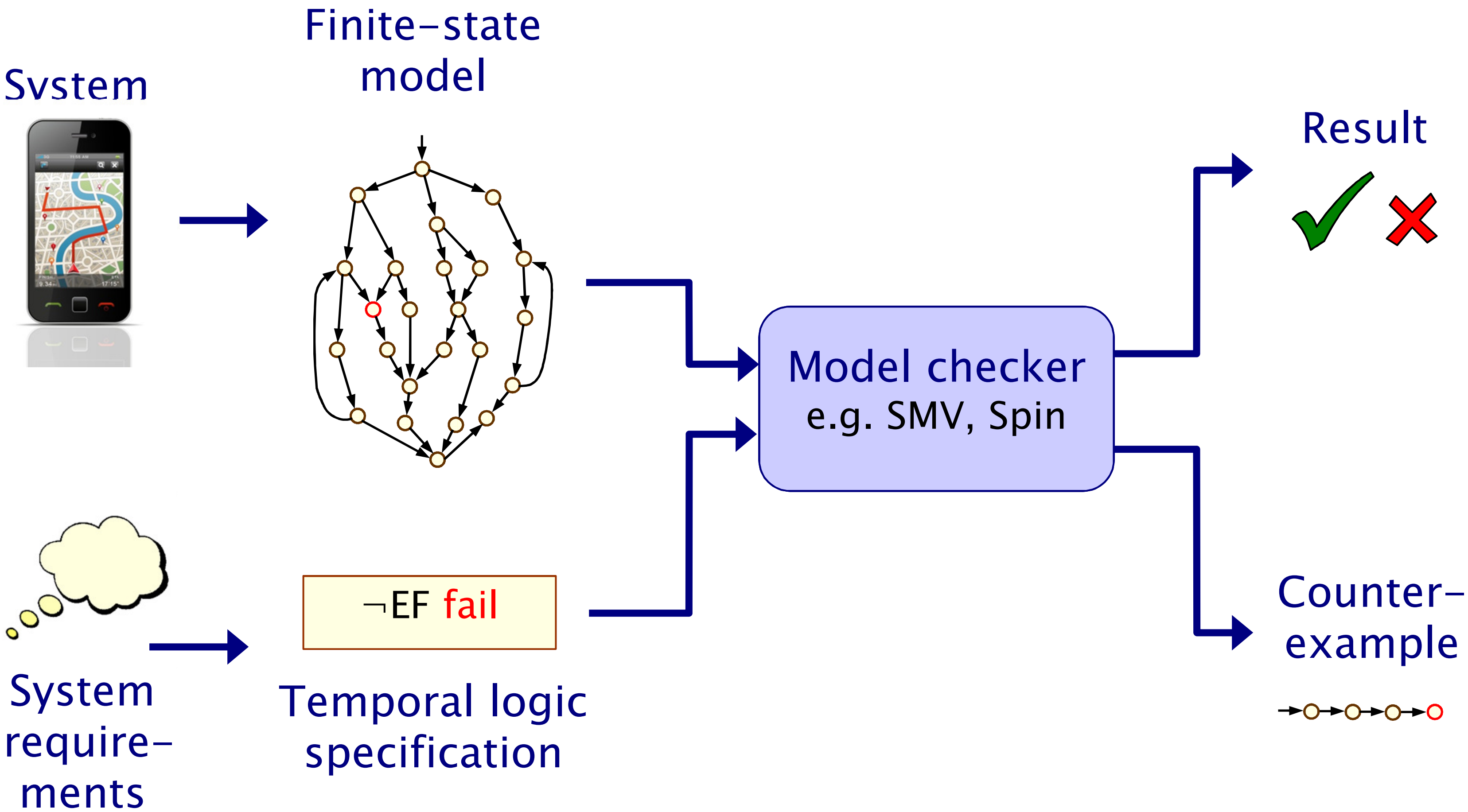
Pentium chip (1994)
Bug found in FPU.
Intel (eventually) offers to replace faulty chips.
Estimated loss: $475m

Infusion pumps (2010)
Patients die because of incorrect dosage.
Cause: software malfunction.
79 recalls.

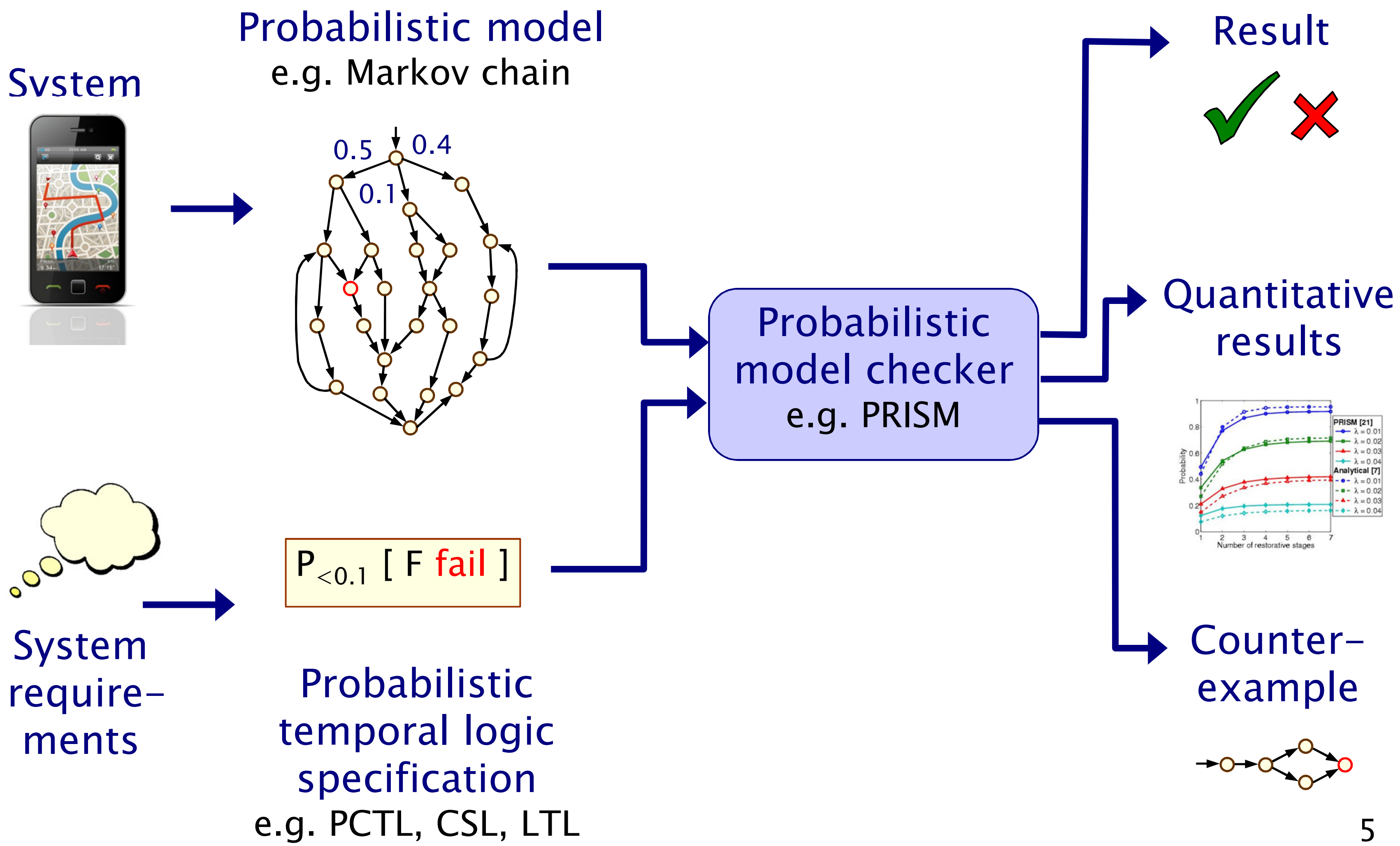
Toyota Prius (2010)
Software "glitch" found in anti-lock braking system.
185,000 cars recalled.

