UNIVERSITY OF OXFORD

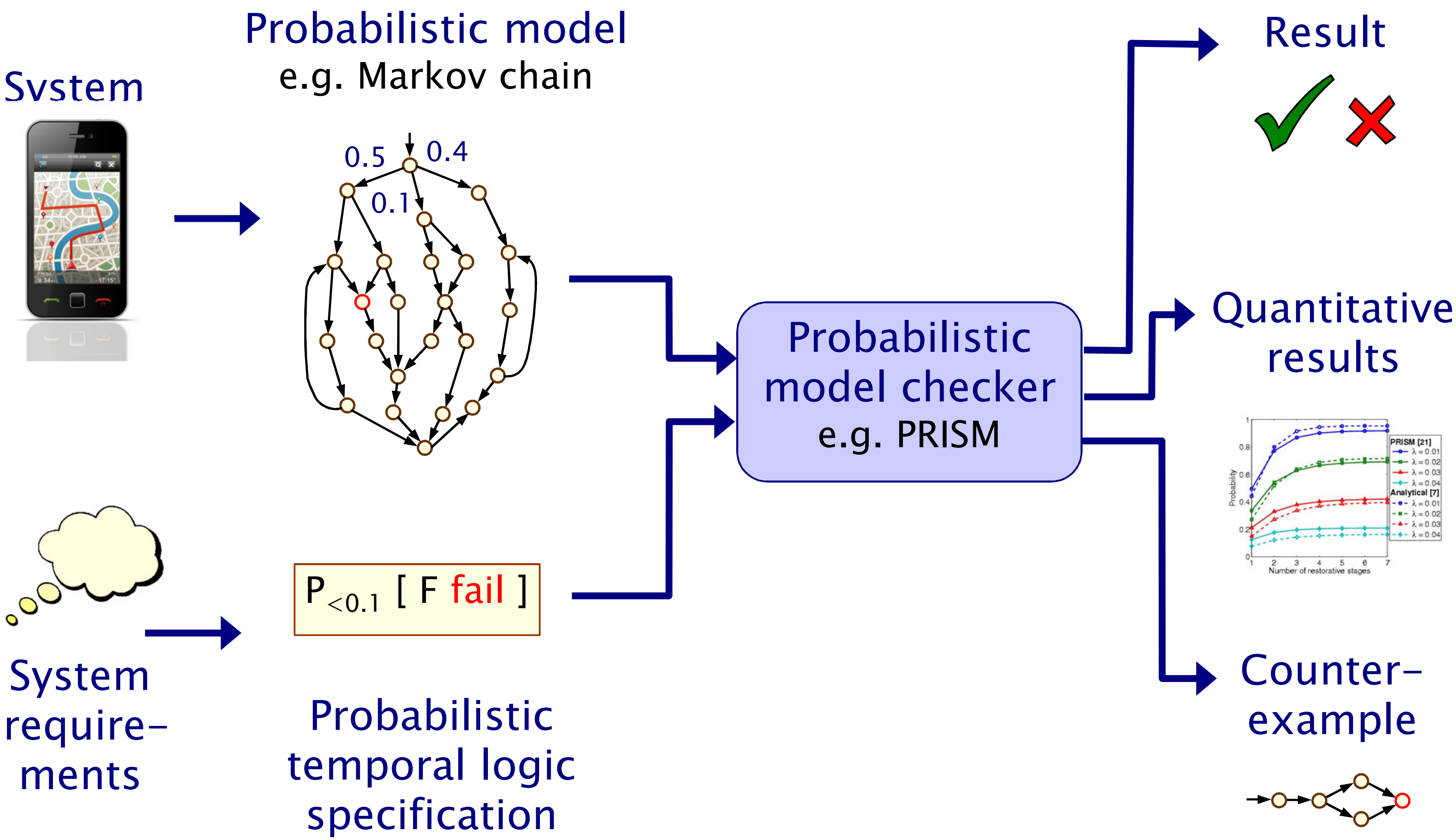
Probabilistic verification and synthesis

Marta Kwiatkowska

Department of Computer Science, University of Oxford

KTH, Stockholm, August 2015

Probabilistic model checking



Lecture plan

- Course slides and lab session
  - http://www.prismmodelchecker.org/courses/kth15/

- 5 sessions: lectures 9-12noon, labs 2.30-5pm
  - 1 – Introduction
  - 2 – Discrete time Markov chains (DTMCs)
  - 3 – Markov decision processes (MDPs)
  - 4 – LTL model checking for DTMCs/MDPs
  - 5 – Probabilistic timed automata (PTAs)

- For extended versions of this material
  - and an accompanying list of references
  - see: http://www.prismmodelchecker.org/lectures/

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)

Part 2

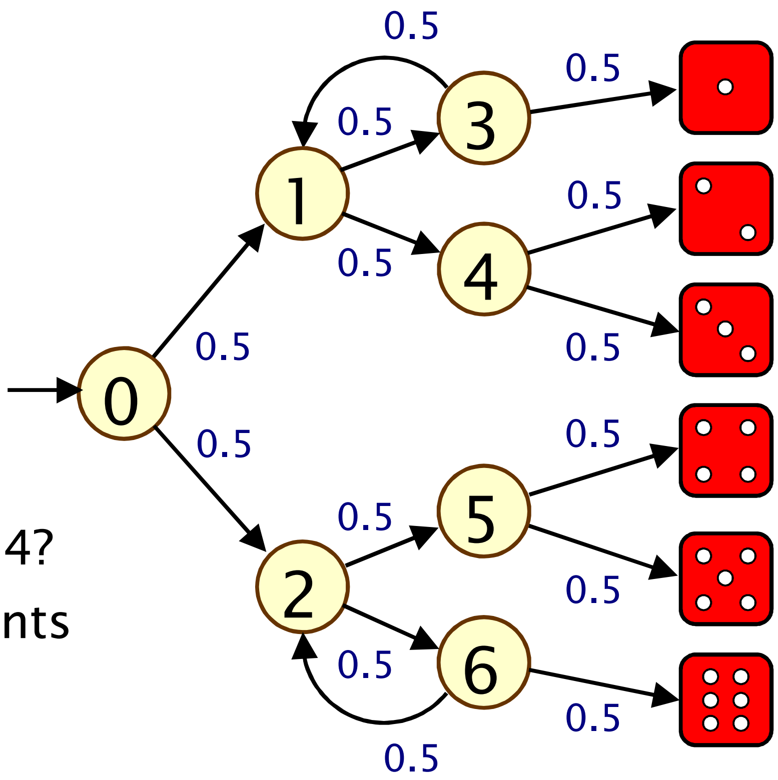
Discrete-time Markov chains

Overview (Part 1)

- Probability basics
- Model checking for discrete-time Markov chains (DTMCs)
  - DTMCs: definition, paths & probability spaces
  - PCTL model checking
  - Costs and rewards
- A glimpse of model checking for continuous-time Markov chains (CTMCs)
- Case studies
  - Bluetooth device discovery
  - Power management
- PRISM developments: parametric model checking
- Summary

Probability example

- Modelling a 6-sided die using a fair coin
  - algorithm due to Knuth/Yao:
  - start at 0, toss a coin
  - upper branch when H
  - lower branch when T
  - repeat until value chosen

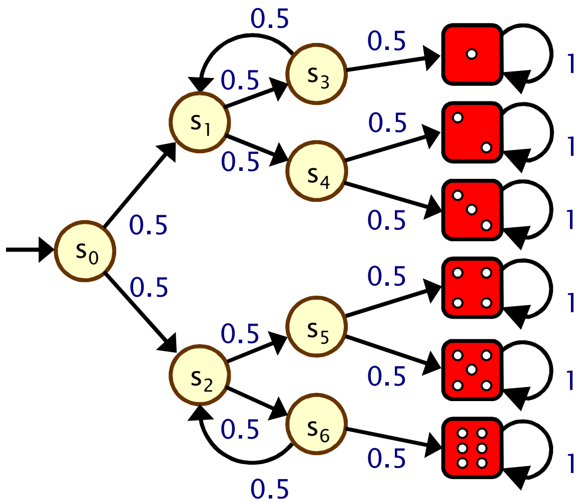


- Is this algorithm correct?
  - e.g. probability of obtaining a 4?
  - obtain as disjoint union of events
  - THH, TTTHH, TTTTTHH, …
  - Pr("eventually 4")

    $= (1/2)^3 + (1/2)^5 + (1/2)^7 + \ldots = 1/6$

Example…

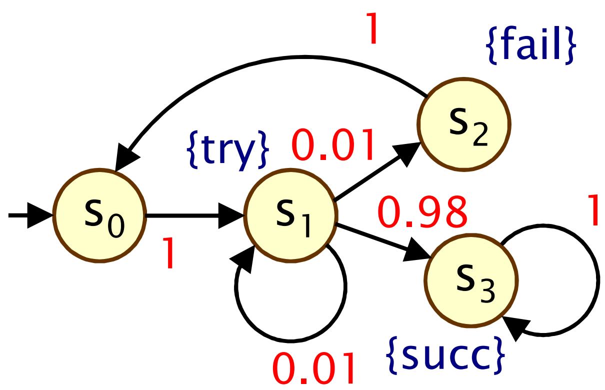
- Other properties?
  - "what is the probability of termination?"
- e.g. efficiency?
  - "what is the probability of needing more than 4 coin tosses?"
  - "on average, how many coin tosses are needed?"



