Quantitative Probabilistic Verification of Systems

Lecturer: Prof. Marta Kwiatkowska

Computing Laboratory, University of Oxford

ARTIST2 Summer School 2008 in Europe

Autrans, 8-12th September 2008

Overview

- Introduction
  - The context: consider trends in software engineering…
  - Verification via model checking
  - Probabilistic/quantitative verification
- A taste of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension
- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

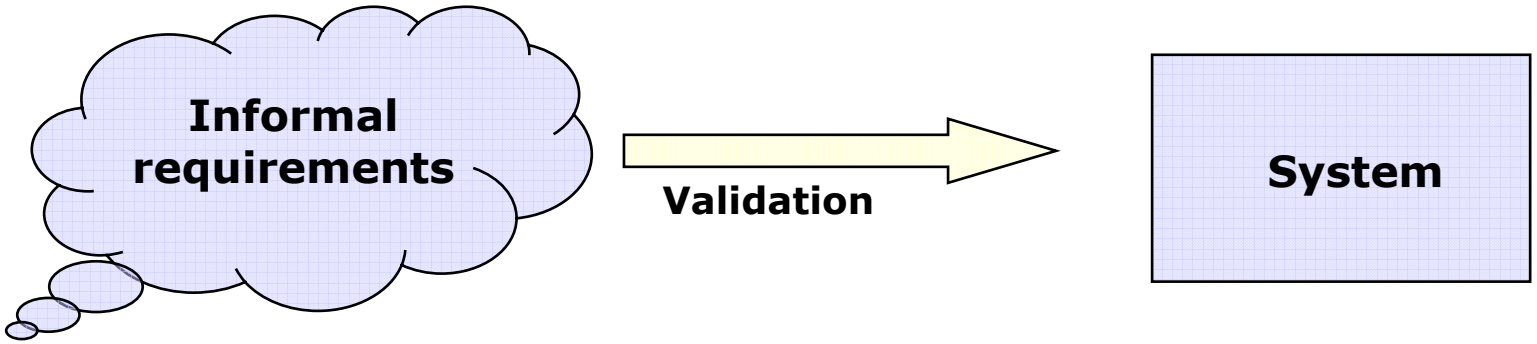
Current trends in software systems…

- Devices, ever smaller
  - Laptops, phones, PDAs, sensors…
- Networking, wireless, wired & global
  - Wireless & Internet everywhere
- Systems/software
  - Self-*
  - Mobile
  - Adaptive
  - Context-aware

- In other words, ubiquitous computing
- Need software engineering techniques to ensure
  - Safety, dependability and performance

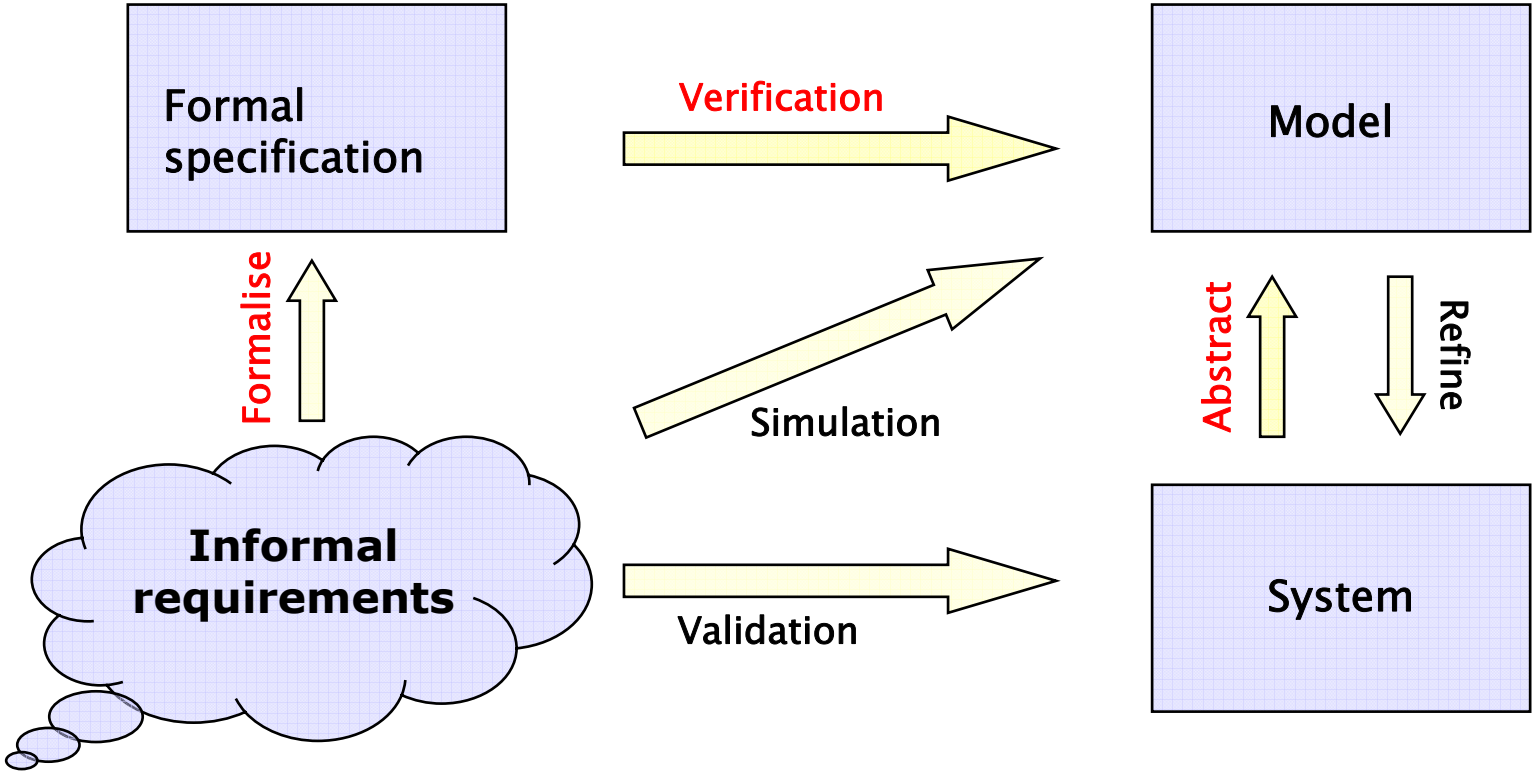
Conventional software engineering

- From requirements to software system
  - Apply design methodologies
  - Code directly in programming language
  - Validation via testing, code walkthroughs



Modern software engineering

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



Why must we verify?

"Testing can only show the presence of errors, not their absence."

"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, computers are without precedent in the cultural history of mankind."

Edsger Wybe Dijkstra

1930–2002

To rule out errors must consider all possible executions – often not feasible mechanically!

But my program works!

- True, there are many successful large-scale complex computer systems…
  - Online banking, electronic commerce
  - Information services, online libraries, business processes
  - Mobile phone networks
- Yet many new potential application domains, far greater complexity, safety-critical context, higher expectations
  - Automotive drive-by-wire
  - Medical sensors: heart rate & blood pressure monitors
  - Intelligent buildings and spaces: WiFi hotspots, sensors
- Learning from mistakes costly…
  - Toyota Prius, BMW 7 series, and many more

Toyota Prius

Drive-by-wire, in car network

100s of embedded components used in modern cars

2005 Toyota Prius hybrid

In May 2005, Toyota recalls about 75,000 cars.
Some Prius drivers have reported sudden stalling or stopping at highway speeds.

According to reports "the stalling problem is due to a software glitch in its sophisticated computer system."

Such problems are becoming more common: BMW 7 series, …
Cost $?

Our approach

- The paradigm: Model, Analyse, Improve Design
- Complementary part of the software engineering process
- Why challenging?
  - Real-world problems
  - State-space explosion will not go away…
  - Need novel algorithms and data structures
- The goals
  - Powerful exhaustive analysis
  - Automation of the analysis

- NB, we assume model built before deployment…
  - No longer realistic for adaptive/context-aware systems
  - Seeking novel dynamic verification techniques…

This talk: emphasis on verification

- Collection of techniques for ensuring that the model satisfies requirements for all behaviours
  - Assume properties expressible in temporal logic
  - Aim for automatic techniques, via model checking

- Unique aspect of this work: inclusion of quantitative, as well as qualitative, requirements:
  - "maximum probability of delivery within time deadline is …"
  - "expected power consumption is …"
  - "expected number of messages lost is …"

- Apply probabilistic/quantitative model checking
  - Probabilistic models, analysis of resource usage

Why probability?