- Why verify?
  - "Testing can only show the presence of errors, not their absence." [Edsger Dijstra]

Model checking



Probabilistic model checking



Why probability?

- Some systems are inherently probabilistic…

- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding

- Examples: real-world protocols featuring randomisation:
  - Randomised back-off schemes
    - CSMA protocol, 802.11 Wireless LAN
  - Random choice of waiting time
    - IEEE1394 Firewire (root contention), Bluetooth (device discovery)
  - Random choice over a set of possible addresses
    - IPv4 Zeroconf dynamic configuration (link-local addressing)
  - Randomised algorithms for anonymity, contract signing, …

Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- Examples:
  - computer networks, embedded systems
  - power management policies
  - nano-scale circuitry: reliability through defect-tolerance

Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- To model biological processes
  - reactions occurring between large numbers of molecules are naturally modelled in a stochastic fashion

Verifying probabilistic systems

- We are not just interested in correctness
- We want to be able to quantify:
  - security, privacy, trust, anonymity, fairness
  - safety, reliability, performance, dependability
  - resource usage, e.g. battery life
  - and much more…
- Quantitative, as well as qualitative requirements:
  - how reliable is my car's Bluetooth network?
  - how efficient is my phone's power management policy?
  - is my bank's web-service secure?
  - what is the expected long-run percentage of protein X?

Probabilistic models

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) Simple stochastic games (SMGs)
Continuous time	Continuous-time Markov chains (CTMCs)	Probabilistic timed automata (PTAs) Interactive Markov chains (IMCs)

Probabilistic models

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) Simple stochastic games (SMGs)
Continuous time	Continuous-time Markov chains (CTMCs)	Probabilistic timed automata (PTAs) Interactive Markov chains (IMCs)

Course material

- 4th SSFT slides and lab session
  - http://www.prismmodelchecker.org/courses/ssft14/
- Reading
  - [MDPs/LTL] Forejt, Kwiatkowska, Norman and Parker. Automated Verification Techniques for Probabilistic Systems. LNCS vol 6659, p53-113, Springer 2011.
  - [DTMCs/CTMCs] Kwiatkowska, Norman and Parker. Stochastic Model Checking. LNCS vol 4486, p220-270, Springer 2007.
  - [DTMCs/MDPs/LTL] Principles of Model Checking by Baier and Katoen, MIT Press 2008
- See also
  - 20 lecture course taught at Oxford
  - http://www.prismmodelchecker.org/lectures/pmc/
- PRISM website www.prismmodelchecker.org

Part 1

Discrete-time Markov chains

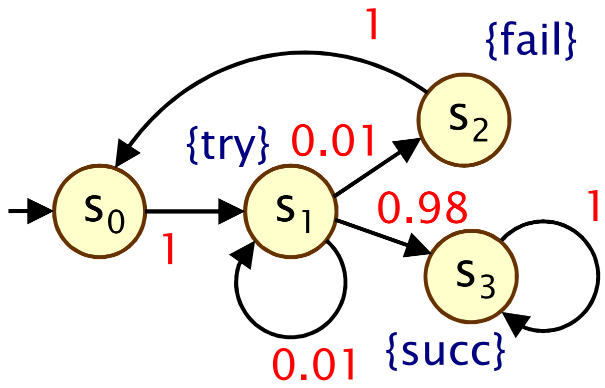
Overview (Part 1)

- Introduction
- Model checking for discrete-time Markov chains (DTMCs)
  - DTMCs: definition, paths & probability spaces
  - PCTL model checking
  - Costs and rewards
- PRISM: overview
  - Modelling language
  - Properties
  - GUI, etc
  - Case studies: Bluetooth, DNA programming
- Summary

Discrete-time Markov chains

- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities

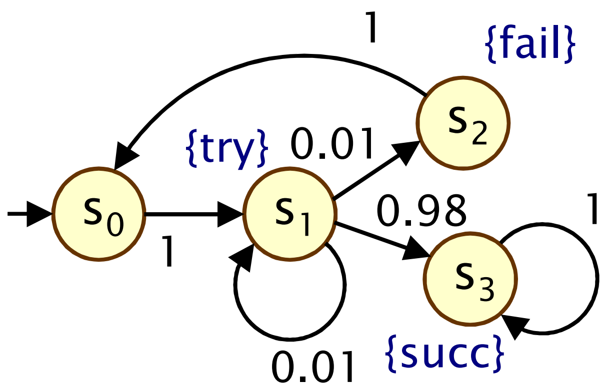
- States
  - discrete set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability of making transitions between states is given by discrete probability distributions



Discrete-time Markov chains

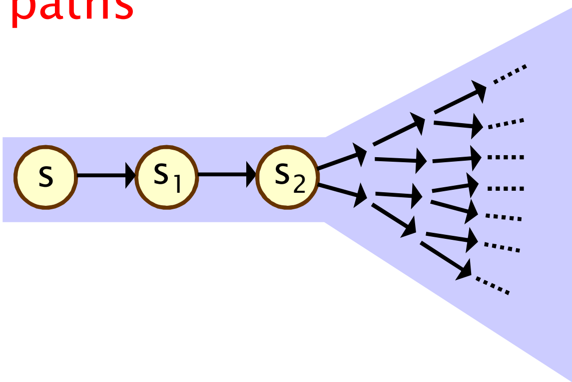
- Formally, a DTMC D is a tuple (S, s_{init}, P, L) where:
  - S is a finite set of states (“state space”)
  - $s_{init} \in S$ is the initial state
  - $P : S \times S \rightarrow [0,1]$ is the transition probability matrix where $\Sigma_{s' \in S} P(s,s') = 1$ for all $s \in S$
  - $L : S \rightarrow 2^{AP}$ is function labelling states with atomic propositions

- Note: no deadlock states
  - i.e. every state has at least one outgoing transition
  - can add self loops to represent final/terminating states



Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0 s_1 s_2 s_3 \ldots$ such that $\mathbf{P}(s_i, s_{i+1}) > 0 \; \forall i$
  - represents an execution (i.e. one possible behaviour) of the system which the DTMC is modelling
- To reason (quantitatively) about this system
  - need to define a probability space over paths
- Intuitively:
  - sample space: Path(s) = set of all infinite paths from a state s
  - events: sets of infinite paths from s
  - basic events: cylinder sets (or "cones")
  - cylinder set C(ω), for a finite path ω = set of infinite paths with the common finite prefix ω
  - for example: $C(ss_1s_2)$

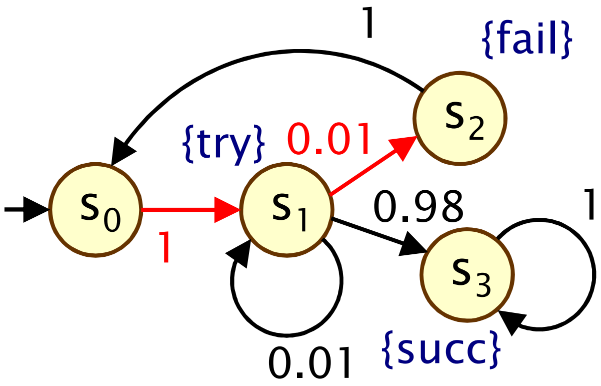


Probability space over paths

- Sample space Ω = Path(s)
  set of infinite paths with initial state s
- Event set $\Sigma_{Path(s)}$
  - the cylinder set $C(\omega) = \{ \omega' \in Path(s) \mid \omega$ is prefix of $\omega' \}$
  - $\Sigma_{Path(s)}$ is the least σ-algebra on Path(s) containing C(ω) for all finite paths ω starting in s
- Probability measure Pr_s
  - define probability $P_s(\omega)$ for finite path $\omega = ss_1 \ldots s_n$ as:
    - $P_s(\omega) = 1$ if ω has length one (i.e. ω = s)
    - $P_s(\omega) = P(s,s_1) \cdot \ldots \cdot P(s_{n-1},s_n)$ otherwise
    - define $Pr_s(C(\omega)) = P_s(\omega)$ for all finite paths · ω
  - Pr_s extends uniquely to a probability measure $Pr_s:\Sigma_{Path(s)} \rightarrow [0,1]$
- See [KSK76] for further details