- Probabilistic model checking provides a framework for these kinds of properties…
  - modelling languages
  - property specification languages
  - model checking algorithms, techniques and tools

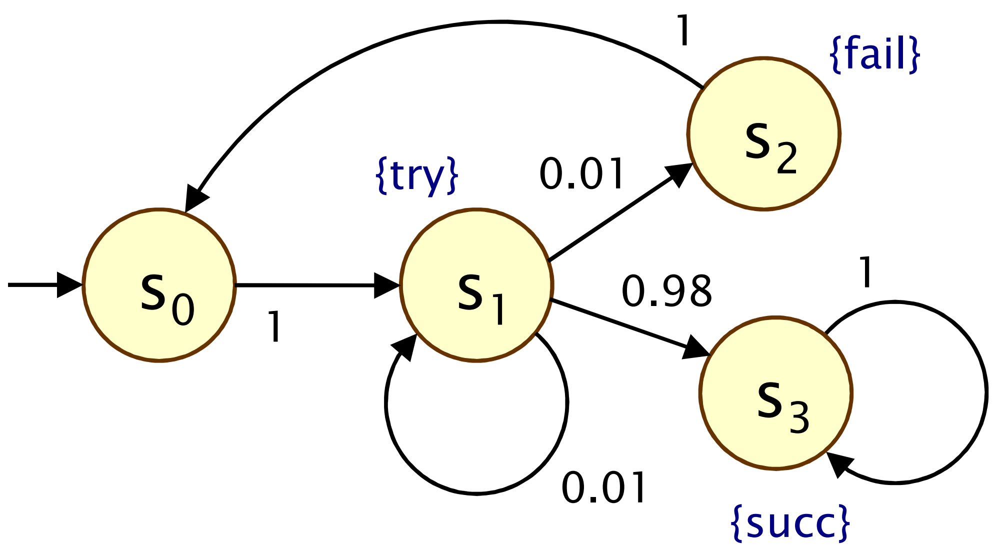
Discrete-time Markov chains

- 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



Simple DTMC example

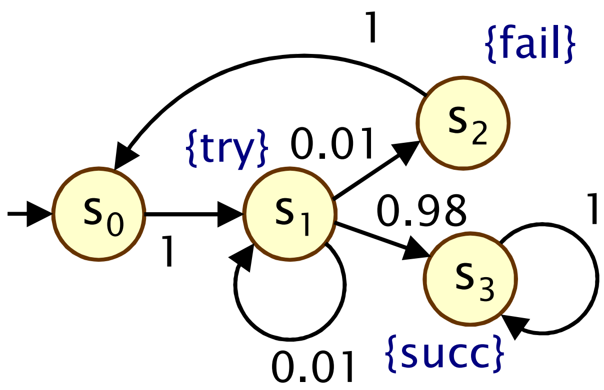
- Modelling a very simple communication protocol
  - after one step, process starts trying to send a message
  - with probability 0.01, channel unready so wait a step
  - with probability 0.98, send message successfully and stop
  - with probability 0.01, message sending fails, restart



Discrete-time Markov chains

- 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



Simple DTMC example

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

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

$s_{init} = s_0$

$AP = \{try, fail, succ\}$

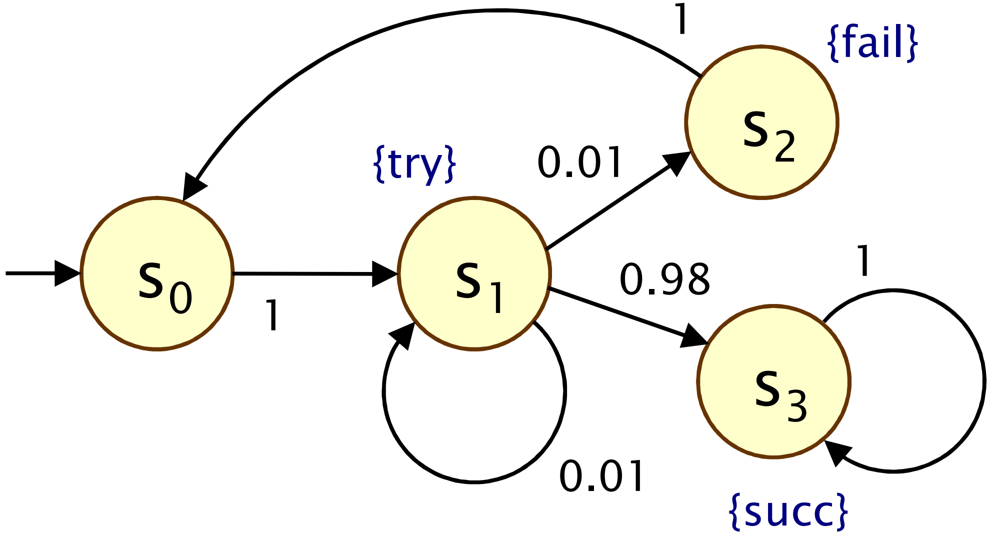
$L(s_0) = \emptyset$,

$L(s_1) = \{try\}$,

$L(s_2) = \{fail\}$,

$L(s_3) = \{succ\}$

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



Some more terminology

- P is a stochastic matrix, meaning it satisifes:
  - $P(s,s') \in [0,1]$ for all $s,s' \in S$ and $\Sigma_{s' \in S} P(s,s') = 1$ for all $s \in S$
- A sub-stochastic matrix satisfies:
  - $P(s,s') \in [0,1]$ for all $s,s' \in S$ and $\Sigma_{s' \in S} P(s,s') \leq 1$ for all $s \in S$
- An absorbing state is a state s for which:
  - $P(s,s) = 1$ and $P(s,s') = 0$ for all $s \neq s'$
  - the transition from s to itself is sometimes called a self-loop
- Note: Since we assume P is stochastic…
  - every state has at least one outgoing transition
  - i.e. no deadlocks (in model checking terminology)

DTMCs: An alternative definition

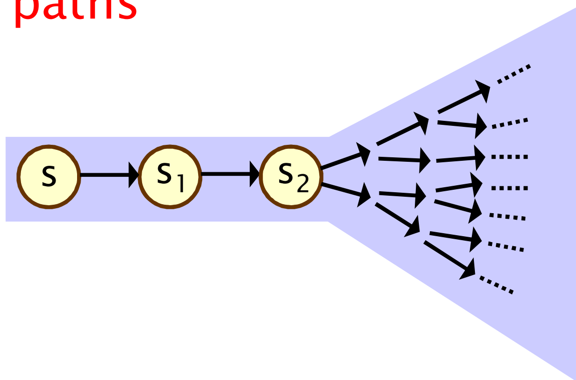
- Alternative definition… a DTMC is:
  - a family of random variables { X(k) | k=0,1,2,… }
  - where X(k) are observations at discrete time-steps
  - i.e. X(k) is the state of the system at time-step k
  - which satisfies…
- The Markov property ("memorylessness")
  - $Pr(X(k)=s_k \mid X(k-1)=s_{k-1}, \ldots , X(0)=s_0)$
    $= Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$
  - for a given current state, future states are independent of past
- This allows us to adopt the "state-based" view presented so far (which is better suited to this context)

Other assumptions made here

- We consider time-homogenous DTMCs
  - transition probabilities are independent of time
  - $P(s_{k-1},s_k) = Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$
  - otherwise: time-inhomogenous
- We will (mostly) assume that the state space S is finite
  - in general, S can be any countable set
- Initial state $s_{init} \in S$ can be generalised…
  - to an initial probability distribution $s_{init} : S \rightarrow [0,1]$
- Focus on path-based properties
  - rather than steady-state

Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0s_1s_2s_3\ldots$ 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(ω), for a finite path ω = set of infinite paths with the common finite prefix ω
  - for example: $C(ss_1s_2)$



Probability spaces

- Let Ω be an arbitrary non-empty set
- A σ-algebra (or σ-field) on Ω is a family Σ of subsets of Ω closed under complementation and countable union, i.e.:
  - if $A \in \Sigma$, the complement $\Omega \setminus A$ is in Σ
  - if $A_i \in \Sigma$ for $i \in \mathbb{N}$, the union $\cup_i A_i$ is in Σ
  - the empty set $\varnothing$ is in Σ
- Theorem: For any family F of subsets of Ω, there exists a unique smallest σ-algebra on Ω containing F
- Probability space (Ω, Σ, Pr)
  - Ω is the sample space
  - Σ is the set of events: σ-algebra on Ω
  - $Pr : \Sigma \rightarrow [0,1]$ is the probability measure:

    $Pr(\Omega) = 1$ and $Pr(\cup_i A_i) = \Sigma_i\, Pr(A_i)$ for countable disjoint A_i

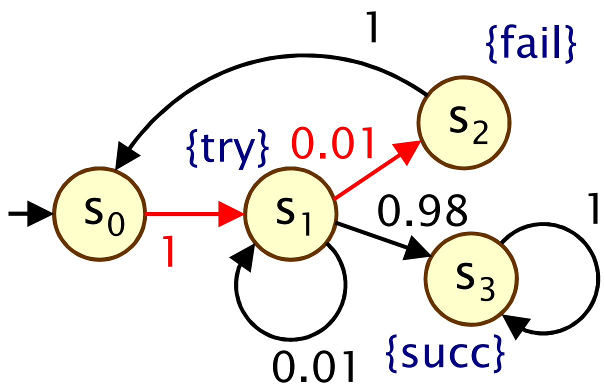
Probability space over paths

- Sample space Ω = Path(s)
  set of infinite paths with initial state s
- Event set $\Sigma_{Path(s)}$
  - the cylinder set $C(\omega) = \{ \omega' \in Path(s) \mid \omega \text{ is prefix of } \omega' \}$
  - $\Sigma_{Path(s)}$ 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
  - $C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots$
  - $Pr_{s0}(\ C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots\ )$
    $= P_{s0}(s_0 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_1 s_3) + \ldots$
    $= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots$
    $= 0.9898989898\ldots$
    $= 98/99$

Reachability

- Key property: probabilistic reachability
  - probability of a path reaching a state in some target set T ⊆ S
  - e.g. “probability of the algorithm terminating successfully?”
  - e.g. “probability that an error occurs during execution?”

- Dual of reachability: invariance
  - probability of remaining within some class of states
  - Pr(“remain in set of states T”) = 1 - Pr(“reach set S\T”)
  - e.g. “probability that an error never occurs”

- We will also consider other variants of reachability
  - time-bounded, constrained (“until”), …

Reachability probabilities

- Formally: ProbReach(s, T) = Pr_s(Reach(s, T))
  - where Reach(s, T) = { $s_0 s_1 s_2 \ldots \in$ Path(s) | s_i in T for some i }

- Is Reach(s, T) measurable for any $T \subseteq S$? Yes…
  - Reach(s, T) is the union of all basic cylinders Cyl($s_0 s_1 \ldots s_n$) where $s_0 s_1 \ldots s_n$ in $Reach_{fin}$(s, T)
  - $Reach_{fin}$(s, T) contains all finite paths $s_0 s_1 \ldots s_n$ such that: s_0=s, $s_0,\ldots,s_{n-1} \notin T$, $s_n \in T$ (reaches T first time)
  - set of such finite paths $s_0 s_1 \ldots s_n$ is countable

- Probability
  - in fact, the above is a disjoint union
  - so probability obtained by simply summing…