- Probability is fundamental
  - Coordination of distributed systems
  - Quantifying uncertainty, reliability, service
  - Modelling, e.g. molecular reactions

- For quantitative analysis of software and systems
  - To quantify resource usage given a policy
    - "the minimum battery capacity for a given scenario is .."
  - To quantify rate of failures, express Quality of Service
    - "the probability that the message is delivered in 10ms is … "

- In evidence-based, statistical analysis of behaviours
  - To estimate likelihood of faulty runs
  - To quantify trust, anonymity, etc

Information Society Technologies

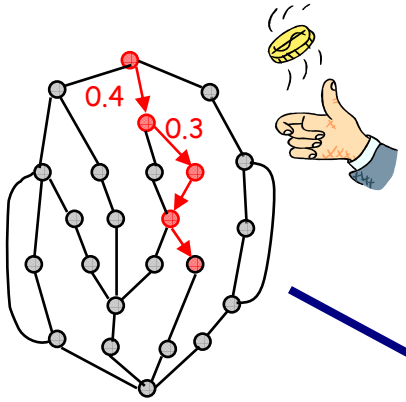
Real-world protocol examples

- Protocols featuring randomisation
  - Randomised back-off schemes
    - E.g. IEEE 802.11 (WiFi) Wireless LAN MAC protocol
  - Random choice of waiting time
    - Bluetooth, device discovery phase
  - Random choice of routes to destination
    - Crowds, anonymity protocol for internet routing
  - Random choice over a set of possible addresses
    - IPv4 dynamic configuration (link-local addressing)
  - and more

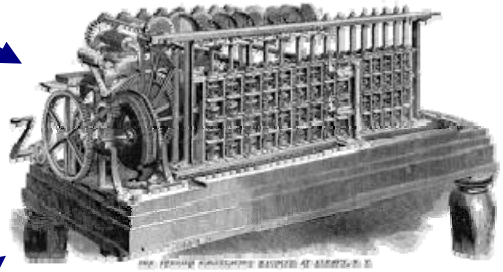
- Continuous probability distribution needed to model network traffic, node mobility, random delays…

Probabilistic model checking…

in a nutshell



Probabilistic model



Probabilistic
Model Checker

or

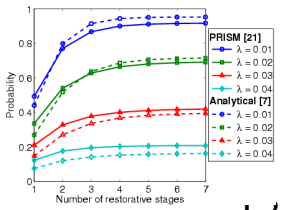
or

The probability

send → $P_{\geq p}$ [F deliver]

Probabilistic temporal
logic specification

State 5: 0.6789
State 6: 0.9789
State 7: 1.0
…
State 12: 0
State 13: 0.1245



Probabilistic model checking inputs

- Models: variants of Markov chains
  - Discrete-Time Markov Chains (DTMCs)
  - Markov Decision Processes (MDPs)
  - Continuous-Time Markov Chains (CTMCs)
  - Probabilistic Time Automata (PTAs)

- Specifications (informally)
  - "probability of delivery within time deadline is …"
  - "expected time to message delivery is …"
  - "expected power consumption is …"

- Specifications (formally)
  - Probabilistic temporal logics (PCTL, CSL, PTCTL)
  - Probability, time, cost/rewards

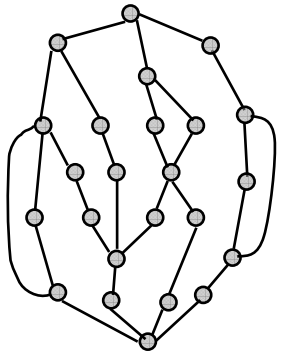
Probabilistic model checking involves…

- Construction of models
  - from a high-level modelling language
  - e.g. probabilistic process algebra

- Implementation of probabilistic model checking algorithms
  - graph-theoretical algorithms, combined with
    - (probabilistic) reachability
  - numerical computation – iterative methods
    - quantitative model checking (plot values for a range of parameters)
    - typically, linear equation or linear optimisation
    - exhaustive, unlike simulation
  - also sampling-based (statistical) for approximate analysis
    - e.g. hypothesis testing based on simulation runs

Model derivation techniques

- Models are typically state-transition systems (automata)
- Manual construction
  - derive a model from description
    - e.g. IEEE standards document
  - express in high-level language, then build
- Automated extraction
  - extract a model from software
    - using e.g. abstract interpretation, slicing, static analysis…
  - build a data structure
- Challenges
  - state space explosion, infinite state systems
  - need to consider augmenting with additional information
    - action labels, state labels, time, probability, rate, etc



Model

Overview

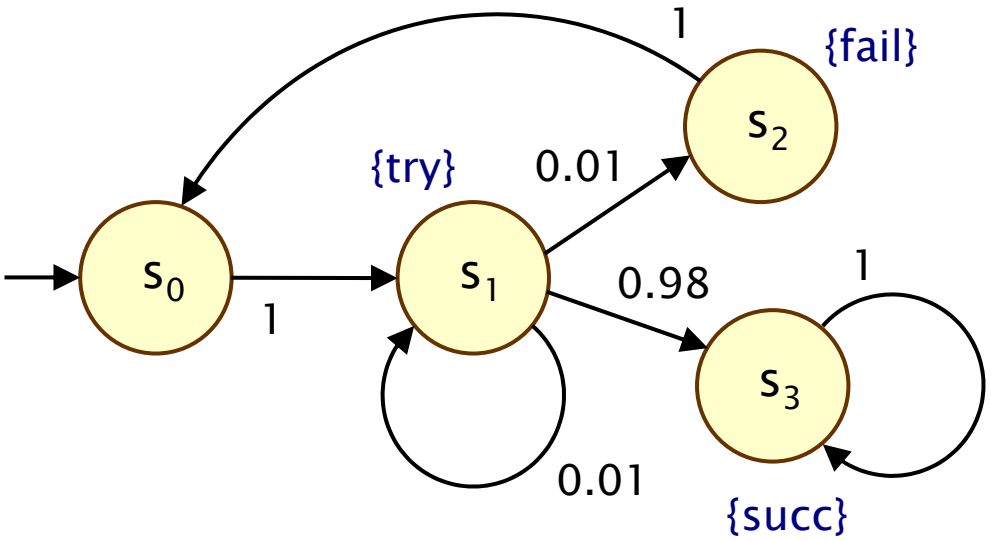
- Introduction
  - The context: trends in software engineering…
  - Verification via model checking
  - Probabilistic/quantitative verification

- A taste of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension

- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

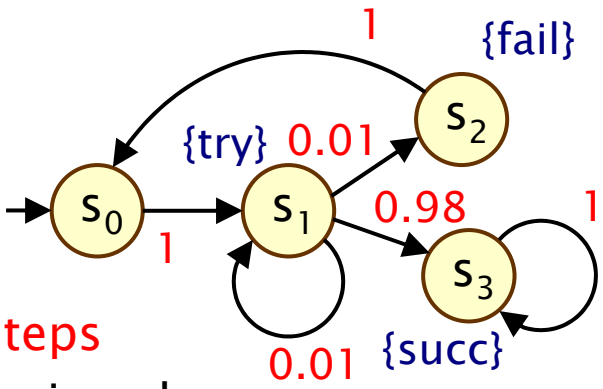
Simple DTMC example

- Modelling a very simple communication protocol
  - after one step, process starts trying to send a message
  - with probability 0.01, channel unready so wait a step
  - with probability 0.98, send message successfully and stop
  - with probability 0.01, message sending fails, restart



Discrete-time Markov chains

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



- Essentially:
  - a state-transition system
  - discrete set of states
  - transitions occur in discrete time-steps
  - probability of making transitions is given by discrete probability distributions