Probability space – Example

- Paths where sending fails the first time
  - $\omega = s_0 s_1 s_2$
  - $C(\omega)$ = all paths starting $s_0 s_1 s_2 \ldots$
  - $P_{s0}(\omega) = P(s_0,s_1) \cdot P(s_1,s_2)$
    $= 1 \cdot 0.01 = 0.01$
  - $Pr_{s0}(C(\omega)) = P_{s0}(\omega) = 0.01$



- Paths which are eventually successful and with no failures
  - $C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots$
  - $Pr_{s0}(\, C(s_0s_1s_3) \cup C(s_0s_1s_1s_3) \cup C(s_0s_1s_1s_1s_3) \cup \ldots\,)$
    $= P_{s0}(s_0s_1s_3) + P_{s0}(s_0s_1s_1s_3) + P_{s0}(s_0s_1s_1s_1s_3) + \ldots$
    $= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots$
    $= 0.9898989898\ldots$
    $= 98/99$

PCTL

- Temporal logic for describing properties of DTMCs
  - PCTL = Probabilistic Computation Tree Logic [HJ94]
  - essentially the same as the logic pCTL of [ASB+95]

- Extension of (non-probabilistic) temporal logic CTL
  - key addition is probabilistic operator P
  - quantitative extension of CTL's A and E operators

- Example
  - send → $P_{\geq 0.95}$ [true $U^{\leq 10}$ deliver]
  - "if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95"

PCTL syntax

- PCTL syntax:
  - $\phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg\phi \mid P_{\sim p} [\, \psi \,]$ (state formulas)
  - $\psi ::= X\,\phi \mid \phi\, U^{\leq k}\, \phi \mid \phi\, U\, \phi$ (path formulas)

ψ is true with probability ~p

"next" "bounded until" "until"

  - define $F\,\phi \equiv \text{true}\; U\; \phi$ (eventually), $G\,\phi \equiv \neg(F\,\neg\phi)$ (globally)
  - where a is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

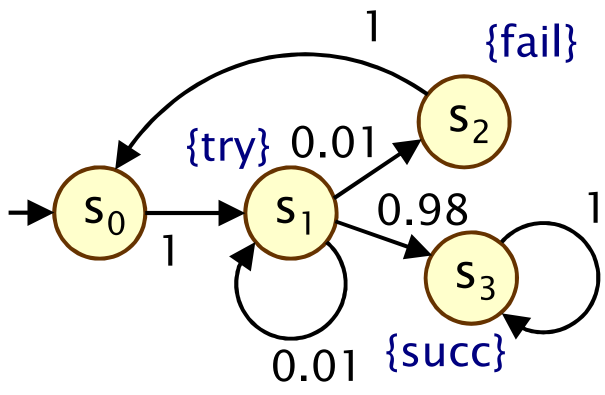
- A PCTL formula is always a state formula
  - path formulas only occur inside the P operator

PCTL semantics for DTMCs

- PCTL formulas interpreted over states of a DTMC
  - $s \vDash \phi$ denotes ϕ is "true in state s" or "satisfied in state s"

- Semantics of (non-probabilistic) state formulas:
  - for a state s of the DTMC (S,s_{init},P,L):
  - $s \vDash a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \vDash \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \vDash \phi_1$ and $s \vDash \phi_2$
  - $s \vDash \neg\phi \quad \Leftrightarrow \quad s \vDash \phi$ is false

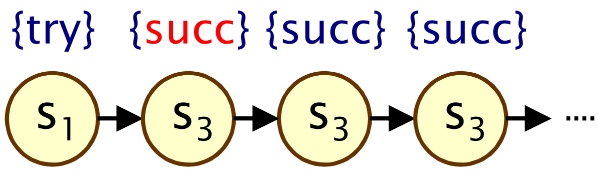
- Examples
  - $s_3 \vDash succ$
  - $s_1 \vDash try \wedge \neg fail$



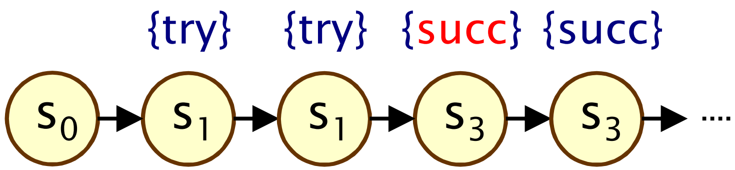
PCTL semantics for DTMCs

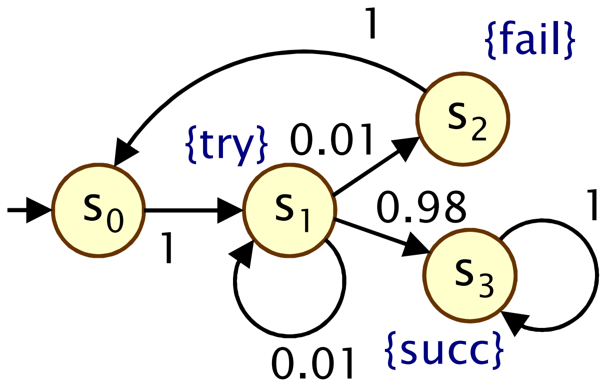
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \models X\ \phi \quad \Leftrightarrow \quad s_1 \models \phi$
  - $\omega \models \phi_1\ U^{\leq k}\ \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \models \phi_2$ and $\forall j < i,\ s_j \models \phi_1$
  - $\omega \models \phi_1\ U\ \phi_2 \quad \Leftrightarrow \quad \exists k \geq 0$ such that $\omega \models \phi_1\ U^{\leq k}\ \phi_2$

- Some examples of satisfying paths:
  - X succ



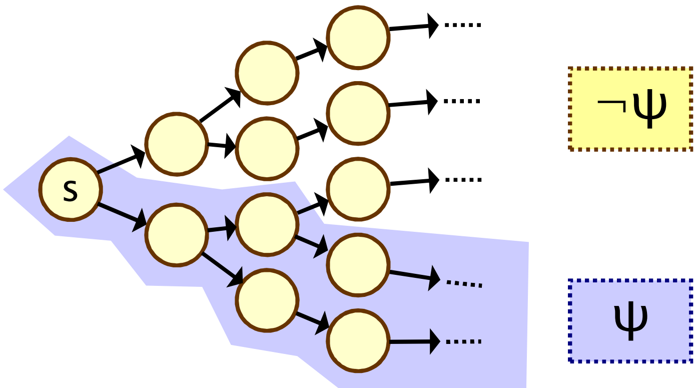
  - ¬fail U succ





PCTL semantics for DTMCs

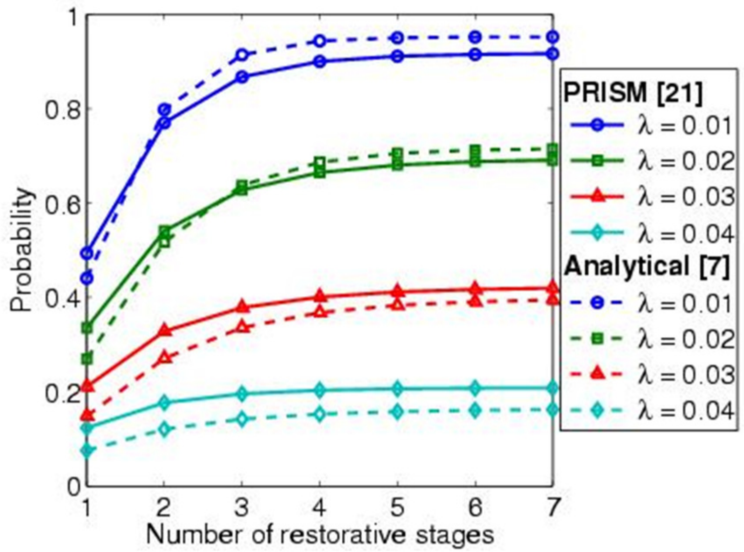
- Semantics of the probabilistic operator P
  - informal definition: $s \vDash P_{\sim p} [\ \psi\ ]$ means that "the probability, from state s, that ψ is true for an outgoing path satisfies ~p"
  - example: $s \vDash P_{<0.25} [\ X\ fail\ ] \Leftrightarrow$ "the probability of atomic proposition fail being true in the next state of outgoing paths from s is less than 0.25"
  - formally: $s \vDash P_{\sim p} [\psi] \ \Leftrightarrow\ Prob(s, \psi) \sim p$
  - where: $Prob(s, \psi) = Pr_s \{ \omega \in Path(s) \mid \omega \vDash \psi \}$
  - (sets of paths satisfying ψ are always measurable [Var85])



Prob(s, ψ) ~ p ?