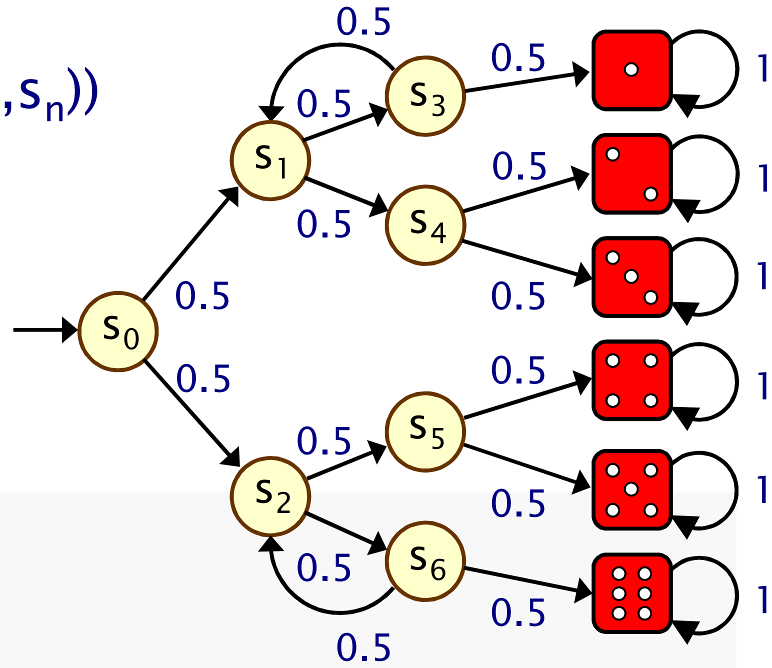
Computing reachability probabilities

- Compute as (infinite) sum…

- $\Sigma_{s_0,\ldots,s_n \in \text{Reachfin}(s,\,T)} \Pr_{s_0}(\text{Cyl}(s_0,\ldots,s_n))$

  $= \Sigma_{s_0,\ldots,s_n \in \text{Reachfin}(s,\,T)} \mathbf{P}(s_0,\ldots,s_n)$

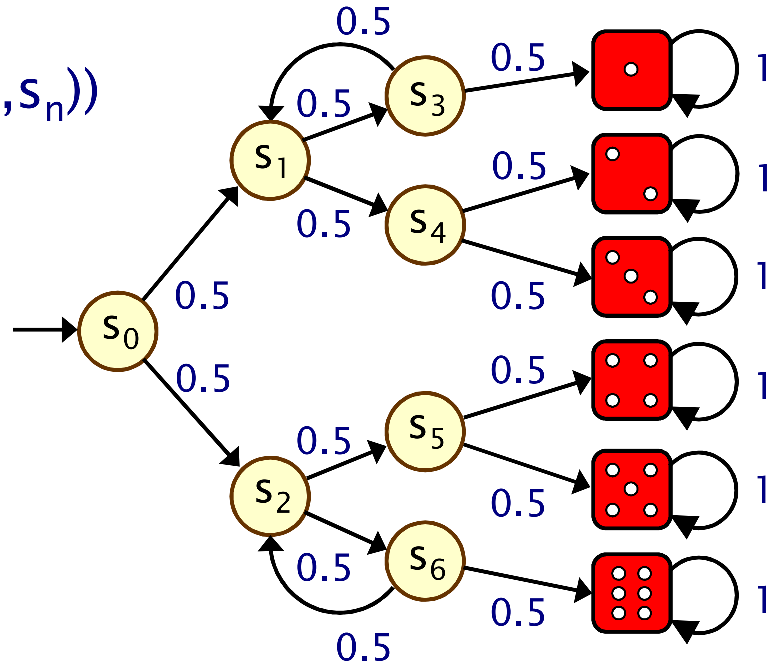
- Example:
  - ProbReach(s_0, {4})



Computing reachability probabilities

- Compute as (infinite) sum…
- $\Sigma_{s_0,\ldots,s_n \in \text{Reachfin}(s,\,T)} \Pr_{s0}(\text{Cyl}(s_0,\ldots,s_n))$

  $= \Sigma_{s_0,\ldots,s_n \in \text{Reachfin}(s,\,T)} \mathbf{P}(s_0,\ldots,s_n)$



- Example:
  - ProbReach(s_0, {4})
  - = $\Pr_{s0}$(Reach(s_0, {4}))
  - Finite path fragments:
  - $s_0(s_2s_6)^n s_2s_5 4$ for $n \geq 0$
  - $\mathbf{P}_{s0}(s_0s_2s_5 4) + \mathbf{P}_{s0}(s_0s_2s_6s_2s_5 4) + \mathbf{P}_{s0}(s_0s_2s_6s_2s_6s_2s_5 4) + \ldots$

  $= (1/2)^3 + (1/2)^5 + (1/2)^7 + \ldots = 1/6$

Computing reachability probabilities

- Alternative: derive a linear equation system
  - solve for all states simultaneously
  - i.e. compute vector ProbReach(T)

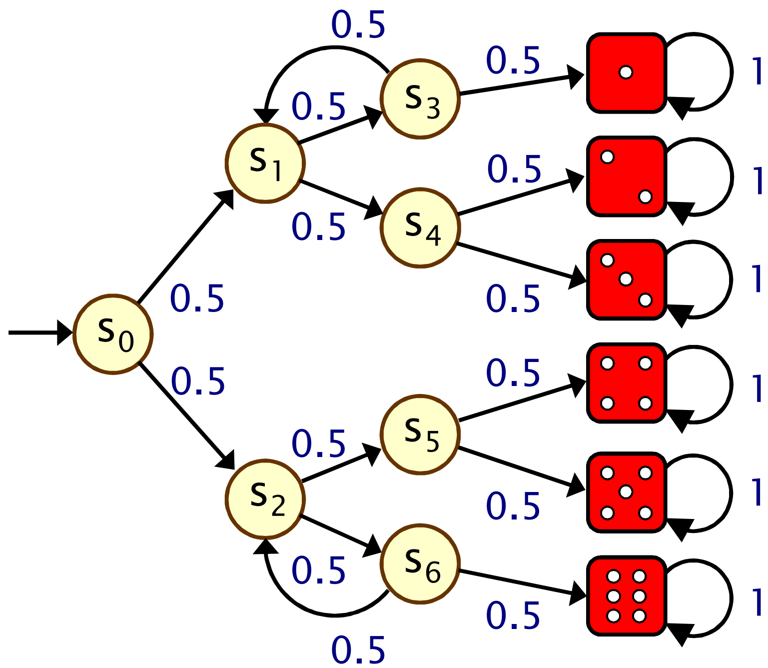
- Let x_s denote ProbReach(s, T)

- Solve:

$$x_s = \begin{cases} 1 & \text{if } s \in T \\ 0 & \text{if T is not reachable from s} \\ \sum_{s' \in S} P(s,s') \cdot x_{s'} & \text{otherwise} \end{cases}$$

Example

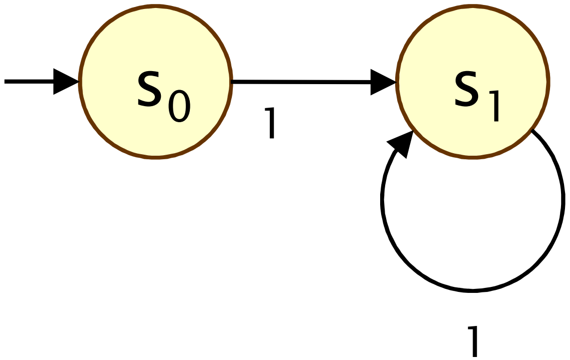
- Compute ProbReach(s_0, {4})



Unique solutions

- Why the need to identify states that cannot reach T?

- Consider this simple DTMC:
  - compute probability of reaching $\{s_0\}$ from s_1



  - linear equation system: $x_{s_0} = 1$, $x_{s_1} = x_{s_1}$
  - multiple solutions: $(x_{s_0}, x_{s_1}) = (1,p)$ for any $p \in [0,1]$

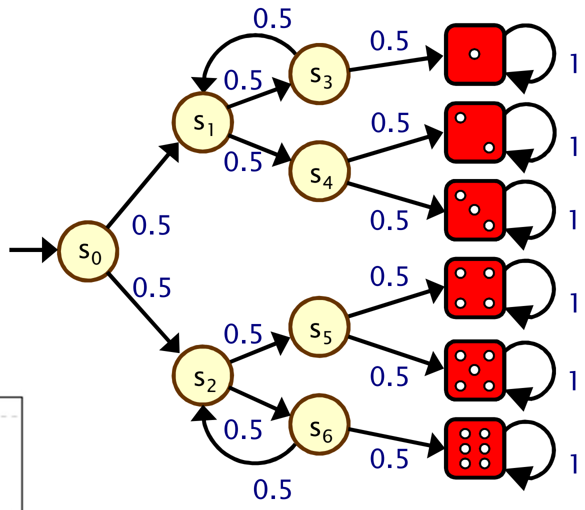
Bounded reachability probabilities

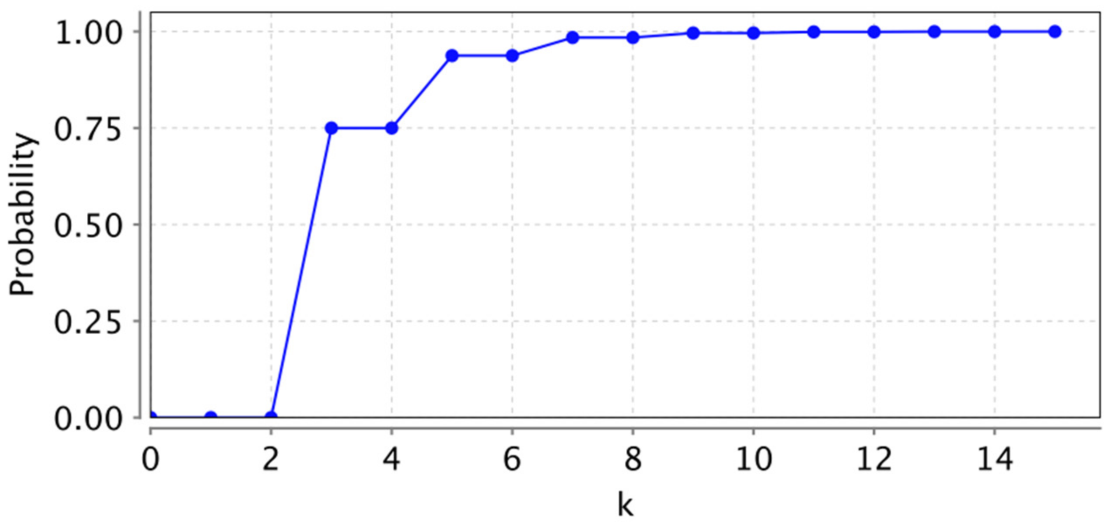
- Probability of reaching T from s within k steps
- Formally: $\text{ProbReach}^{\leq k}(s, T) = \Pr_s(\text{Reach}^{\leq k}(s, T))$ where:
  - $\text{Reach}^{\leq k}(s, T) = \{ s_0 s_1 s_2 \ldots \in \text{Path}(s) \mid s_i \text{ in } T \text{ for some } i \leq k \}$
- $\underline{\text{ProbReach}}^{\leq k}(T) = \underline{x}^{(k+1)}$ from the previous fixed point
  - which gives us…

$$\text{ProbReach}^{\leq k}(s, T) = \begin{cases} 1 & \text{if } s \in T \\ 0 & \text{if } k = 0 \;\&\; s \notin T \\ \sum_{s' \in S} P(s,s') \cdot \text{ProbReach}^{\leq k-1}(s', T) & \text{if } k > 0 \;\&\; s \notin T \end{cases}$$