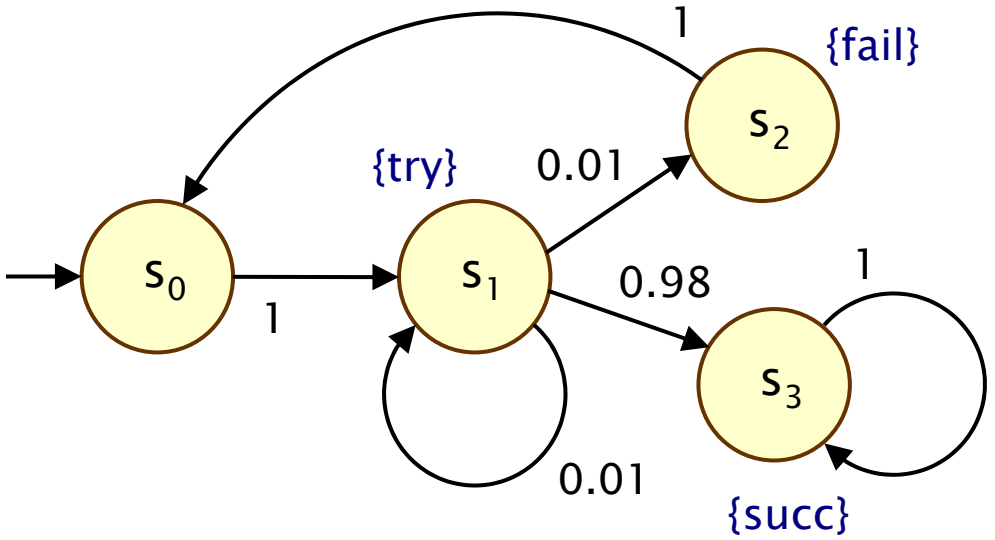
Simple DTMC example

$D = (S, s_{init}, P, L)$

$S = \{s_0, s_1, s_2, s_3\}$
$s_{init} = s_0$

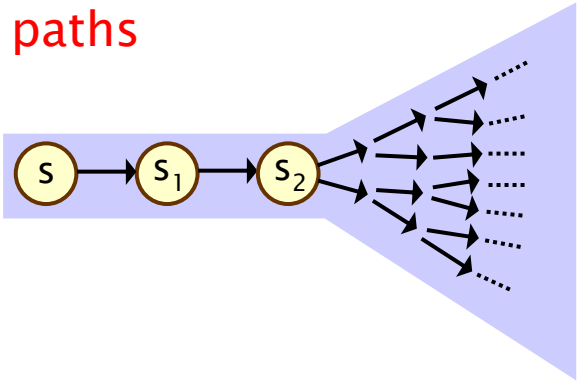
$AP = \{try, fail, succ\}$
$L(s_0) = \emptyset$,
$L(s_1) = \{try\}$,
$L(s_2) = \{fail\}$,
$L(s_3) = \{succ\}$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



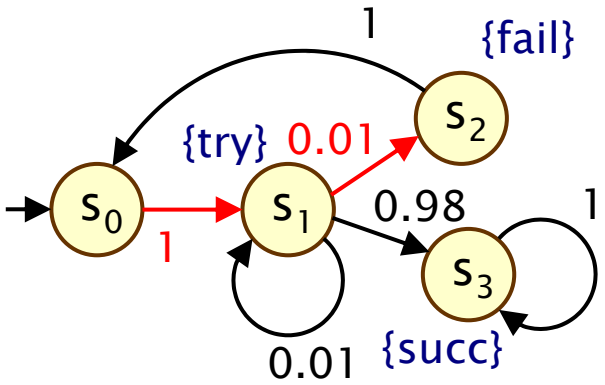
Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0 s_1 s_2 s_3 \ldots$ such that $P(s_i, s_{i+1}) > 0 \ \forall i$
  - 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 $C(\omega)$, for a finite path ω = set of infinite paths with the common finite prefix ω
  - for example: $C(ss_1s_2)$



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
  - $C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots$
  - $Pr_{s0}(C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots)$
    $= P_{s0}(s_0 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_1 s_3) + \ldots$
    $= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots$
    $= 98/99$
    $= 0.9898989898\ldots$

PCTL

- Temporal logic for describing properties of DTMCs
  - PCTL = Probabilistic Computation Tree Logic [HJ94]
  - essentially the same as the logic pCTL
- 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:
  - ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $P_{\sim p}[\psi]$ (state formulas)
  - ψ ::= $X\,\phi$ | $\phi\, U^{\leq k}\, \phi$ | $\phi\, U\, \phi$ (path formulas)

ψ is true with probability $\sim p$

"next"

"bounded until"

"unbounded until"

  - where a is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

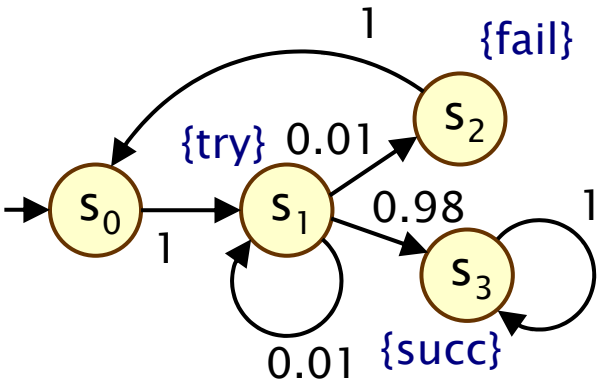
- A PCTL formula is always a state formula
  - path formulas only occur inside the P operator

PCTL semantics for DTMCs

- PCTL formulas interpreted over states of a DTMC
  - $s \models \phi$ 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 $(S, s_{init}, \mathbf{P}, L)$:
  - $s \models a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \models \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \models \phi_1$ and $s \models \phi_2$
  - $s \models \neg\phi \quad \Leftrightarrow \quad s \models \phi$ is false

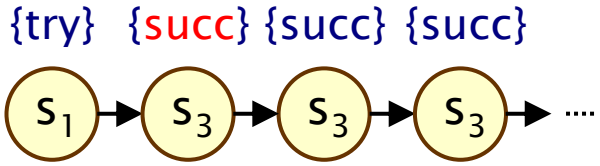
- Examples
  - $s_3 \models$ succ
  - $s_1 \models$ try $\wedge \neg$fail



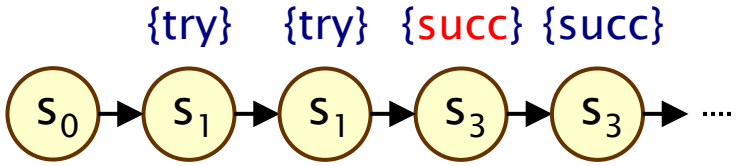
PCTL semantics for DTMCs

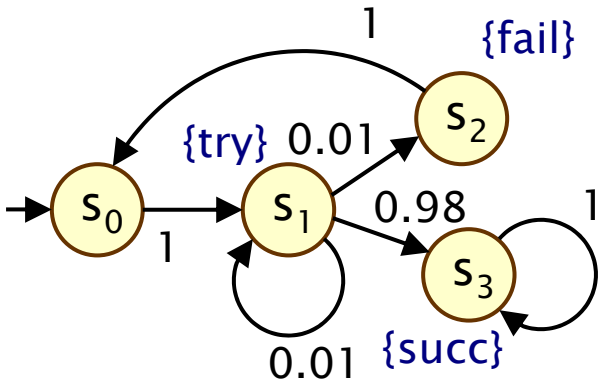
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \models X\, \phi \quad \Leftrightarrow \quad s_1 \models \phi$
  - $\omega \models \phi_1\, U^{\leq k}\, \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \models \phi_2$ and $\forall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

- Some examples of satisfying paths:
  - X succ



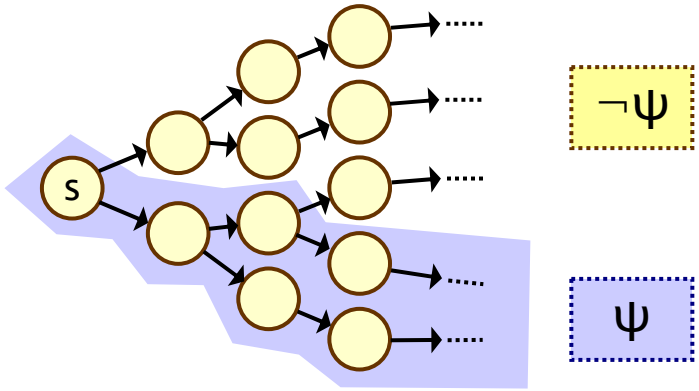
  - ¬fail U succ





PCTL semantics

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



$Prob(s, \psi) \sim p$?