Quantitative properties

- Consider a PCTL formula $P_{\sim p}$ [ψ]
  - if the probability is unknown, how to choose the bound p?
- When the outermost operator of a PTCL formula is P
  - we allow the form $P_{=?}$ [ψ]
  - "what is the probability that path formula ψ is true?"
- Model checking is no harder: compute the values anyway
- Useful to spot patterns, trends



- Example
  - $P_{=?}$ [F err/total>0.1]
  - "what is the probability that 10% of the NAND gate outputs are erroneous?"

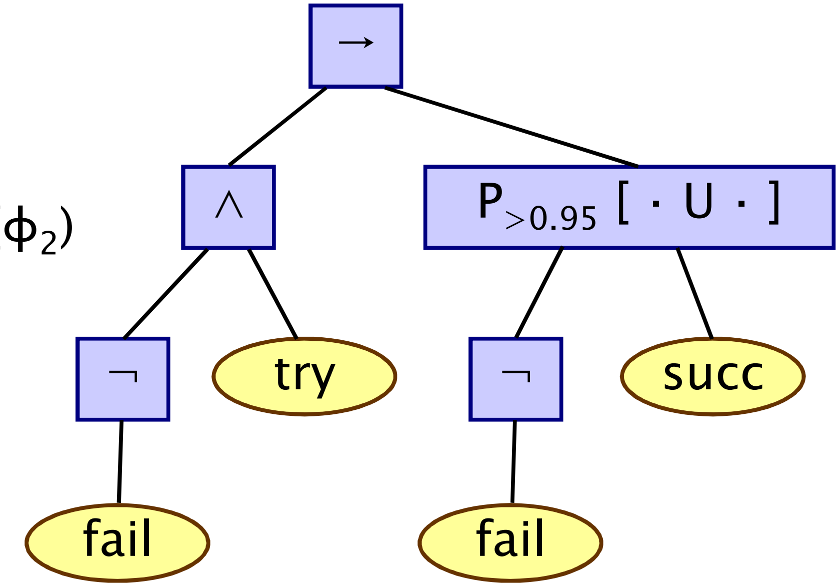
PCTL model checking for DTMCs

- Algorithm for PCTL model checking [CY88,HJ94,CY95]
  - inputs: DTMC $D=(S,s_{init},P,L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = set of states satisfying ϕ
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - sometimes, want to check that $s \vDash \phi \ \forall\ s \in S$, i.e. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{init} \vDash \phi$, i.e. if $s_{init} \in Sat(\phi)$
- Sometimes, focus on quantitative results
  - e.g. compute result of P=? [F error]
  - e.g. compute result of P=? [$F^{\leq k}$ error] for $0 \leq k \leq 100$

PCTL model checking for DTMCs

- Basic algorithm proceeds by induction on parse tree of ϕ
  - example: $\phi = (\neg\text{fail} \wedge \text{try}) \rightarrow P_{>0.95} [\, \neg\text{fail}\ U\ \text{succ}\,]$

- For the non-probabilistic operators:
  - $\text{Sat}(\text{true}) = S$
  - $\text{Sat}(a) = \{ s \in S \mid a \in L(s) \}$
  - $\text{Sat}(\neg\phi) = S \setminus \text{Sat}(\phi)$
  - $\text{Sat}(\phi_1 \wedge \phi_2) = \text{Sat}(\phi_1) \cap \text{Sat}(\phi_2)$



- For the $P_{\sim p} [\, \psi\,]$ operator
  - need to compute the probabilities $\text{Prob}(s, \psi)$ for all states $s \in S$
  - focus here on "until" case: $\psi = \phi_1\ U\ \phi_2$

PCTL until for DTMCs

- Computation of probabilities $Prob(s, \phi_1 \text{ U } \phi_2)$ for all $s \in S$
- First, identify all states where the probability is 1 or 0
  - $S^{yes} = Sat(P_{\geq 1} [\ \phi_1 \text{ U } \phi_2\ ])$
  - $S^{no} = Sat(P_{\leq 0} [\ \phi_1 \text{ U } \phi_2\ ])$
- Then solve linear equation system for remaining states

- We refer to the first phase as "precomputation"
  - two algorithms: Prob0 (for S^{no}) and Prob1 (for S^{yes})
  - algorithms work on underlying graph (probabilities irrelevant)
- Important for several reasons
  - reduces the set of states for which probabilities must be computed numerically (which is more expensive)
  - gives exact results for the states in S^{yes} and S^{no} (no round-off)
  - for $P_{\sim p}[\cdot]$ where p is 0 or 1, no further computation required

PCTL until – Linear equations

- Probabilities Prob(s, ϕ_1 U ϕ_2) can now be obtained as the unique solution of the following set of linear equations:

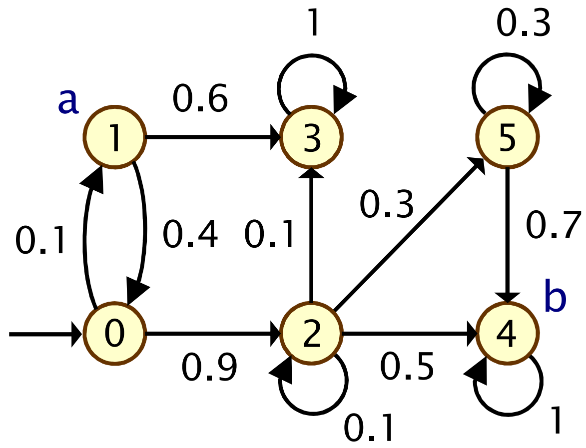
$$\mathrm{Prob}(s, \phi_1 \ U \ \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot \mathrm{Prob}(s', \phi_1 \ U \ \phi_2) & \text{otherwise} \end{cases}$$

  - can be reduced to a system in $|S^?|$ unknowns instead of $|S|$ where $S^? = S \setminus (S^{yes} \cup S^{no})$

- This can be solved with (a variety of) standard techniques
  - direct methods, e.g. Gaussian elimination
  - iterative methods, e.g. Jacobi, Gauss-Seidel, … (preferred in practice due to scalability)

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

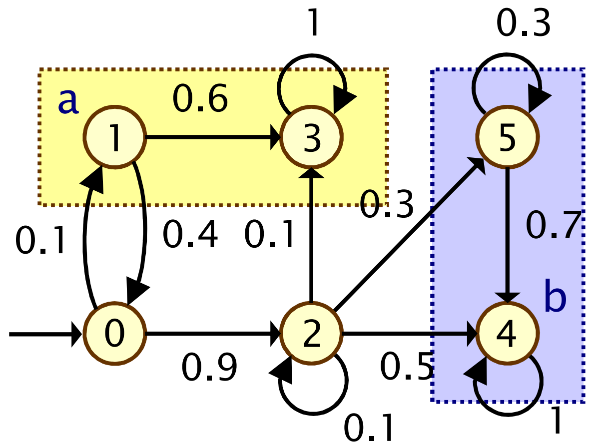


PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

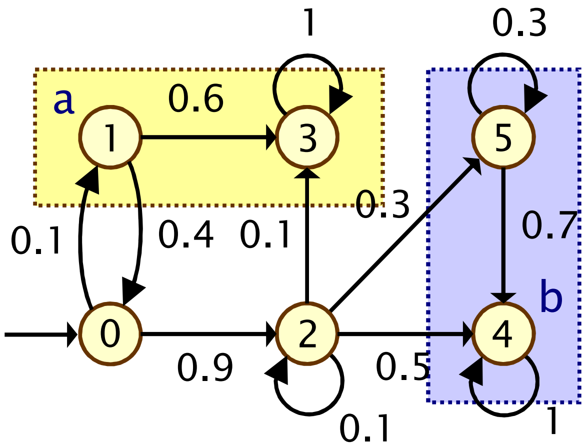
Sat($P_{\geq 1}$ [¬a U b])