(Bounded) reachability

- $\text{ProbReach}(s_0, \{1,2,3,4,5,6\}) = 1$
- $\text{ProbReach}^{\leq k}(s_0, \{1,2,3,4,5,6\}) = \ldots$



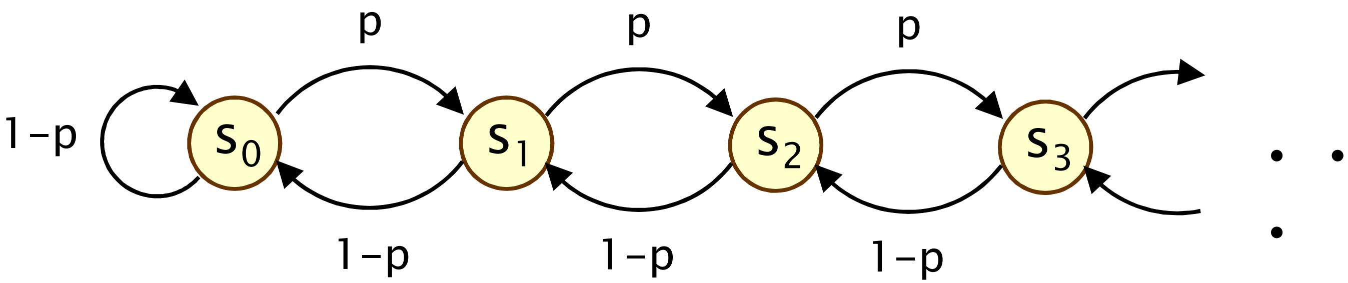


Qualitative properties

- Quantitative properties:
  - "what is the probability of event A?"
- Qualititative properties:
  - "the probability of event A is 1"   ("almost surely A")
  - or: "the probability of event A is > 0"   ("possibly A")
- For finite DTMCs, qualititative properties do not depend on the transition probabilities - only need underlying graph
  - e.g. to determine "is target set T reached with probability 1?" (see DTMC model checking later)

Aside: Infinite Markov chains

- Infinite-state random walk

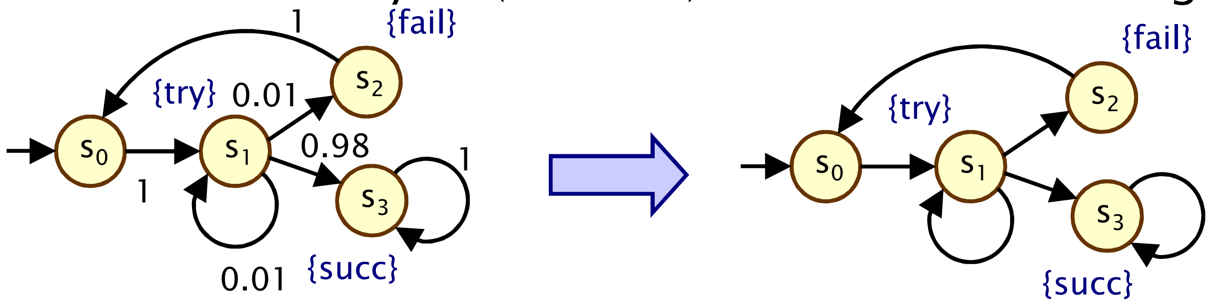


- Value of probability p does affect qualitative properties
  - ProbReach(s, $\{s_0\}$) = 1 if $p \leq 0.5$
  - ProbReach(s, $\{s_0\}$) < 1 if $p > 0.5$

Temporal logic

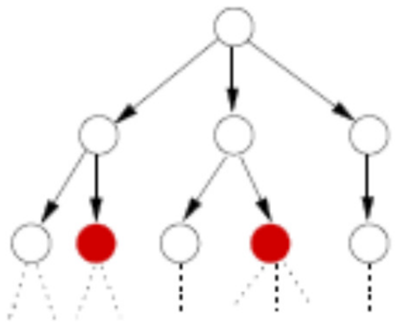
- Temporal logic
  - formal language for specifying and reasoning about how the behaviour of a system changes over time
  - defined over paths, i.e. sequences of states $s_0 s_1 s_2 s_3 \ldots$ such that $P(s_i,s_{i+1}) > 0 \; \forall i$

- Logics used in this course are probabilistic extensions of temporal logics devised for non-probabilistic systems (CTL, LTL)
  - So we revert briefly to (labelled) state-transition diagrams

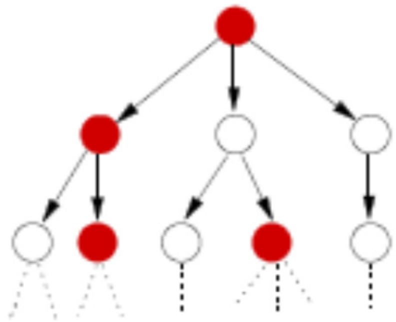


CTL semantics

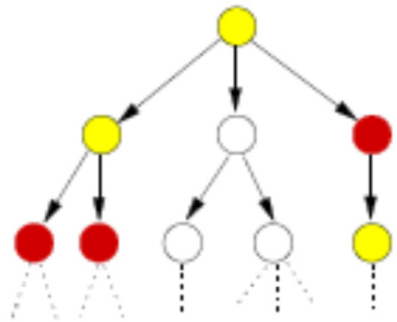
- Intuitive semantics:
  - of quantifiers (A/E) and temporal operators (F/G/U)



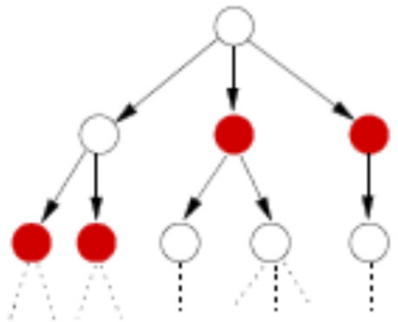
EF red



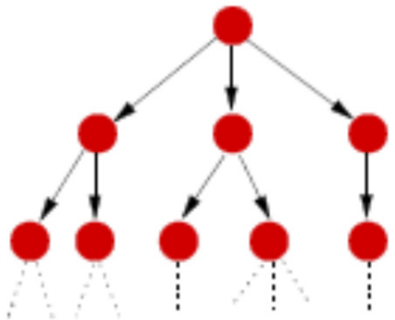
EG red



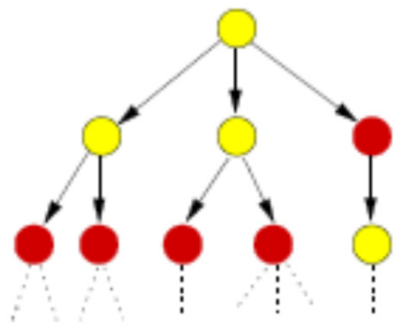
E [yellow U red]



AF red



AG red



A [yellow U red]

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:
  - ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $P_{\sim p}$ [ψ]   (state formulas)
  - ψ ::= X ϕ | ϕ $U^{\leq k}$ ϕ | ϕ U ϕ (path formulas)

ψ is true with probability ~p

"next" "bounded until" "until"

  - define F $\phi \equiv$ true U ϕ (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 $\in \mathbb{N}$

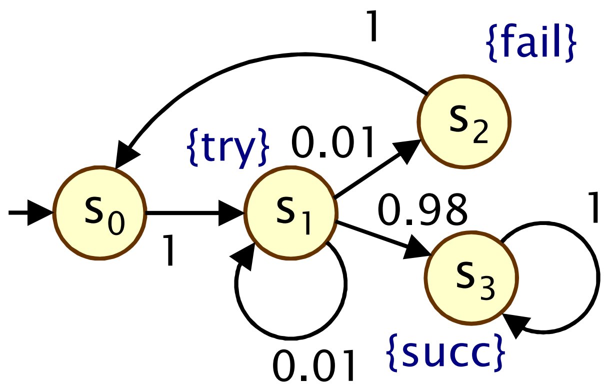
- A PCTL formula is always a state formula
  - path formulas only occur inside the P operator

PCTL semantics for DTMCs

- PCTL formulas interpreted over states of a DTMC
  - $s \models \phi$ 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},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

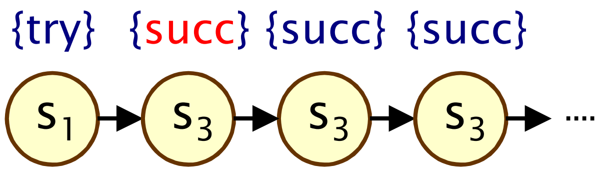
- Examples
  - $s_3 \models succ$
  - $s_1 \models try \wedge \neg fail$



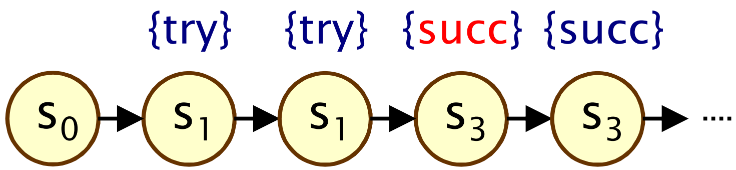
PCTL semantics for DTMCs

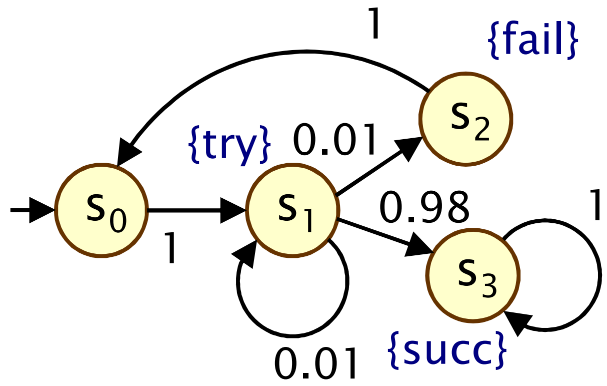
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \vDash X\ \phi \quad \Leftrightarrow \quad s_1 \vDash \phi$
  - $\omega \vDash \phi_1\ U^{\leq k}\ \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \vDash \phi_2$ and $\forall j<i,\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