More PCTL

- The "always" path operator
  - $G\ \phi \equiv \neg(F\ \neg\phi) \equiv \neg(\text{true}\ U\ \neg\phi)$ (G = "globally")
  - sometimes written as $\Box\ \phi$ ("box")
  - "ϕ is always true"
  - bounded version: $G^{\leq k}\ \phi \equiv \neg(F^{\leq k}\ \neg\phi)$
  - strictly speaking, G ϕ cannot be derived from the PCTL syntax in this way since there is no negation of path formulas)

- F and G represent two useful classes of properties:
  - reachability: the probability of reaching a state satisfying ϕ
  - i.e. $P_{\sim p}$ [F ϕ]
  - invariance: the probability of ϕ always remaining true
  - i.e. $P_{\sim p}$ [G ϕ]

Qualitative vs. quantitative properties

- P operator of PCTL can be seen as a quantitative analogue of the CTL operators A (for all) and E (there exists)

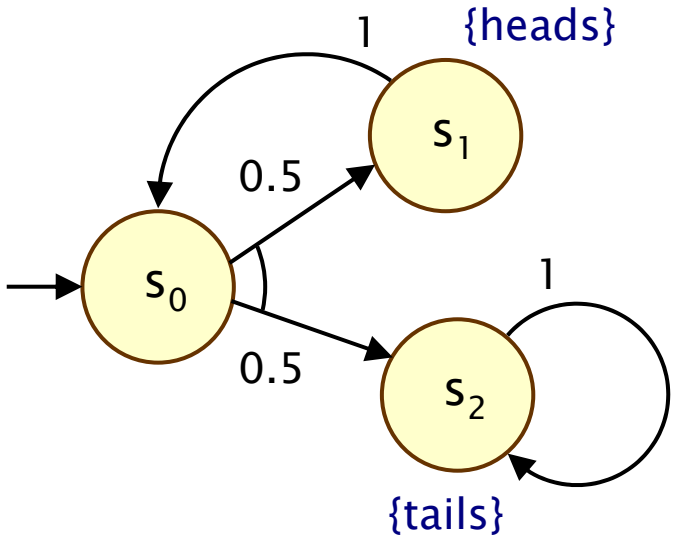
- Qualitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is either 0 or 1
- Quantitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is in the range (0,1)

- $P_{>0}$ [F ϕ] is identical to EF ϕ
  - there exists a finite path to a ϕ-state
- $P_{\geq 1}$ [F ϕ] is (similar to but) weaker than AF ϕ
  - see next slide…

Example: Qualitative/quantitative

- Toss a coin repeatedly until “tails” is thrown

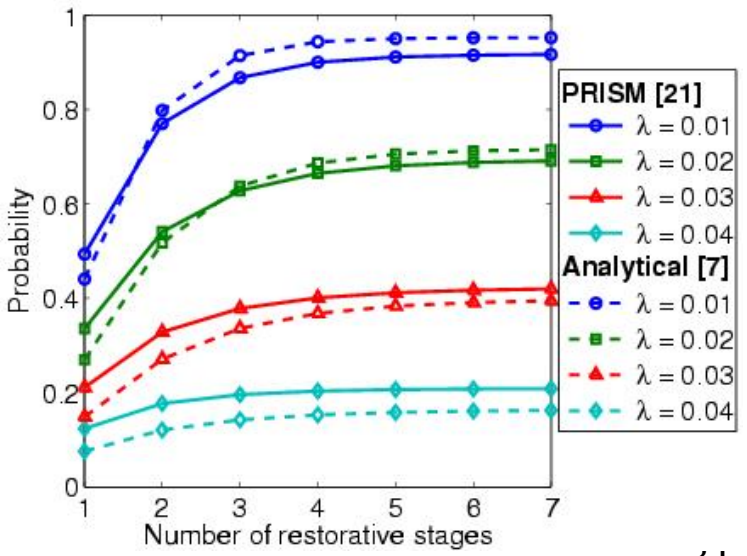
- Is “tails” always eventually thrown?
  - CTL: AF “tails”
  - Result: false
  - Counterexample: $s_0 s_1 s_0 s_1 s_0 s_1 \ldots$



- Does the probability of eventually throwing “tails” equal one?
  - PCTL: $P_{\geq 1}$ [F “tails”]
  - Result: true
  - Infinite path $s_0 s_1 s_0 s_1 s_0 s_1 \ldots$ has zero probability

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?"



Some real PCTL examples

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

- Bluetooth wireless communication protocol
  - $P_{=?}$ [$F^{\leq t}$ reply_count=k]
  - "what is the probability that the sender has received k acknowledgements within t clock-ticks?"

- Security: EGL contract signing protocol
  - $P_{=?}$ [F (pairs_a=0 & pairs_b>0)]
  - "what is the probability that the party B gains an unfair advantage during the execution of the protocol?"

Overview

- Introduction
  - The context: trends in software engineering…
  - Verification via model checking
  - Probabilistic/quantitative verification
- A taste of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension
- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

PCTL model checking for DTMCs

- Algorithm for PCTL model checking [HJ94, BK98]
  - inputs: DTMC $D=(S,s_{init},P,L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \models \phi \}$ = set of states satisfying ϕ

- What does it mean for a DTMC D to satisfy a formula ϕ?
  - sometimes, want to check that $s \models \phi$ $\forall$ $s \in S$, i.e. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{init} \models \phi$, i.e. if $s_{init} \in Sat(\phi)$

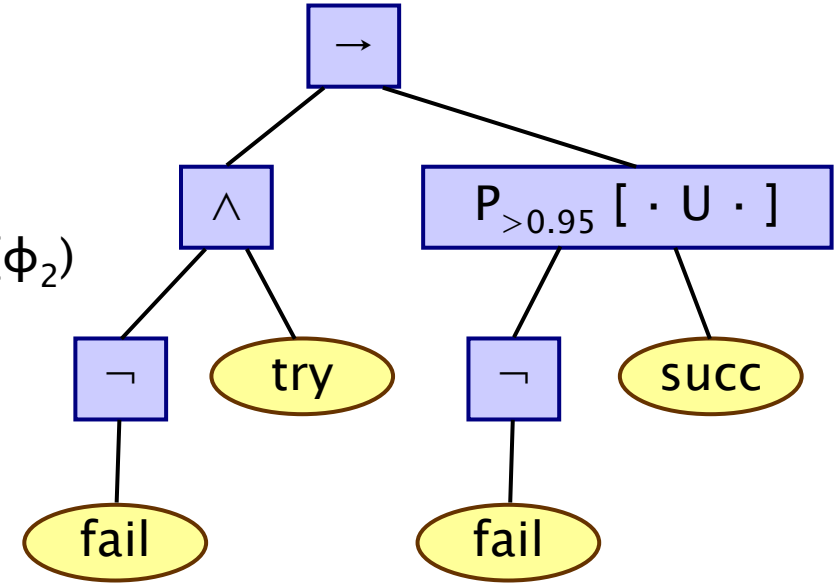
- Sometimes, focus on quantitative results
  - e.g. compute result of P=? [F error]
  - e.g. compute result of P=? [$F^{\leq k}$ error] for $0 \leq k \leq 100$

PCTL model checking for DTMCs

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

- For the non-probabilistic operators:
  - Sat(true) = S
  - Sat(a) = { s ∈ S | a ∈ L(s) }
  - Sat($\neg\phi$) = S \ Sat(ϕ)
  - Sat($\phi_1 \wedge \phi_2$) = Sat(ϕ_1) ∩ Sat(ϕ_2)

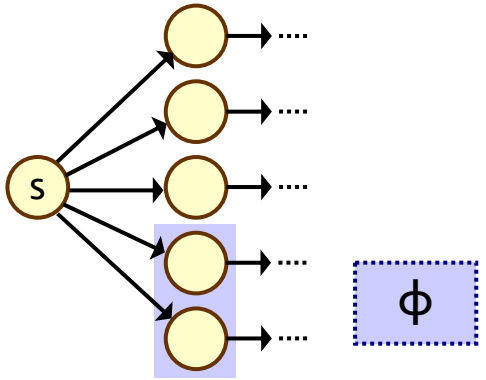
- For the $P_{\sim p}$ [ψ] operator
  - need to compute the probabilities Prob(s, ψ) for all states s ∈ S



PCTL next for DTMCs

- Computation of probabilities for PCTL next operator
  - $Sat(P_{\sim p}[\ X\ \phi\ ]) = \{\ s \in S \mid Prob(s, X\ \phi) \sim p\ \}$
  - need to compute $Prob(s, X\ \phi)$ for all $s \in S$

- Sum outgoing probabilities for transitions to ϕ-states
  - $Prob(s, X\ \phi) = \Sigma_{s' \in Sat(\phi)}\ \mathbf{P}(s,s')$

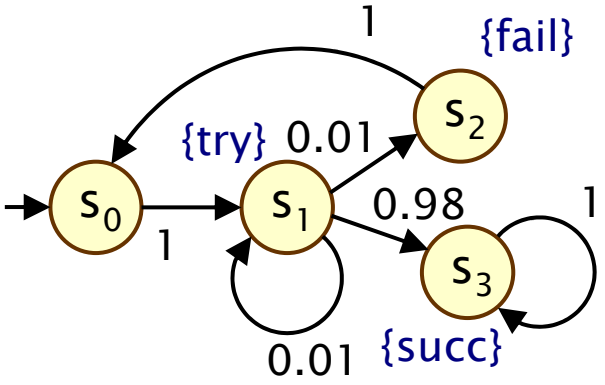


- Compute vector $\underline{Prob}(X\ \phi)$ of probabilities for all states s
  - $\underline{Prob}(X\ \phi) = \mathbf{P} \cdot \underline{\phi}$
  - where $\underline{\phi}$ is a 0-1 vector over S with $\underline{\phi}(s) = 1$ iff $s \models \phi$
  - computation requires a single matrix-vector multiplication