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]
- Let x_s = Prob(s, ¬a U b)
- Solve:

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

Sat($P_{\geq 1}$ [¬a U b])

$x_4 = x_5 = 1$

$x_1 = x_3 = 0$

$x_0 = 0.1x_1+0.9x_2 = 0.8$

$x_2 = 0.1x_2+0.1x_3+0.3x_5+0.5x_4 = 8/9$

Prob(¬a U b) = x = [0.8, 0, 8/9, 0, 1, 1]

Sat($P_{>0.8}$ [¬a U b]) = { s_2,s_4,s_5 }

PCTL model checking – Summary

- Computation of set Sat(Φ) for DTMC D and PCTL formula Φ
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation

- Probabilistic operator P:
  - X Φ : one matrix-vector multiplication, $O(|S|^2)$
  - $\Phi_1 \; U^{\leq k} \; \Phi_2$: k matrix-vector multiplications, $O(k|S|^2)$
  - $\Phi_1 \; U \; \Phi_2$: linear equation system, at most |S| variables, $O(|S|^3)$

- Complexity:
  - linear in |Φ| and polynomial in |S|

Limitations of PCTL

- PCTL, although useful in practice, has limited expressivity
  - essentially: probability of reaching states in X, passing only through states in Y (and within k time-steps)

- More expressive logics can be used, for example:
  - LTL [Pnu77] – (non-probabilistic) linear-time temporal logic
  - PCTL* [ASB+95,BdA95] – which subsumes both PCTL and LTL
  - both allow path operators to be combined
  - (in PCTL, $P_{\sim p}$ […] always contains a single temporal operator)
  - supported by PRISM
  - (not covered in this lecture)

- Another direction: extend DTMCs with costs and rewards…

Costs and rewards

- We augment DTMCs with rewards (or, conversely, costs)
  - real-valued quantities assigned to states and/or transitions
  - these can have a wide range of possible interpretations

- Some examples:
  - elapsed time, power consumption, size of message queue, number of messages successfully delivered, net profit, …

- Costs? or rewards?
  - mathematically, no distinction between rewards and costs
  - when interpreted, we assume that it is desirable to minimise costs and to maximise rewards
  - we will consistently use the terminology “rewards” regardless

Reward-based properties

- Properties of DTMCs augmented with rewards
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards
  - formal property specifications will be in an extension of PCTL

- More precisely, we use two distinct classes of property…

- Instantaneous properties
  - the expected value of the reward at some time point

- Cumulative properties
  - the expected cumulated reward over some period

DTMC reward structures

- For a DTMC $(S, s_{init}, \mathbf{P}, L)$, a reward structure is a pair $(\underline{\rho}, \iota)$
  - $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is the state reward function (vector)
  - $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is the transition reward function (matrix)

- Example (for use with instantaneous properties)
  - "size of message queue": $\underline{\rho}$ maps each state to the number of jobs in the queue in that state, ι is not used

- Examples (for use with cumulative properties)
  - "time-steps": $\underline{\rho}$ returns 1 for all states and ι is zero (equivalently, $\underline{\rho}$ is zero and ι returns 1 for all transitions)
  - "number of messages lost": $\underline{\rho}$ is zero and ι maps transitions corresponding to a message loss to 1
  - "power consumption": $\underline{\rho}$ is defined as the per-time-step energy consumption in each state and ι as the energy cost of each transition

PCTL and rewards

- Extend PCTL to incorporate reward-based properties
  - add an R operator, which is similar to the existing P operator

expected reward is ~r

- ϕ ::= … | $P_{\sim p}$ [ψ] | $R_{\sim r}$ [$I^{=k}$] | $R_{\sim r}$ [$C^{\leq k}$] | $R_{\sim r}$ [F ϕ]

"instantaneous" "cumulative" "reachability"

  - where $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

- $R_{\sim r}$ [·] means "the expected value of · satisfies ~r"

Reward formula semantics

- Formal semantics of the three reward operators
  - based on random variables over (infinite) paths

- Recall:
  - $s \models P_{\sim p}\ [\ \psi\ ]\ \Leftrightarrow\ Pr_s\ \{\ \omega \in Path(s)\ |\ \omega \models \psi\ \} \sim p$

- For a state s in the DTMC (see [KNP07a] for full definition):
  - $s \models R_{\sim r}\ [\ I^{=k}\ ]\ \Leftrightarrow\ Exp(s, X_{I=k}) \sim r$
  - $s \models R_{\sim r}\ [\ C^{\leq k}\ ]\ \Leftrightarrow\ Exp(s, X_{C\leq k}) \sim r$
  - $s \models R_{\sim r}\ [\ F\ \Phi\ ]\ \Leftrightarrow\ Exp(s, X_{F\Phi}) \sim r$

  where: Exp(s, X) denotes the expectation of the random variable $X : Path(s) \rightarrow \mathbb{R}_{\geq 0}$ with respect to the probability measure Pr_s

Model checking reward properties

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - variant of the method for computing bounded until probabilities
  - solution of recursive equations
- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to computing until probabilities
  - precomputation phase (identify infinite reward states)
  - then reduces to solving a system of linear equation
- For more details, see e.g. [KNP07a]
  - complexity not increased wrt classical PCTL

PRISM

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
- Construction/analysis of probabilistic models…
  - discrete-time Markov chains, continuous-time Markov chains, Markov decision processes, probabilistic timed automata, stochastic multi-player games, …
- Simple but flexible high-level modelling language
  - based on guarded commands; see later…
- Many import/export options, tool connections
  - in: (Bio)PEPA, stochastic π-calculus, DSD, SBML, Petri nets, …
  - out: Matlab, MRMC, INFAMY, PARAM, …

PRISM…

- Model checking for various temporal logics…
  - PCTL, CSL, LTL, PCTL*, rPATL, CTL, …
  - quantitative extensions, costs/rewards, …
- Various efficient model checking engines and techniques
  - symbolic methods (binary decision diagrams and extensions)
  - explicit-state methods (sparse matrices, etc.)
  - statistical model checking (simulation-based approximations)
  - and more: symmetry reduction, quantitative abstraction refinement, fast adaptive uniformisation, ...
- Graphical user interface
  - editors, simulator, experiments, graph plotting
- See: http://www.prismmodelchecker.org/
  - downloads, tutorials, case studies, papers, …

PRISM modelling language

- Simple, textual, state-based modelling language
  - used for all probabilistic models supported by PRISM
  - based on Reactive Modules [AH99]
- Language basics
  - system built as parallel composition of interacting modules
  - state of each module given by finite-ranging variables
  - behaviour of each module specified by guarded commands
    - annotated with probabilities/rates and (optional) action label
  - transitions are associated with state-dependent probabilities
  - interactions between modules through synchronisation

[send] (s=2) -> p_{loss} : (s'=3)&(lost'=lost+1) + (1-p_{loss}) : (s'=4);

action | guard | probability | update | probability | update

Simple example

```
dtmc

module M1
    x : [0..3] init 0;
    [a] x=0 -> (x'=1);
    [b] x=1 -> 0.5 : (x'=2) + 0.5 : (x'=3);
endmodule

module M2
    y : [0..3] init 0;
    [a] y=0 -> (y'=1);
    [b] y=1 -> 0.4 : (y'=2) + 0.6 : (y'=3);
endmodule
```