- Some examples of satisfying paths:
  - X succ



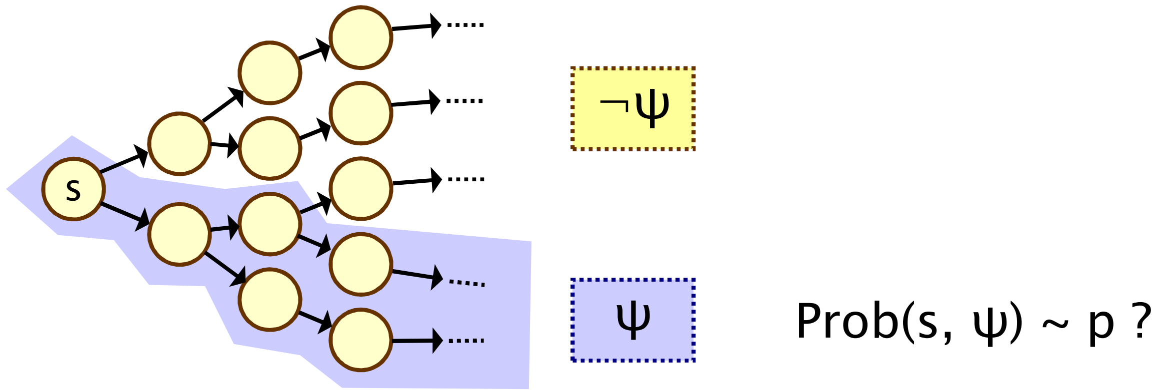
  - ¬fail U succ





PCTL semantics for DTMCs

- Semantics of the probabilistic operator P
  - informal definition: $s \models P_{\sim p} [\ \psi\ ]$ means that "the probability, from state s, that ψ is true for an outgoing path satisfies ~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: $Prob(s, \psi) = Pr_s \{\ \omega \in Path(s) \mid \omega \models \psi\ \}$
  - (sets of paths satisfying ψ are always measurable [Var85])



More PCTL…

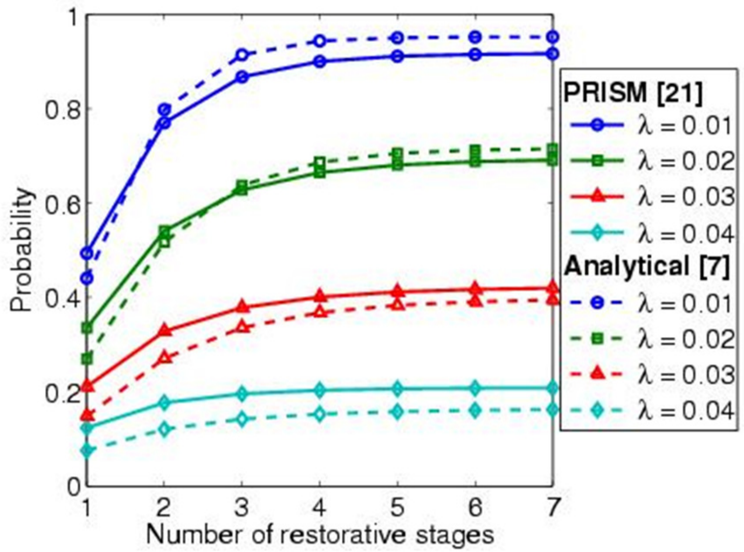
- Usual temporal logic equivalences:
  - false ≡ ¬true (false)
  - $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$ (disjunction)
  - $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$ (implication)

  - $F\ \phi \equiv \Diamond\ \phi \equiv \text{true}\ U\ \phi$ (eventually, “future”)
  - $G\ \phi \equiv \Box\ \phi \equiv \neg(F\ \neg\phi)$ (always, “globally”)
  - bounded variants: $F^{\leq k}\ \phi$, $G^{\leq k}\ \phi$

- Negation and probabilities
  - e.g. $\neg P_{>p}\ [\ \phi_1\ U\ \phi_2\ ] \equiv P_{\leq p}\ [\phi_1\ U\ \phi_2\ ]$
  - e.g. $P_{>p}\ [\ G\ \phi\ ] \equiv P_{<1-p}\ [\ F\ \neg\phi\ ]$

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?"

Reachability and invariance

- Derived temporal operators, like CTL…
- Probabilistic reachability: $P_{\sim p}$ [F ϕ]
  - the probability of reaching a state satisfying ϕ
  - $F\ \phi \equiv \text{true}\ U\ \phi$
  - "ϕ is eventually true"
  - bounded version: $F^{\leq k}\ \phi \equiv \text{true}\ U^{\leq k}\ \phi$
- Probabilistic invariance: $P_{\sim p}$ [G ϕ]
  - the probability of ϕ always remaining true
  - $G\ \phi \equiv \neg(F\ \neg\phi) \equiv \neg(\text{true}\ U\ \neg\phi)$
  - "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 formulae

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)

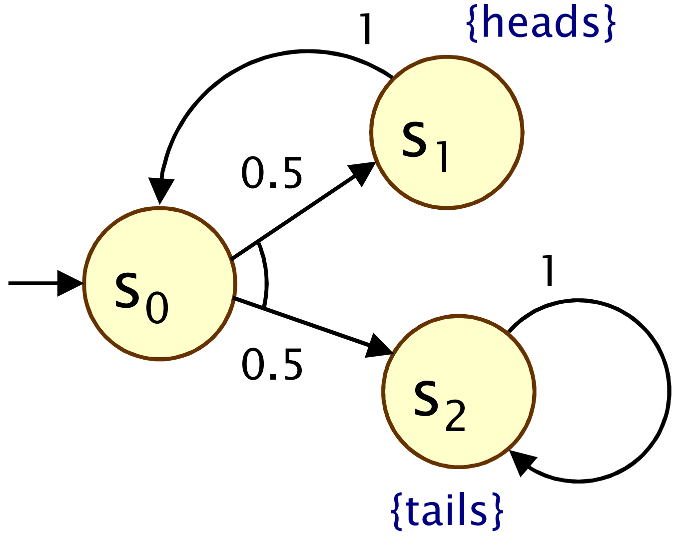
- Qualitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is either 0 or 1
- Quantitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is in the range (0,1)

- $P_{>0}$ [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
  - a ϕ-state is reached “almost surely”
  - 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

Overview (Part 1)

- Probability basics
- Model checking for discrete-time Markov chains (DTMCs)
  - DTMCs: definition, paths & probability spaces
  - PCTL model checking
  - Costs and rewards
- A glimpse of model checking for continuous-time Markov chains (CTMCs)
- Case studies
  - Bluetooth device discovery
  - Power management
- PRISM developments: parametric model checking
- Summary

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: $Sat(\phi) = \{ s \in S \mid s \models \phi \}$ = set of states satisfying ϕ

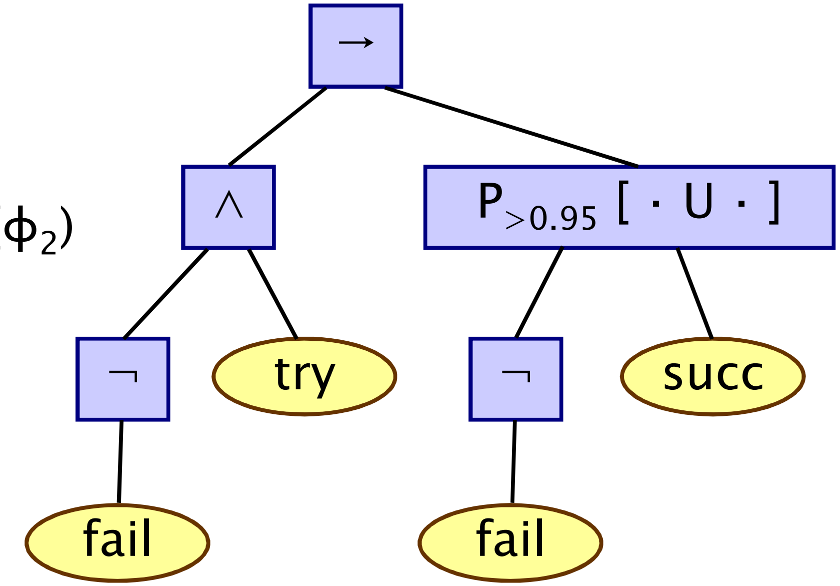
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - sometimes, want to check that $s \models \phi \ \forall \ s \in S$, i.e. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{init} \models \phi$, i.e. if $s_{init} \in Sat(\phi)$

- 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: $\phi = (\neg fail \wedge try) \rightarrow P_{>0.95} [\ \neg fail\ U\ succ\ ]$

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



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

Probability computation

- Three temporal operators to consider:

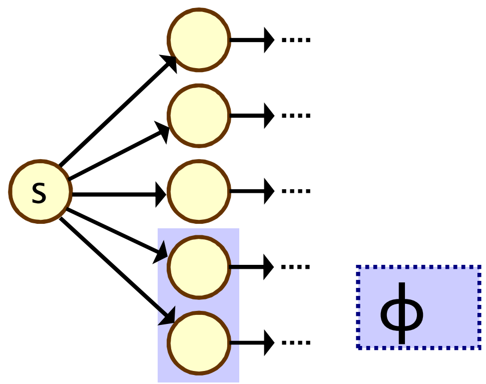
- Next: $P_{\sim p}[\ X\ \phi\ ]$

- Bounded until: $P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]$ (omitted)
  - adaptation of bounded reachability for DTMCs

- Until: $P_{\sim p}[\ \phi_1\ U\ \phi_2\ ]$
  - adaptation of reachability for DTMCs
  - graph-based “precomputation” algorithms
  - techniques for solving large linear equation systems

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
  - $Prob(s, X\ \phi) = \Sigma_{s' \in Sat(\phi)}\ P(s,s')$

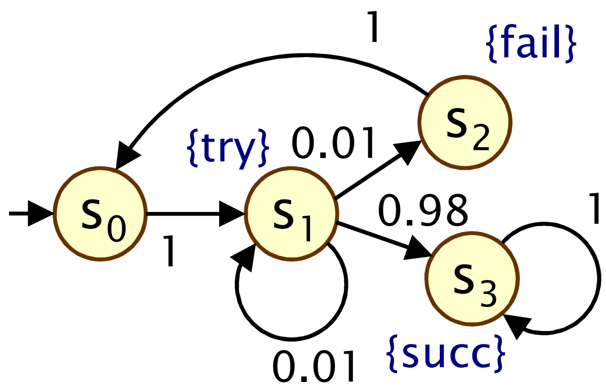


- Compute vector $\underline{Prob}(X\ \phi)$ of probabilities for all states s
  - $\underline{Prob}(X\ \phi) = P \cdot \underline{\phi}$
  - where $\underline{\phi}$ is a 0-1 vector over S with $\underline{\phi}(s) = 1$ iff $s \vDash \phi$
  - computation requires a single matrix-vector multiplication