PCTL next – Example

- Model check: $P_{\geq 0.9}$ [X (¬try ∨ succ)]
  - Sat (¬try ∨ succ) = (S \ Sat(try)) ∪ Sat(succ)
    = ({s_0,s_1,s_2,s_3} \ {s_1}) ∪ {s_3} = {s_0,s_2,s_3}
  - Prob(X (¬try ∨ succ)) = **P** · (¬try ∨ succ) = …

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{bmatrix}$$



- Results:
  - Prob(X (¬try ∨ succ)) = [0, 0.99, 1, 1]
  - Sat($P_{\geq 0.9}$ [X (¬try ∨ succ)]) = {s_1, s_2, s_3}

PCTL until for DTMCs

- Computation of probabilities Prob(s, ϕ_1 U ϕ_2) for all s $\in$ S
- We first identify all states where the probability is 1 or 0
  - S^{yes} = Sat($P_{\geq 1}$ [ϕ_1 U ϕ_2])
  - S^{no} = Sat($P_{\leq 0}$ [ϕ_1 U ϕ_2])
- We refer to this as the "precomputation" phase
  - two precomputation algorithms: Prob0 and Prob1
- Important for several reasons
  - reduces the set of states for which probabilities must be computed numerically
  - for $P_{\sim p}[\cdot]$ where p is 0 or 1, no further computation required
  - gives exact results for the states in S^{yes} and S^{no} (no round-off)

Precomputation algorithms

- Prob0 algorithm to compute $S^{no} = Sat(P_{\leq 0}[\ \phi_1\ U\ \phi_2\ ])$:
  - first compute $Sat(P_{>0}[\ \phi_1\ U\ \phi_2\ ])$
  - i.e. find all states which can, with non-zero probability, reach a ϕ_2-state without leaving ϕ_1-states
  - i.e. find all states from which there is a finite path through ϕ_1-states to a ϕ_2-state: simple graph-based computation
  - subtract the resulting set from S

- Prob1 algorithm to compute $S^{yes} = Sat(P_{\geq 1}[\ \phi_1\ U\ \phi_2\ ])$:
  - first compute $Sat(P_{<1}[\ \phi_1\ U\ \phi_2\ ])$, reusing S^{no}
  - this is equivalent to the set of states which have a non-zero probability of reaching S^{no}, passing only through ϕ_1-states
  - again, this is a simple graph-based computation
  - subtract the resulting set from S

PCTL until for DTMCs

- Probabilities $\text{Prob}(s, \phi_1 \, U \, \phi_2)$ can now be obtained as the unique solution of the following set of linear equations:

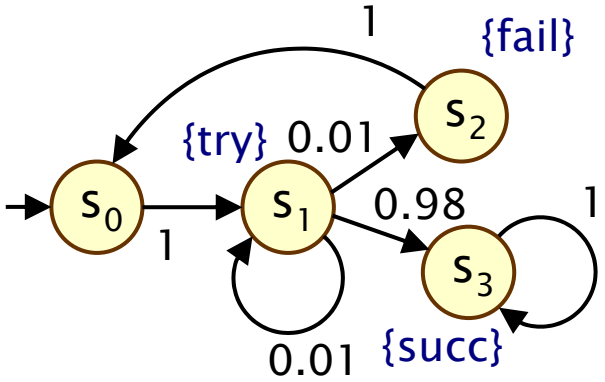
$$\text{Prob}(s, \phi_1 \, U \, \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot \text{Prob}(s', \phi_1 \, U \, \phi_2) & \text{otherwise} \end{cases}$$

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

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

PCTL until – Example

- Model check: $P_{>0.99}$ [try U succ]
  - Sat(try) = $\{s_1\}$, Sat(succ) = $\{s_3\}$
  - S^{no} = Sat($P_{\leq 0}$ [try U succ]) = $\{s_0, s_2\}$
  - S^{yes} = Sat($P_{\geq 1}$ [try U succ]) = $\{s_3\}$
  - $S^{?} = \{s_1\}$
- Linear equation system:
  - $x_0 = 0$
  - $x_1 = 0.01 \cdot x_1 + 0.01 \cdot x_2 + 0.98 \cdot x_3$
  - $x_2 = 0$
  - $x_3 = 1$
- Which yields:
  - $\underline{Prob}$(try U succ) = $\underline{x}$ = [0, 98/99, 0, 1]
  - Sat($P_{>0.99}$ [try U succ]) = $\{s_3\}$



Overview

- Introduction
  - The context: consider software engineering…
  - … and trends in software systems
  - Verification via model checking
  - Probabilistic/quantitative verification

- Overview of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension

- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

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
  - useful to quantify resource usage

- 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

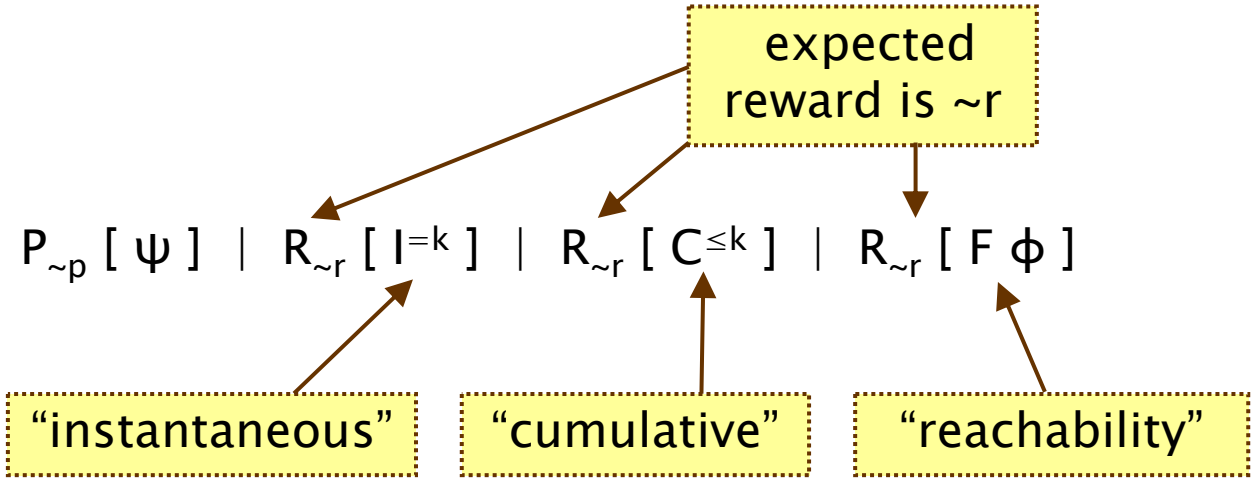
- Properties of DTMCs augmented with rewards
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards
  - formal property specifications will be in an extension of PCTL
- More precisely, we use two distinct classes of property…
- Instantaneous properties
  - 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}, P, L), a reward structure is a pair $(\underline{\rho}, \iota)$
  - $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ is the state reward function (vector)
  - $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ 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
  - ϕ ::= ... | $P_{\sim p}[\psi]$ | $R_{\sim r}[I^{=k}]$ | $R_{\sim r}[C^{\leq k}]$ | $R_{\sim r}[F\,\phi]$



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

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

Types of reward formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - "the expected value of the state reward at time-step k is ~r"
  - e.g. "the expected queue size after exactly 90 seconds"

- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - "the expected reward cumulated up to time-step k is ~r"
  - e.g. "the expected power consumption over one hour"

- Reachability: $R_{\sim r}$ [F ϕ]
  - "the expected reward cumulated before reaching a state satisfying ϕ is ~r"
  - e.g. "the expected time for the algorithm to terminate"

Reward formula semantics

- Formal semantics of the three reward operators:
  - for a state s in the DTMC:
  - $s \models R_{\sim r} [I^{=k}] \Leftrightarrow Exp(s, X_{I=k}) \sim r$
  - $s \models R_{\sim r} [C^{\leq k}] \Leftrightarrow Exp(s, X_{C \leq k}) \sim r$
  - $s \models R_{\sim r} [F \Phi] \Leftrightarrow Exp(s, X_{F\Phi}) \sim r$