Costs and rewards

- We augment models with rewards (or, conversely, costs)
  - real-valued quantities assigned to states and/or transitions
  - these can have a wide range of possible interpretations
- Some examples:
  - elapsed time, power consumption, size of message queue, number of messages successfully delivered, net profit, …
- Costs? or rewards?
  - mathematically, no distinction between rewards and costs
  - when interpreted, we assume that it is desirable to minimise costs and to maximise rewards
  - we consistently use the terminology "rewards" regardless
- Properties (see later)
  - reason about expected cumulative/instantaneous reward

Rewards in the PRISM language

```
rewards "total_queue_size"
  true : queue1+queue2;
endrewards
```

(instantaneous, state rewards)

```
rewards "time"
  true : 1;
endrewards
```

(cumulative, state rewards)

```
rewards "dropped"
  [receive] q=q_max : 1;
endrewards
```

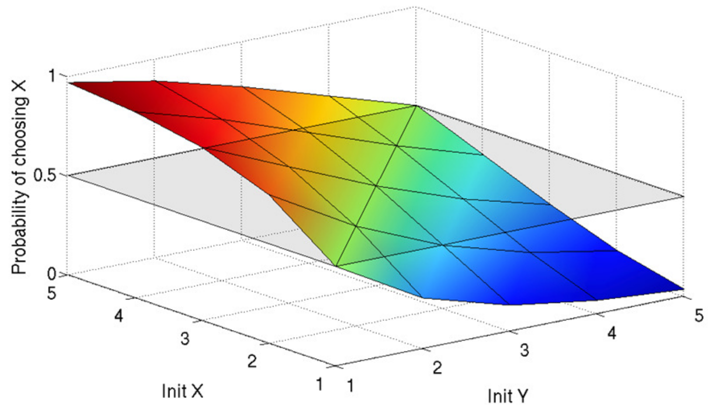
(cumulative, transition rewards)
(q = queue size, q_max = max. queue size, receive = action label)

```
rewards "power"
  sleep=true : 0.25;
  sleep=false : 1.2 * up;
  [wake] true : 3.2;
endrewards
```

(cumulative, state/trans. rewards)
(up = num. operational components, wake = action label)

PRISM – Property specification

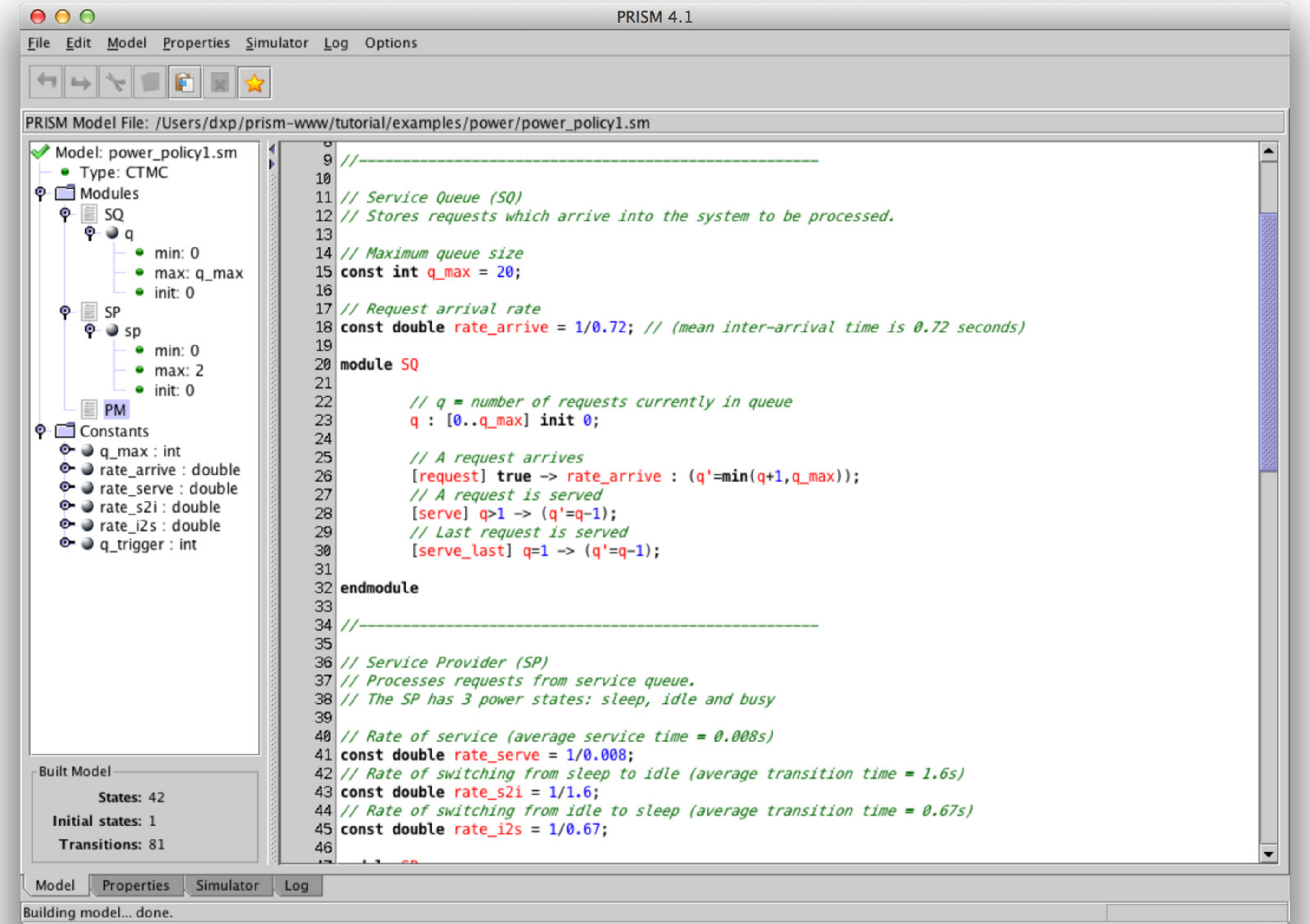
- Temporal logic-based property specification language
  - subsumes PCTL, CSL, probabilistic LTL, PCTL*, …
- Simple examples:
  - $P_{\leq 0.01}$ [F "crash"] – "the probability of a crash is at most 0.01"
  - $S_{>0.999}$ ["up"] – "long-run probability of availability is >0.999"
- Usually focus on quantitative (numerical) properties:
  - $P_{=?}$ [F "crash"]
    "what is the probability of a crash occurring?"
  - then analyse trends in quantitative properties as system parameters vary