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}
  - <u>Prob</u>(X (¬try ∨ succ)) = P · <u>(¬try ∨ succ)</u> = …

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



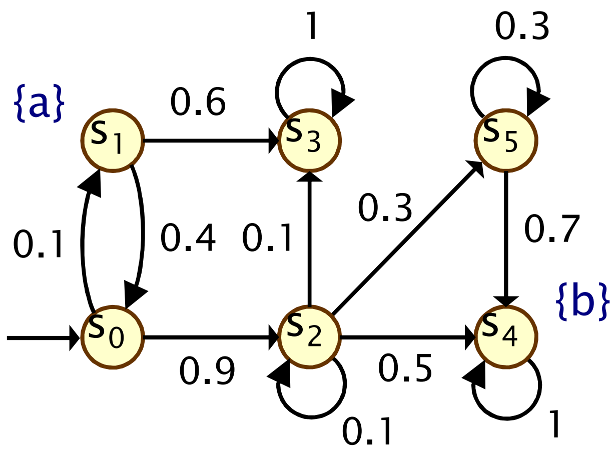
- Results:
  - <u>Prob</u>(X (¬try ∨ succ)) = [0, 0.99, 1, 1]
  - Sat($P_{\geq 0.9}$ [X (¬try ∨ succ)]) = {s_1, s_2, s_3}

PCTL until for DTMCs

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

- Running example:

$P_{>0.8} [\neg a \ U \ b \ ]$



Precomputation

- We refer to the first phase (identifying sets S^{yes} and S^{no}) as "precomputation"
  - two algorithms: Prob0 (for S^{no}) and Prob1 (for S^{yes})
  - algorithms work on underlying graph (probabilities irrelevant)

- Important for several reasons
  - ensures unique solution to linear equation system
    - only need Prob0 for uniqueness, Prob1 is optional
  - reduces the set of states for which probabilities must be computed numerically
  - gives exact results for the states in S^{yes} and S^{no} (no round-off)
  - for model checking of qualitative properties ($P_{\sim p}[\cdot]$ where p is 0 or 1), no further computation required

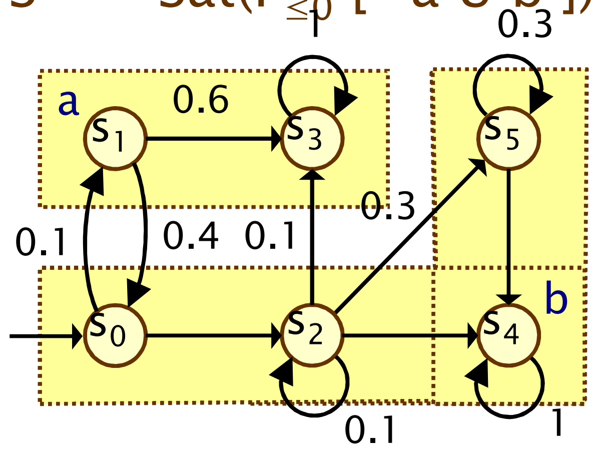
Precomputation – Prob0

- Prob0 algorithm to compute $S^{no} = Sat(P_{\leq 0}[\ \phi_1\ U\ \phi_2\ ])$:
  - first compute $Sat(P_{>0}[\ \phi_1\ U\ \phi_2\ ]) \equiv Sat(E[\ \phi_1\ U\ \phi_2\ ])$
  - 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

$S^{no} = Sat(P_{\leq 0}[\neg a\ U\ b\ ])$

Example:

$P_{>0.8}[\neg a\ U\ b\ ]$



Sat(b)

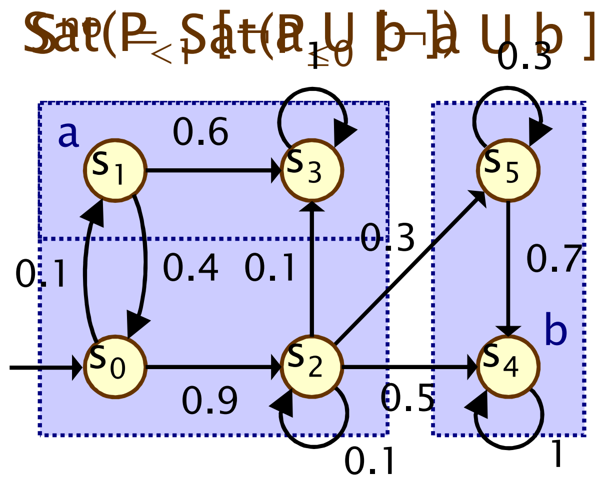
$Sat(P_{>0}[\neg a\ U\ b\ ])$

Precomputation – Prob1

- 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

Example:

$P_{>0.8}\ [\neg a\ U\ b\ ]$



$S^{yes} =$

$Sat(P_{\geq 1}\ [\neg a\ U\ b\ ])$

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
  - essentially the same as for probabilistic reachability

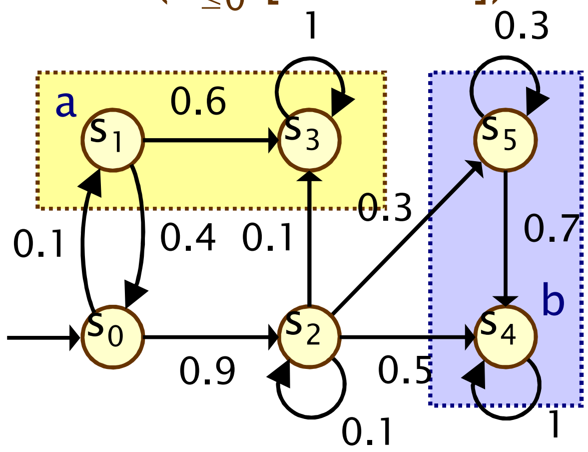
$$\text{Prob}(s, \phi_1 \text{ U } \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot \text{Prob}(s', \phi_1 \text{ U } \phi_2) & \text{otherwise} \end{cases}$$

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

PCTL until – linear equations

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

S^{no} = Sat($P_{\leq 0}$ [¬a U b])



S^{yes} = Sat($P_{\geq 1}$ [¬a U b])

$x_1 = x_3 = 0$

$x_4 = x_5 = 1$

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

$x_0 = 0.1x_1+0.9x_2 = 0.8$

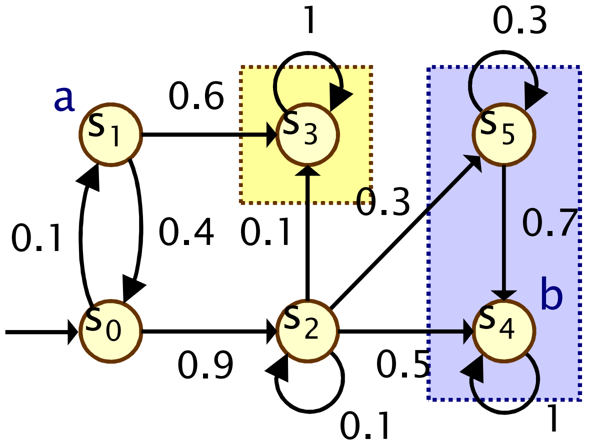
<u>Prob</u>(¬a U b) = <u>x</u> = [0.8, 0, 8/9, 0, 1, 1]

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

PCTL Until – Example 2

- Example: $P_{>0.5}$ [G¬b]
- Prob(s_i, G¬b)
  = 1 – Prob(s_i, ¬(G¬b))
  = 1 – Prob(s_i, F b)
- Let x_i = Prob(s_i, F b)

S^{no} = Sat($P_{\leq 0}$ [F b])



S^{yes} = Sat($P_{\geq 1}$ [F b])

$x_3 = 0$ and $x_4 = x_5 = 1$

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

$x_1 = 0.6x_3+0.4x_0 = 0.4x_0$

$x_0 = 0.1x_1+0.9x_2 = 5/6$ and $x_1 = 1/3$

$\underline{\text{Prob}}$(G¬b) = $\underline{1}$–$\underline{x}$ = [1/6, 2/3, 1/9, 1, 0, 0]

Sat($P_{>0.5}$ [G¬b]) = { s_1,s_3 }

Linear equation systems

- Solution of large (sparse) linear equation systems
  - size of system (number of variables) typically O(|S|)
  - state space S gets very large in practice
- Two main classes of solution methods:
  - direct methods – compute exact solutions in fixed number of steps, e.g. Gaussian elimination, L/U decomposition
  - iterative methods, e.g. Power, Jacobi, Gauss-Seidel, …
  - the latter are preferred in practice due to scalability
- General form: $\mathrm{A} \cdot \underline{x} = \underline{b}$
  - indexed over integers,
  - i.e. assume S = { 0,1,…,|S|-1 }

$$\sum_{j=0}^{|S|-1} \mathrm{A}(i,j) \cdot \underline{x}(j) \;=\; \underline{b}(i)$$

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},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

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"

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
  - based on random variables over (infinite) paths

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

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

  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 $\omega = s_0 s_1 s_2 \ldots$

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

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

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

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

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

Overview (Part 1)

- Probability basics
- Model checking for discrete-time Markov chains (DTMCs)
  - DTMCs: definition, paths & probability spaces
  - PCTL model checking
  - Costs and rewards
- A glimpse of model checking for continuous-time Markov chains (CTMCs)
- Case studies
  - Bluetooth device discovery
  - Power management
- PRISM developments: parametric model checking
- Summary

Continuous-time Markov chains

- Continuous-time Markov chains (CTMCs)
  - labelled transition systems augmented with rates
  - discrete states, continuous time-steps
  - delays exponentially distributed

- Formally, a CTMC C is a tuple (S, s_{init}, R, L) where:
  - S is a finite set of states ("state space")
  - $s_{init} \in S$ is the initial state
  - $R : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition rate matrix
  - $L : S \rightarrow 2^{AP}$ is a labelling with atomic propositions

- Transition rates
  - transition between s and s' when $R(s,s')>0$
  - probability triggered before t time units $1 - e^{-R(s,s')\cdot t}$