  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

Reward formula semantics

- Definition of random variables:
  - for an infinite path $\omega = s_0 s_1 s_2 \ldots$

$$X_{l=k}(\omega) = \underline{\rho}(s_k)$$

$$X_{C \leq k}(\omega) = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{F\phi}(\omega) = \begin{cases} 0 & \text{if } s_0 \in Sat(\phi) \\ \infty & \text{if } s_i \notin Sat(\phi) \text{ for all } i \geq 0 \\ \sum_{i=0}^{k_\phi - 1} \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

  - where $k_\phi = \min\{ j \mid s_j \models \phi \}$

Reward formula model checking

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - reduces to computation of bounded until probabilities
  - solution of recursive equations
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$]
  - variant of the method for computing bounded until probabilities
  - solution of recursive equations
- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to computing until probabilities
  - reduces to solving a system of linear equation

Model checking PCTL summary

- Atomic propositions and logical connectives: trivial

- Probabilistic operator P:
  - X Φ : one matrix-vector multiplications
  - Φ_1 $U^{\leq k}$ Φ_2 : k matrix-vector multiplications
  - Φ_1 U Φ_2 : linear equation system in at most |S| variables

- Expected reward operator R
  - $I^{=k}$: k matrix-vector multiplications
  - $C^{\leq k}$: k iterations of matrix-vector multiplication + summation
  - F Φ : linear equation system in at most |S| variables
  - details for the reward operators are in [KNP07]

Remaining models

- Markov Decision Processes (MDPs)
  - Permit nondeterminism, as well as discrete probability
  - Useful to represent parallel scheduling
  - Only obtain minimum/maximum probability or reward (PCTL)
  - Model checking reduces to linear programming
- Continuous-Time Markov Chains (CTMCs)
  - No nondeterminism: rate of transition, real-valued time
  - Reward extension plus steady-state operator (CSL)
  - Model checking by reduction to embedded (untimed properties) or uniformised (timed) DTMC
- Probabilistic Time Automata (PTAs)
  - Permit nondeterminism, real-valued time and discrete probability
  - Analysis by adaptation of zone or digitisation methods developed for timed automata, logic PTCTL

Overview

- Introduction
  - The context: trends in software engineering…
  - Verification via model checking
  - Probabilistic/quantitative verification
- A taste of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension
- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

The PRISM model checker

- PRISM: Probabilistic symbolic model checker
  - free, open source (GPL), developed since 1999,
  - versions for Linux, Unix, Mac OS X, Windows, 64-bit OSs
- Modelling of:
  - DTMCs, MDPs, CTMCs + costs/rewards
- Verification of:
  - PCTL, CSL + extensions + costs/rewards
- Features:
  - high-level modelling language, wide range of model analysis methods, graphical user interface, efficient implementation
- PRISM website: www.prismmodelchecker.org
  - tool download: binaries, source code (GPL)
  - on-line example repository (40+ case studies)
  - on-line documentation:
    - PRISM manual, tutorial

PRISM application domains

- Communication and multimedia protocols
  - Bluetooth device discovery, IEEE 1394 FireWire, Zeroconf
  - IEEE 802.3 IEEE 802.11 WiFi, IEEE 802.15.4 Zigbee
- Security systems/protocols
  - contract signing, anonymity, fair exchange, PIN cracking schemes, negotiation frameworks, quantum cryptography
- Randomised distributed algorithms
  - Byzantine agreement, consensus, self-stabilisation, leader election, mutual exclusion
- Analysis of performance/reliability
  - dynamic power management systems, manufacturing/control systems, nanotechnology (NAND multiplexing), groupware protocols
- Biological processes
  - biochemical networks (signalling pathways)

PRISM technicalities

- Functionality
  - Modelling language: probabilistic reactive modules
  - Construction of models: DTMCs/CTMCs, MDPs
  - Probabilistic temporal logics: PCTL and CSL
  - Extension with costs/rewards, expectation operator
- Underlying computation involves
  - Reachability, qualitative model checking, BDD-based
  - Linear equation system solution – Jacobi, Gauss-Seidel, ...
  - Uniformisation (CTMCs)
  - Dynamic programming (MDPs)
  - Explicit and symbolic (MTBDDs, etc.)
  - Simulation and sampling-based (DTMCs, CTMCs)
- Similar tools
  - ETMCC/MRMC, PROBMELA, Vesta, Rapture, Ymer, SMART, Mobius

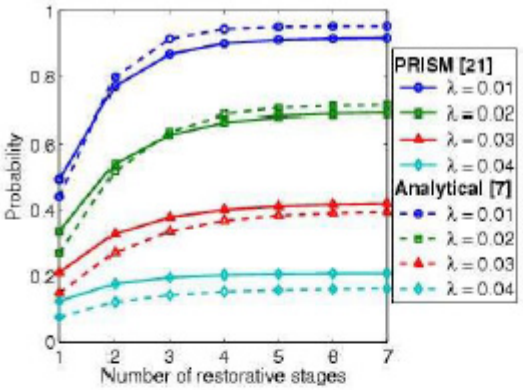
Efficiency: Symbolic techniques

- Symbolic probabilistic model checking
  - data structures based on binary decision diagrams (BDDs)
  - compact storage: exploit model structure and regularity
- PRISM: multiple computation engines
  - MTBDDs (BDD extension): storage/analysis of very large models, numerical computation can blow up
  - sparse matrices: fastest solution for smaller models ($<10^6$ states), prohibitive memory consumption for larger models
  - hybrid: combine MTBDD storage with explicit storage, ten-fold increase in analysable model size ($\sim 10^7$ states)
- Approximate model checking
  - sampling using Monte Carlo discrete-event simulation
- Parallelisation of model checking
  - distribution of storage/computation across multi-processor machines, networked clusters, grids

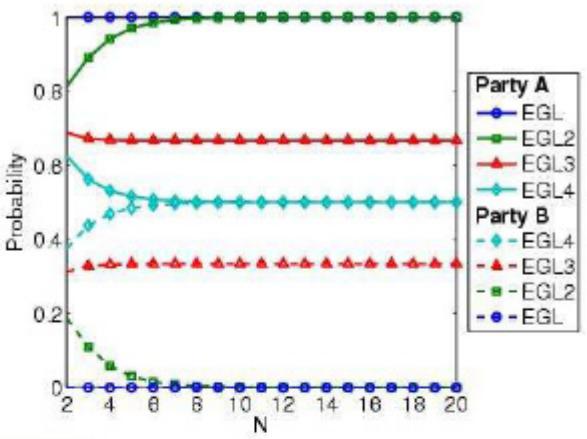
Advantages of quantitative verification

- Analysis combining "quantitative" and "exhaustive" aspects
- Best/worst-case scenarios, not possible with simulation
  - Computing values for a range of states
    - "maximum expected run-time over all possible initial configurations"
  - All possible resolutions of nondeterminism (MDPs)
    - "maximum expected number of bits revealed under any eavesdropping strategy?"
- Identifying trends and anomalies
  - Counterexamples (error traces)
    - widely used in non-probabilistic model checking
    - work in progress for probabilistic model checking...
  - Experiments: ranges of model/property parameters
    - e.g. P=? [true U≤T error] for N=1..5, T=1..100, for N model parameter and T a time bound
    - identify patterns, trends, anomalies in quantitative results

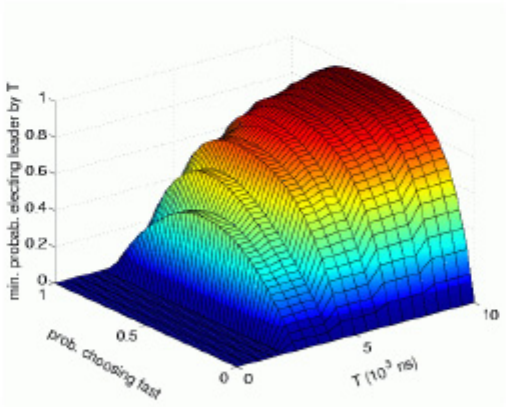
Results: probabilistic analysis



Probability that 10% of gate outputs are erroneous for varying gate failure rates and numbers of stages

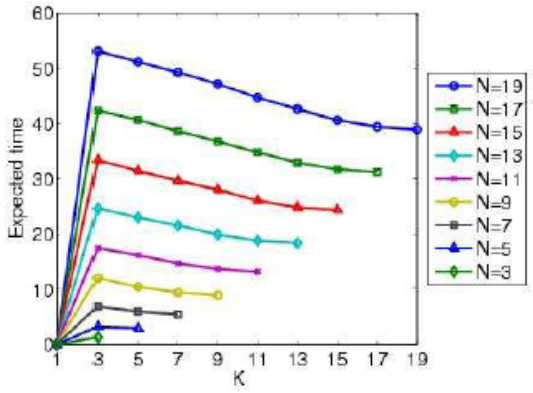


Probability that parties gain unfair advantage for varying numbers of secret packets sent