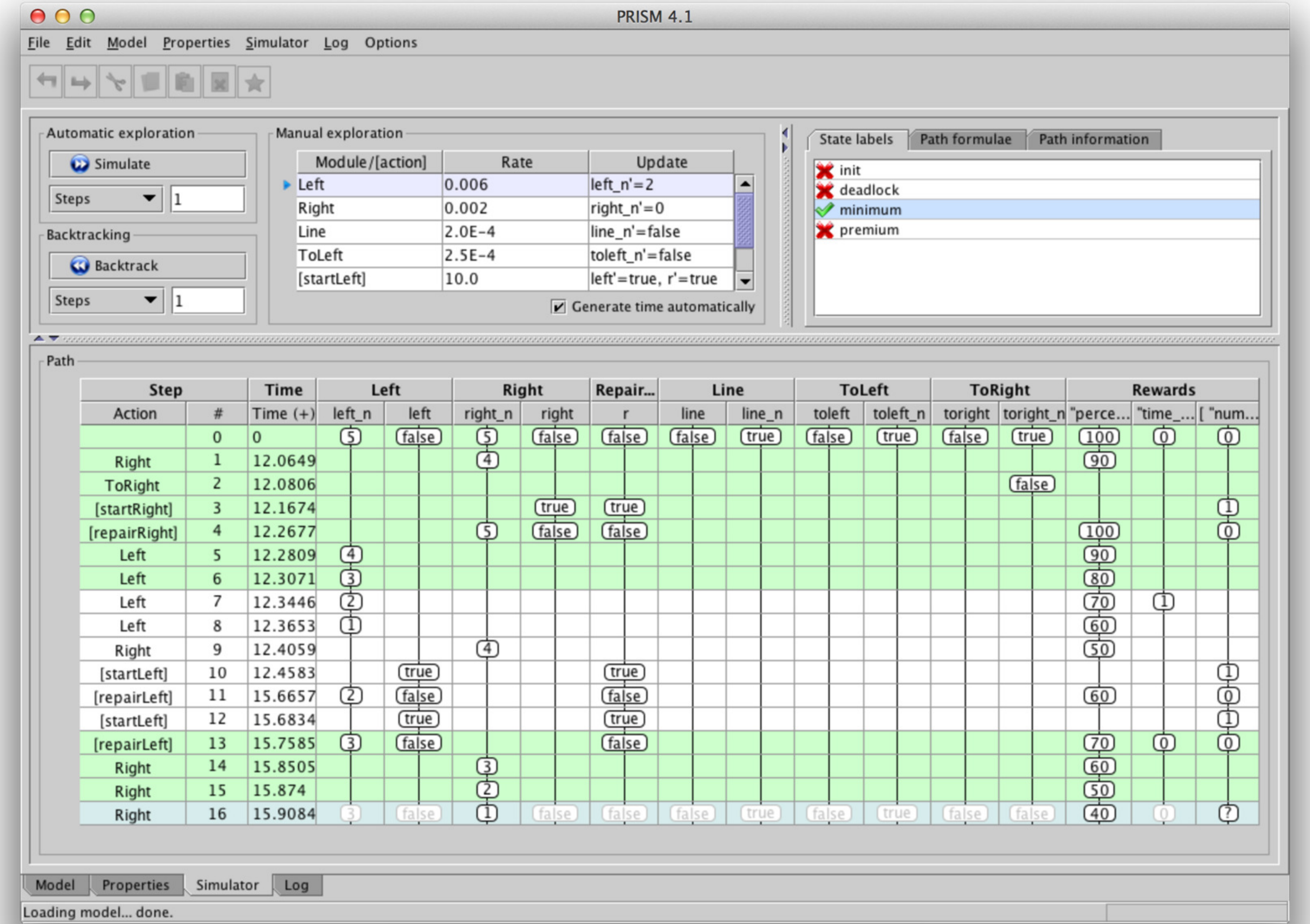
PRISM – Property specification

- Properties can combine numerical + exhaustive aspects
  - $P_{max=?}$ [$F^{\leq 10}$ "fail"] – "worst-case probability of a failure 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-based properties (rewards = costs = prices)
  - $R_{\{\text{"time"}\}=?}$ [F "end"] – "expected algorithm execution time"
  - $R_{\{\text{"energy"}\}max=?}$ [$C^{\leq 7200}$] – "worst-case expected energy consumption during the first 2 hours"
- Properties can be combined with e.g. arithmetic operators
  - e.g. $P_{=?}$ [F $fail_1$] / $P_{=?}$ [F $fail_{any}$] – "conditional failure prob."