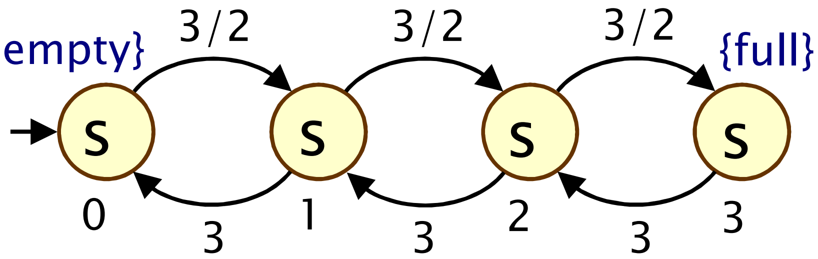
 68

Simple CTMC example

$C = (S, s_{init}, R, L)$

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

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$ $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

$$R = \begin{bmatrix} 0 & 3/2 & 0 & 0 \\ 3 & 0 & 3/2 & 0 \\ 0 & 3 & 0 & 3/2 \\ 0 & 0 & 3 & 0 \end{bmatrix} \qquad Q = \begin{bmatrix} -3/2 & 3/2 & 0 & 0 \\ 3 & -9/2 & 3/2 & 0 \\ 0 & 3 & -9/2 & 3/2 \\ 0 & 0 & 3 & -3 \end{bmatrix}$$

transition rate matrix

infinitesimal generator matrix

Transient and steady-state behaviour

- Transient behaviour
  - state of the model at a particular time instant
  - $\underline{\pi}_{s,t}(s')$ is probability of, having started in state s, being in state s' at time t
  - $\underline{\pi}_{s,t}(s') = Pr_s\{ \omega \in Path(s) \mid \omega@t=s' \}$
- Steady-state behaviour
  - state of the model in the long-run
  - $\underline{\pi}_{s}(s')$ is probability of, having started in state s, being in state s' in the long run
  - $\underline{\pi}_{s}(s') = \lim_{t\to\infty} \underline{\pi}_{s,t}(s')$
  - the percentage of time, in long run, spent in each state
- Can compute these numerically, from rates matrix R
  - e.g. embedded/uniformised DTMC

Temporal logic CSL

- CSL – Continuous Stochastic Logic
- Similar to PCTL, except real-valued time
  - $P_{=?}$ [$F^{[4,5.6]}$ outOfPower] – the (transient) probability of being out of power in time interval of 4 to 5.6 time units
  - $S_{=?}$ [minQoS] – the steady-state probability of satisfying minimum QoS
  - $R_{<10}$ [$C^{\leq 5}$] – cumulated reward up to time 5 is less than 10
- Model checking proceeds essentially via discretisation…
  - discretise CTMC to obtain DTMC (embedded, uniformised)
  - combine with graph-theoretical analysis

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

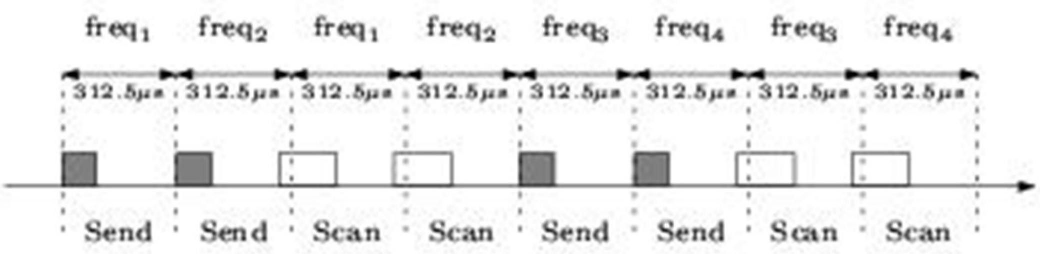
Case study: Bluetooth device discovery

- Bluetooth: short-range low-power wireless protocol
  - widely available in phones, PDAs, laptops, ...
  - 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
- Formation of personal area networks (PANs)
  - piconets (1 master, up to 7 slaves)
  - self-configuring: devices discover themselves
- Device discovery
  - mandatory first step before any communication possible
  - relatively high power consumption so performance is crucial
  - master looks for devices, slaves listens for master

Master (sender) behaviour

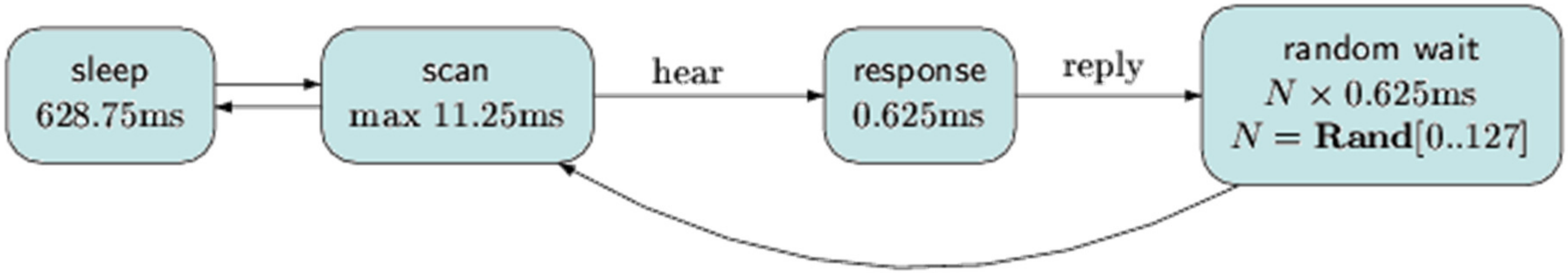
- 28 bit free-running clock CLK, ticks every 312.5µs
- Frequency hopping sequence determined by clock:
  - freq = $[CLK_{16-12}+k+(CLK_{4-2,0}-CLK_{16-12}) \bmod 16] \bmod 32$
  - 2 trains of 16 frequencies (determined by offset k), 128 times each, swap between every 2.56s
- Broadcasts "inquiry packets" on two consecutive frequencies, then listens on the same two

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	16



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 – PRISM model

- Modelled/analysed using PRISM model checker [DKNP06]
  - model scenario with one sender and one receiver
  - synchronous (clock speed defined by Bluetooth spec)
  - model at lowest-level (one clock-tick = one transition)
  - randomised behaviour so model as a DTMC
  - 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, so huge number of possible initial configurations (17,179,869,184)

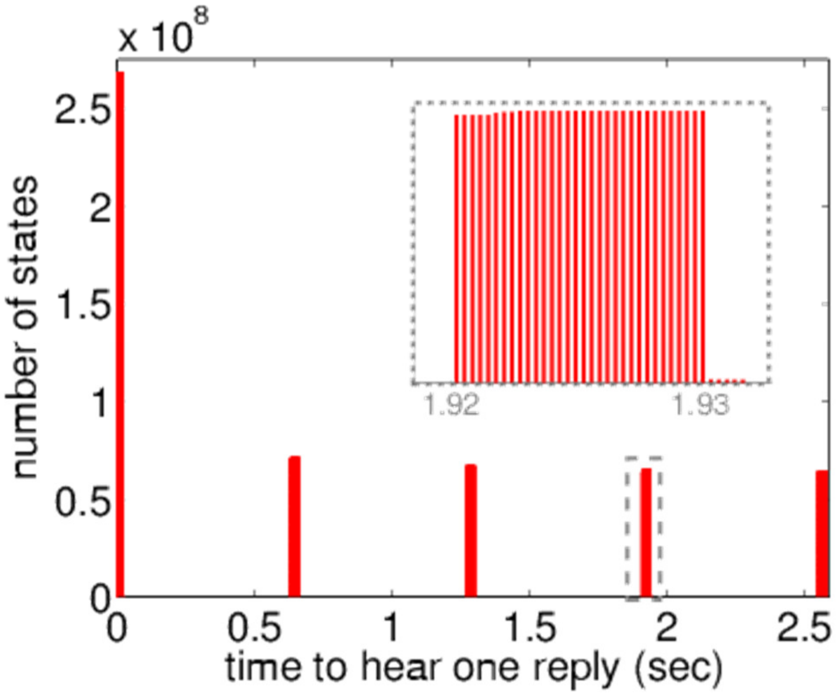
Bluetooth – Results

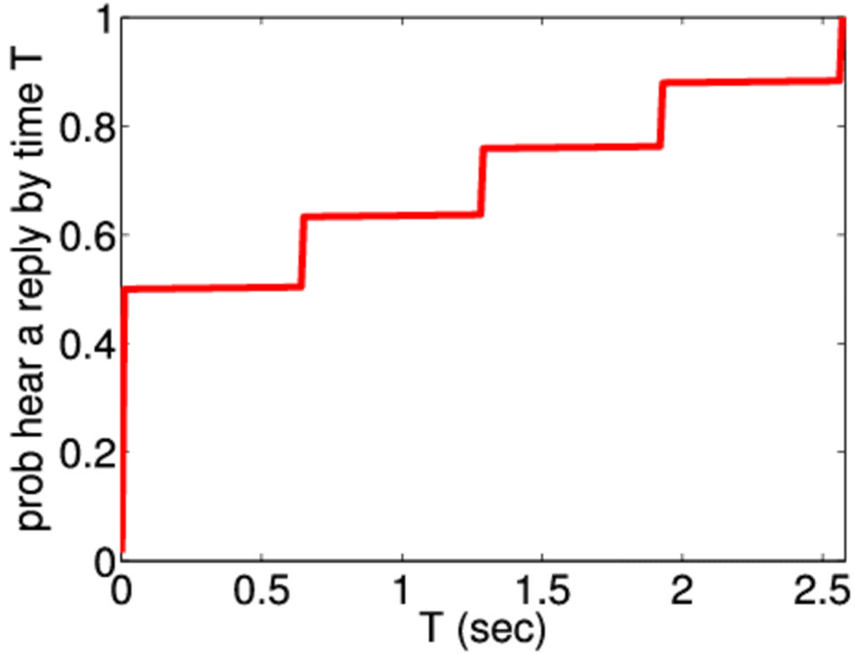
- Huge DTMC – initially, model checking infeasible
  - partition into 32 scenarios, i.e. 32 separate DTMCs
  - on average, approx. 3.4×10^9 states (536,870,912 initial)
  - can be built/analysed with PRISM's MTBDD engine

- We compute:
  - R=? [F replies=K {“init”}{max}]
  - “worst-case expected time to hear K replies over all possible initial configurations”

- Also look at:
  - how many initial states for each possible expected time
  - cumulative distribution function (CDF) for time, assuming equal probability for each initial state