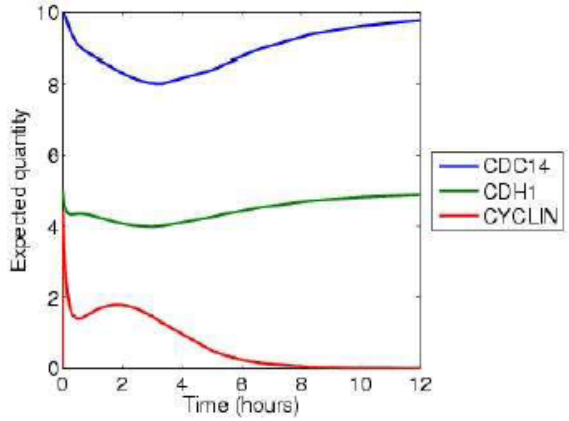


Optimum probability of leader election by time T for various coin biases

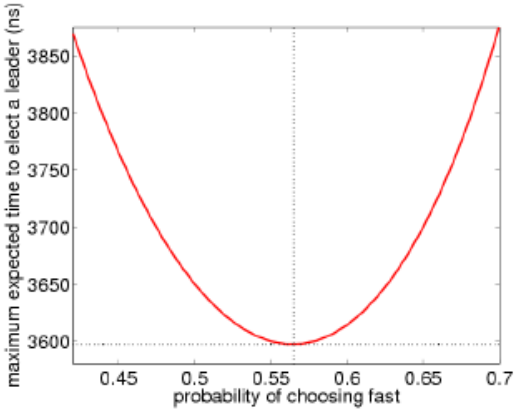
Results: quantitative analysis



Worst-case expected number of steps to stabilise for initial configurations with K tokens amongst N processes



Expected reactant concentrations over the first 12 hours



Maximum expected time for leader election for various coin biases

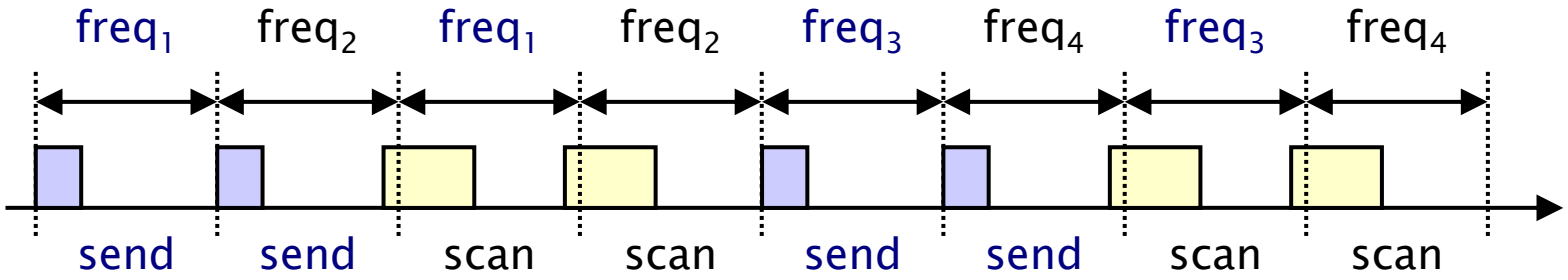
Overview

- Introduction
  - The context: trends in software engineering…
  - Verification via model checking
  - Probabilistic/quantitative verification
- A taste of probabilistic/quantitative verification
  - Discrete-time Markov chain model
  - PCTL model checking
  - Reward extension
- Quantitative verification in practice
  - The PRISM model checker
  - An example of analysis Bluetooth device discovery
  - Challenges and future directions

Bluetooth device discovery

- Bluetooth: short-range low-power wireless protocol
  - widely available in phones, PDAs, laptops, ...
  - personal area networks (PANs)
  - open standard, specification freely available
- Uses frequency hopping scheme
  - to avoid interference (uses unregulated 2.4GHz band)
  - pseudo-random selection over 32 of 79 frequencies
- Network formation
  - piconets (1 master, up to 7 slaves)
  - self-configuring: devices discover themselves

Frequency hopping

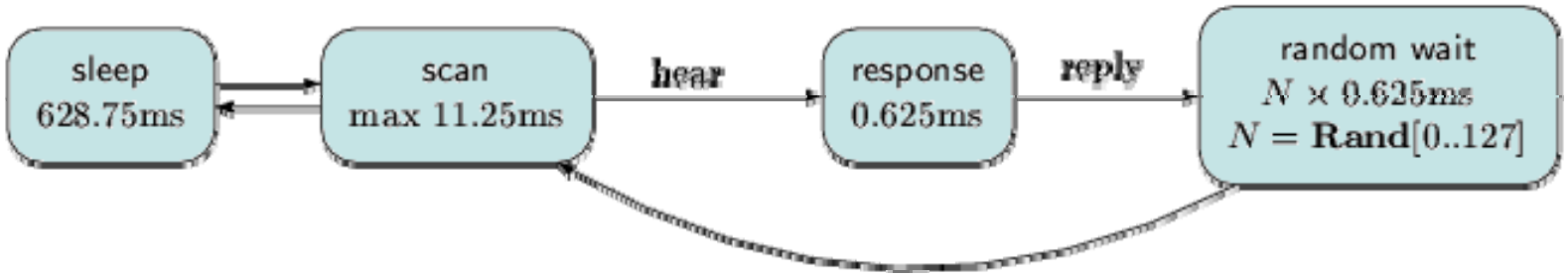


- 28 bit free-running clock CLK, ticks every 312.5μs
- Master broadcasts inquiry packets on two consecutive frequencies, then listens on the same two (plus margin)
- Potential slaves want to be discovered, scan for messages
- Frequency sequence determined by formula, dependent on bits of clock CLK (k defined on next slide):

$$freq = [CLK_{16-12} + k + (CLK_{4-2,0} - CLK_{16-12}) \bmod 16] \bmod 32$$

Slave (receiver) behaviour

- Listens (scans) on frequencies for inquiry packets
  - must listen on right frequency at right time
  - cycles through frequency sequence at much slower speed (every 1.28s)



- On hearing packet, pause, send reply and then wait for a random delay before listening for subsequent packets
  - avoid repeated collisions with other slaves

Bluetooth modelling

- Very complex interaction
  - genuine randomness, probabilistic modelling essential
  - devices make contact only if listen on the right frequency at the right time!
  - sleep/scan periods unbreakable, much longer than listening
  - cannot omit sub-activities, otherwise model is oversimplified
- Huge model, even for one sender and one receiver!
  - initial configurations dependent on 28 bit clock
  - cannot fix start state of receiver, clock value could be arbitrary
- But is a realistic future ubiquitous computing scenario!

Bluetooth – PRISM model

- Modelling in PRISM [DKNP06]
  - model one sender and one receiver
  - synchronous (clock speed defined by Bluetooth spec)
  - randomised behaviour – use DTMC
  - model at lowest-level (one clock-tick = one transition)
  - use real values for delays, etc, from Bluetooth spec

- Modelling challenges
  - complex interaction between sender/receiver
  - combination of short/long time-scales – cannot scale down
  - sender/receiver not initially synchronised, huge number of possible initial configurations (17,179,869,184)