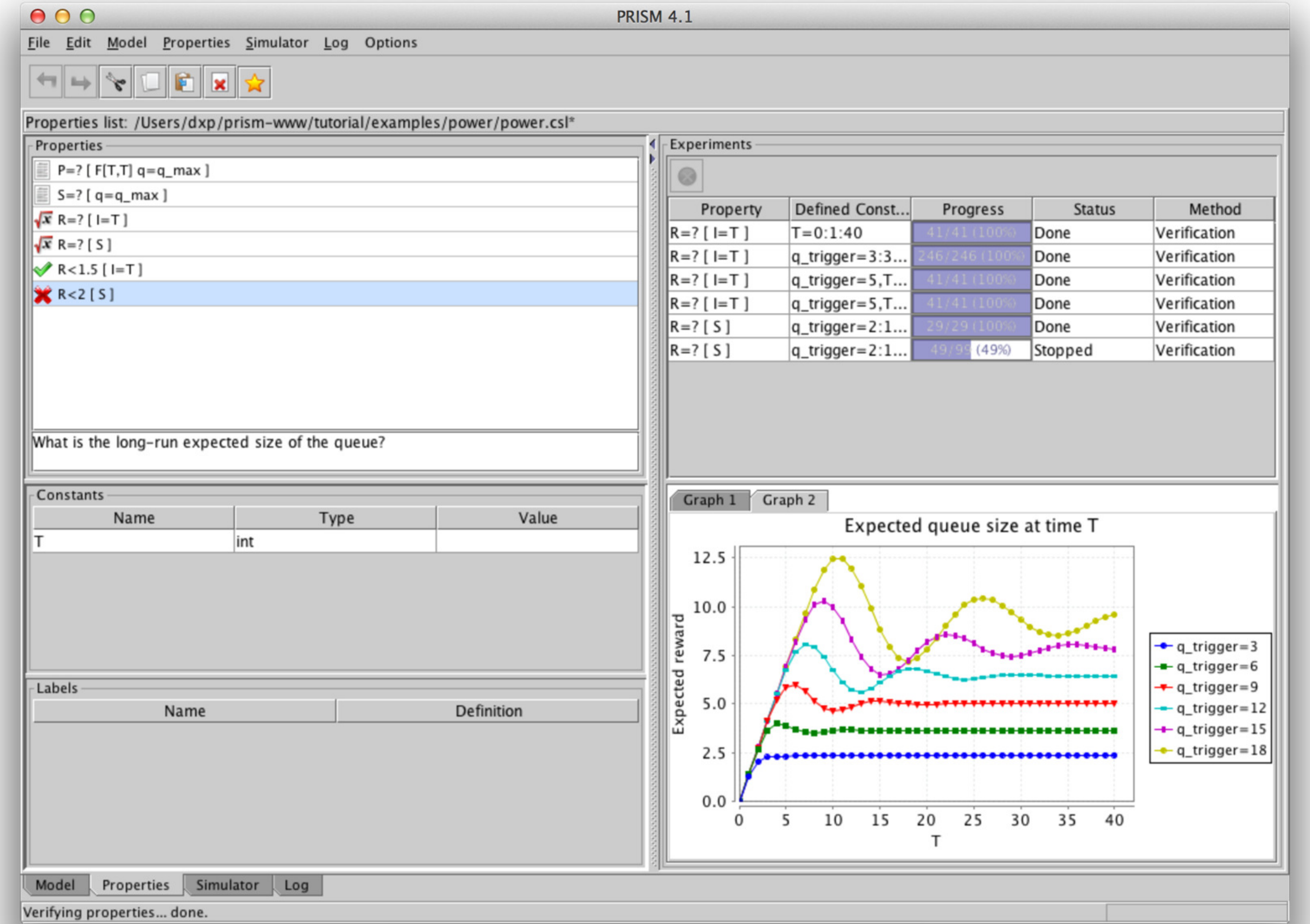
PRISM GUI: Editing a model



PRISM GUI: The Simulator



PRISM GUI: Model checking and graphs



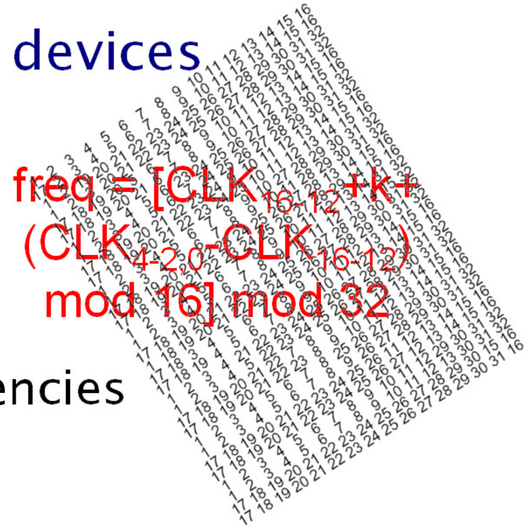
PRISM – Case studies

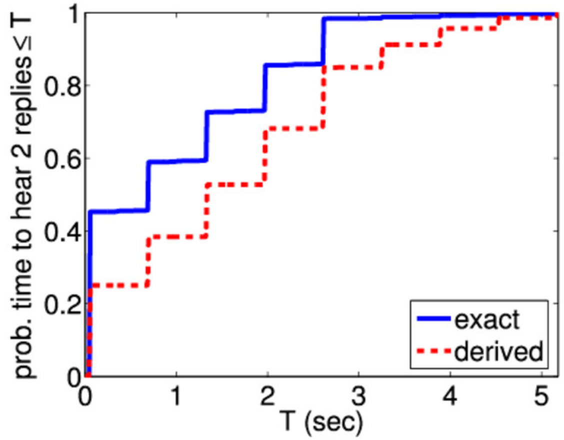
- Randomised distributed algorithms
  - consensus, leader election, self-stabilisation, …
- Randomised communication protocols
  - Bluetooth, FireWire, Zeroconf, 802.11, Zigbee, gossiping, …
- Security protocols/systems
  - contract signing, anonymity, pin cracking, quantum crypto, …
- Biological systems
  - cell signalling pathways, DNA computation, …
- Planning & controller synthesis
  - robotics, dynamic power management, …
- Performance & reliability
  - nanotechnology, cloud computing, manufacturing systems, …

- See: www.prismmodelchecker.org/casestudies

Case study: Bluetooth

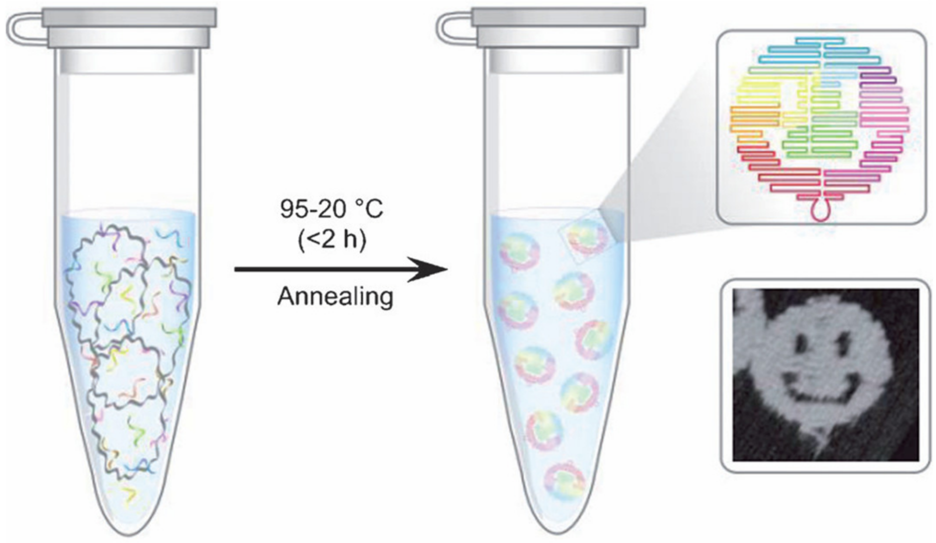
- Device discovery between pair of Bluetooth devices
  - performance essential for this phase
- Complex discovery process
  - two asynchronous 28-bit clocks
  - pseudo-random hopping between 32 frequencies
  - random waiting scheme to avoid collisions
  - 17,179,869,184 initial configurations (too many to sample effectively)
- Probabilistic model checking
  - e.g. "worst-case expected discovery time is at most 5.17s"
  - e.g. "probability discovery time exceeds 6s is always < 0.001"
  - shows weaknesses in simplistic analysis





DNA programming

2nm

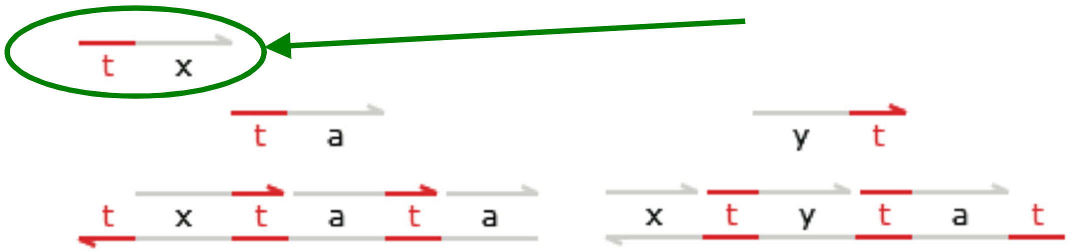


DNA origami

- "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?
  - probability essential!

Case study: DNA programming

- DNA: easily accessible, cheap to synthesise information processing material
- DNA Strand Displacement language, induces CTMC models
  - for designing DNA circuits [Cardelli, Phillips, et al.]
  - accompanying software tool for analysis/simulation
  - now extended to include auto-generation of PRISM models
- Transducer: converts input <t^ x> into output <y t^>



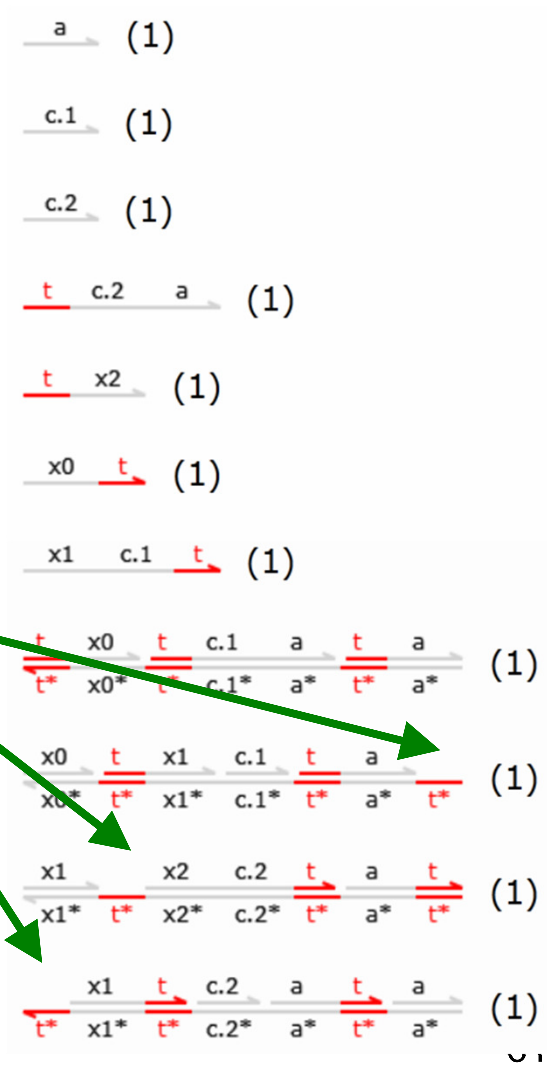
- Formalising correctness: does it finish successfully?…
  - A [G "deadlock" => "all_done"]
  - E [F "all_done"]   (CTL, but probabilistic also…)

Transducer flaw

- PRISM identifies a 5-step trace to the "bad" deadlock state
  - problem caused by "crosstalk" (interference) between DSD species from the two copies of the gates
  - previously found manually [Cardelli'10]
  - detection now fully automated

- Bug is easily fixed
  - (and verified)

reactive gates



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

Summary

- Discrete-time Markov chains (DTMCs)
  - state transition systems + discrete probabilistic choice
  - probability space over paths through a DTMC
- Property specifications
  - probabilistic extensions of temporal logic, e.g. PCTL, LTL
  - also: expected value of costs/rewards
- Model checking algorithms
  - combination of graph-based algorithms, numerical computation, automata constructions
  - also applicable to continuous-time Markov chains via discretisation

- Next: Markov decision processes (MDPs)