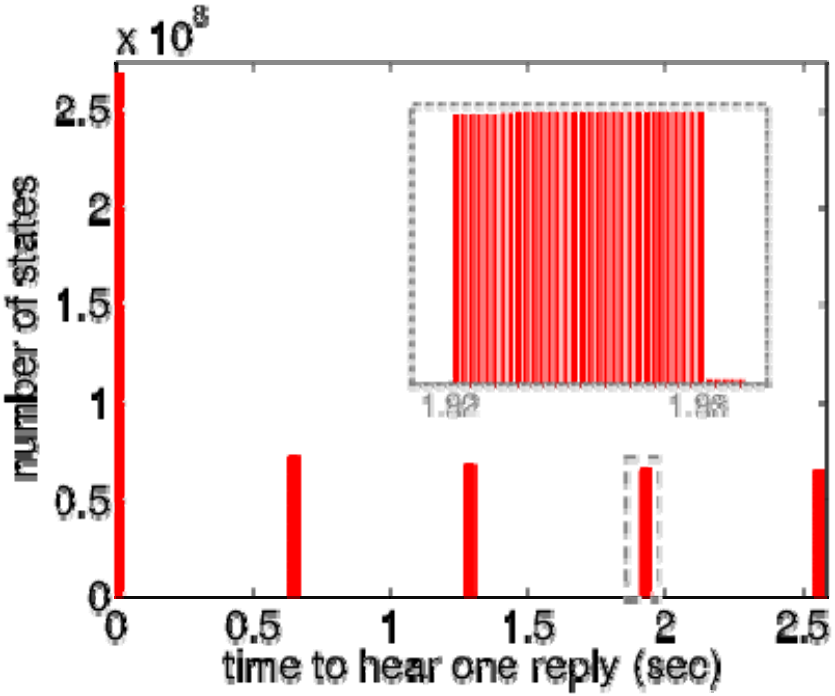
Bluetooth – Results

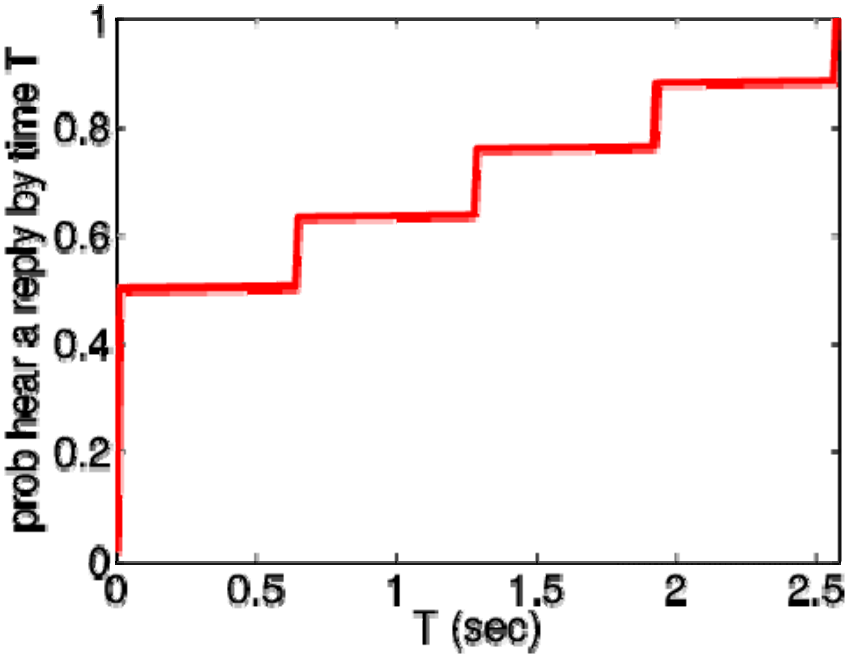
- Huge DTMC!
  - initially, model checking infeasible
  - partition into 32 scenarios, i.e. 32 separate DTMCs
  - on average, approx. 3.4 x 10^9 states, 536,870,912 initial
  - can be built/analysed with PRISM's MTBDD engine

- Property model checked:
  - $R_{=?}$ [F replies=K {"init"}{max}]
  - "worst-case (maximum) expected time to hear K replies, over all possible initial configurations"
  - also: how many initial states for each possible expected time
  - and: cumulative distribution function assuming equal probability for each initial state

Bluetooth – Time to hear 1 reply

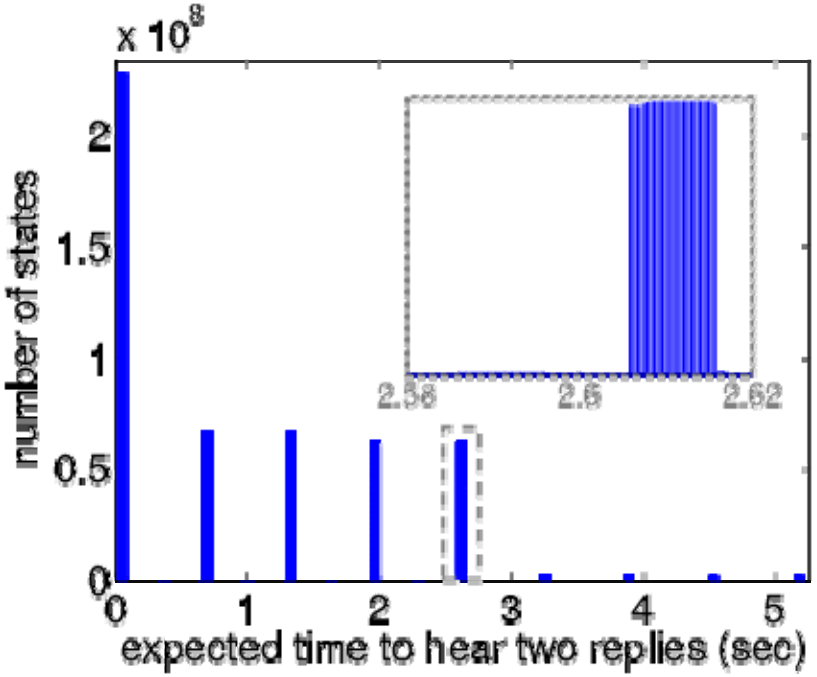
- Worst-case expected time = 2.5716s
  - in 921,600 possible initial states
- Best-case expected time = 635μs

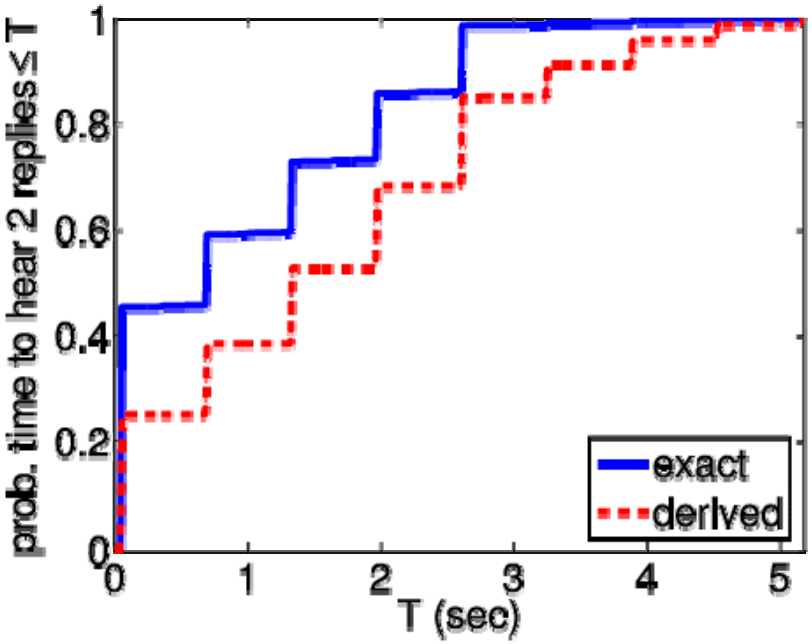




Bluetooth – Time to hear 2 replies

- Worst-case expected time = 5.177s
  - in 444 possible initial states
- Compare actual CDF with derived version which assumes times to reply to first/second messages are independent





Bluetooth – Results

- Other results (see [DKNP06])
  - compare versions 1.2 and 1.1 of Bluetooth, confirm 1.1 slower
  - power consumption analysis (using rewards)

- Conclusions
  - successful analysis of complex real-life model, actual parameters from standard
  - exhaustive analysis: best-/worst-case values
    - can pinpoint scenarios which give rise to them
    - not possible with simulation approaches
  - model still relatively simple
    - consider multiple receivers?
    - combine with simulation?

Challenges

- Scalability remains to be a challenge
  - Develop compositional methods, e.g. assume-guarantee
  - Improve approximate, sampling based methods
  - Automate abstraction/refinement
- Adaptive, context-aware systems?
  - Combine with machine learning?
- Combine with software model checking to enable model extraction from real software, e.g. verify instruction
  - x := random(1..65024); {ZeroConf}
- More powerful analysis
  - Multi-criteria optimisation, [EKVY07]
  - More expressive logics, e.g. LTL
  - Continuous distributions, not just discrete

Ongoing research areas

- Game-based abstraction refinement for probabilistic systems
  - Extension of the CEGAR approach to probabilistic systems/software
- Software verification for context-awareness
  - Extend existing techniques to cater for context adaptation
- Analysis techniques for adaptive systems
  - Power management, online methods
- Quantitative approaches for component-based system architectures (new)
  - Focus on non-functional properties, such as performance

Further information

- Case studies, statistics, group publications
- Download, version 3.2 (12,000+ downloads)
- Unix/Linux, Windows, Apple platforms
- Publications by others
- Courses that feature PRISM…

- See PRISM web page
  www. prismmodelchecker.org