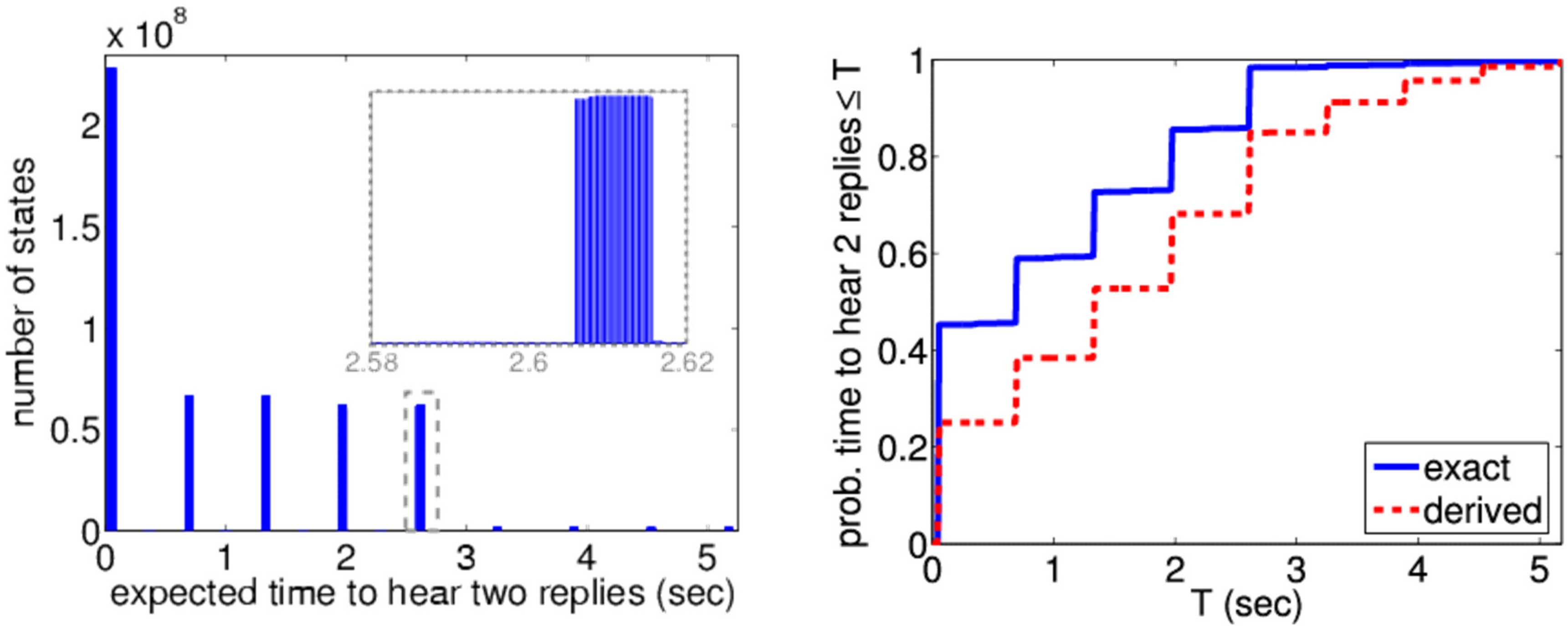
Bluetooth – Time to hear 1 reply





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

Bluetooth – Time to hear 2 replies



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

Case study: Power management

- Power management
  - controls power consumption in battery-operated devices
  - savings in power usage translate to extended battery life
  - important for portable, mobile and handheld electronic devices
- System level power management
  - manages various system devices for power optimisation
  - system components manufactured with several power modes
  - e.g. disk drive has: active, idle, standby, sleep, …
  - modes can be changed by the operating system through APIs
  - exploits application characteristics
  - needs to be implemented at the O/S level

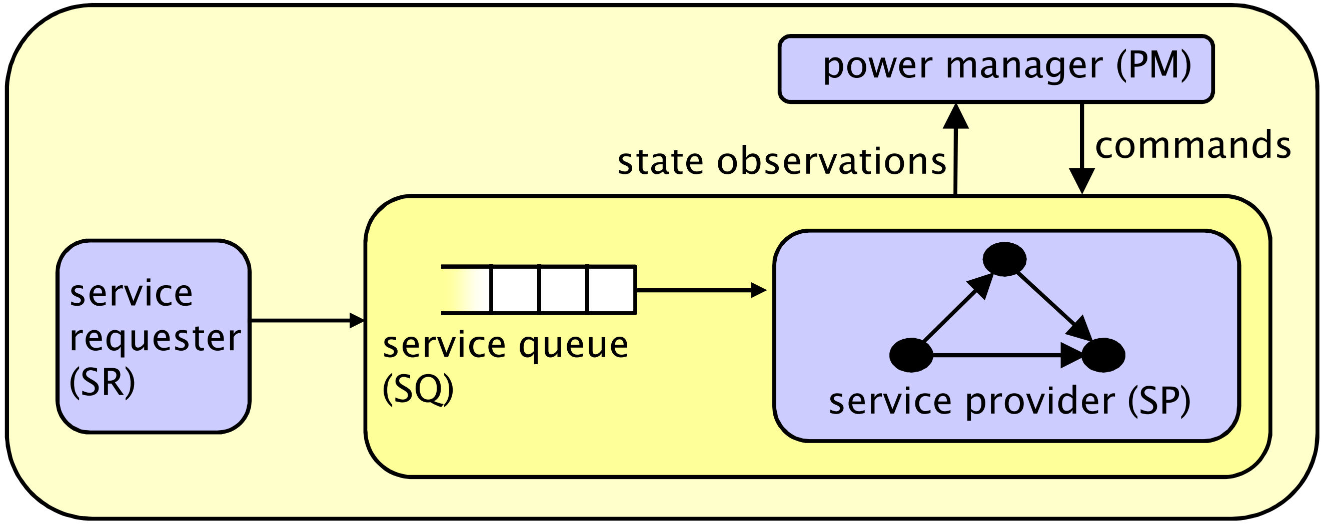
Dynamic Power Management (DPM)

- DPM make optimal decisions at runtime based on:
  - dynamically changing system state
  - workload
  - performance constraints

- Stochastic optimal control strategies for DPM
  - construct a mathematical model of the system in PRISM
  - transition times modelled with exponential distributions
  - formulate stochastic optimisation problems
    e.g. “optimise av. energy usage while av. delay below k”
  - create stochastic strategies by solving optimisation problem
    (exported to Maple for solution externally)
  - analyse strategies in PRISM

DPM – The system model

- Service requester (generates the service requests)
- Service provider (provides service to the requests)
- Service queue (buffers the requests)
- Power manager (monitors the states of the SP and SQ and issues state-transition commands to the SP)



Fujitsu disk drive – The PRISM model

- 4 state Fujitsu disk drive: busy, idle, standby and sleep
- Policies:
  - minimize the average power consumption
  - constraint on the average queue size

- Reward structure "power" (power consumption)
  - state rewards: the av. power consumption of SP in the state
  - transition rewards: energy consumed when SP changes state
- Reward structure "queue" (queue size)
  - state rewards: current size of the queue
- Reward structure "lost" (lost requests)
  - transition rewards: assign 1 to transitions representing the arrival of a request in a state where the queue is full

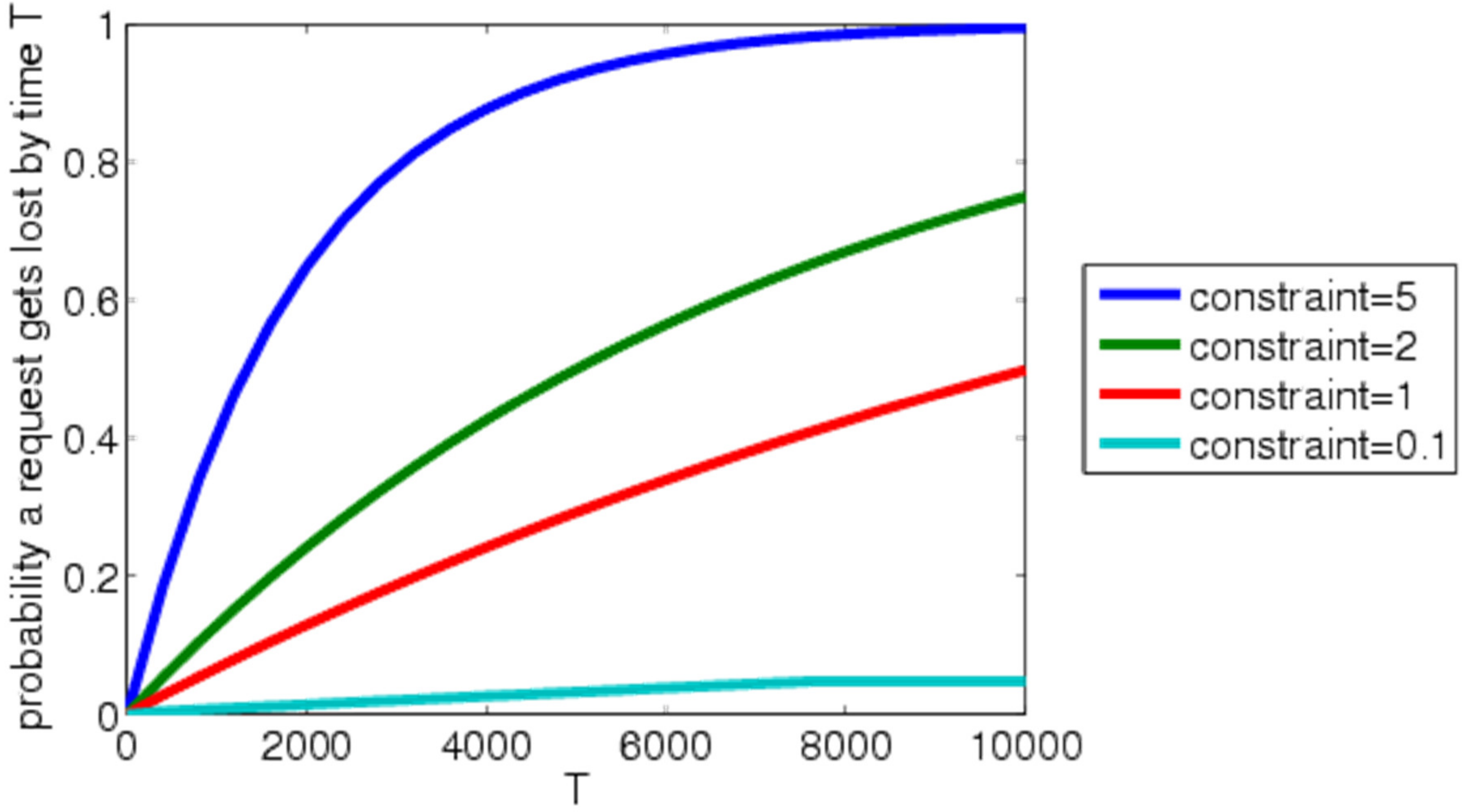
Fujitsu disk drive - Properties

- Selection of properties checked with PRISM

- Probability that queue size becomes ≥ M by time t
  - $P_{=?}[F^{\leq t} (q \geq M)]$
- Probability that at least M requests get lost by time t
  - $P_{=?}[F^{\leq t} (lost \geq M)]$
- Expected queue size at time t
  - $R_{\{\text{“queue”}\}=?}[I^{=t}]$
- Expected power consumption by time t
  - $R_{\{\text{“power”}\}=?}[C^{\leq t}]$
- Long run average number of requests lost
  - $R_{\{\text{“lost”}\}=?}[S]$

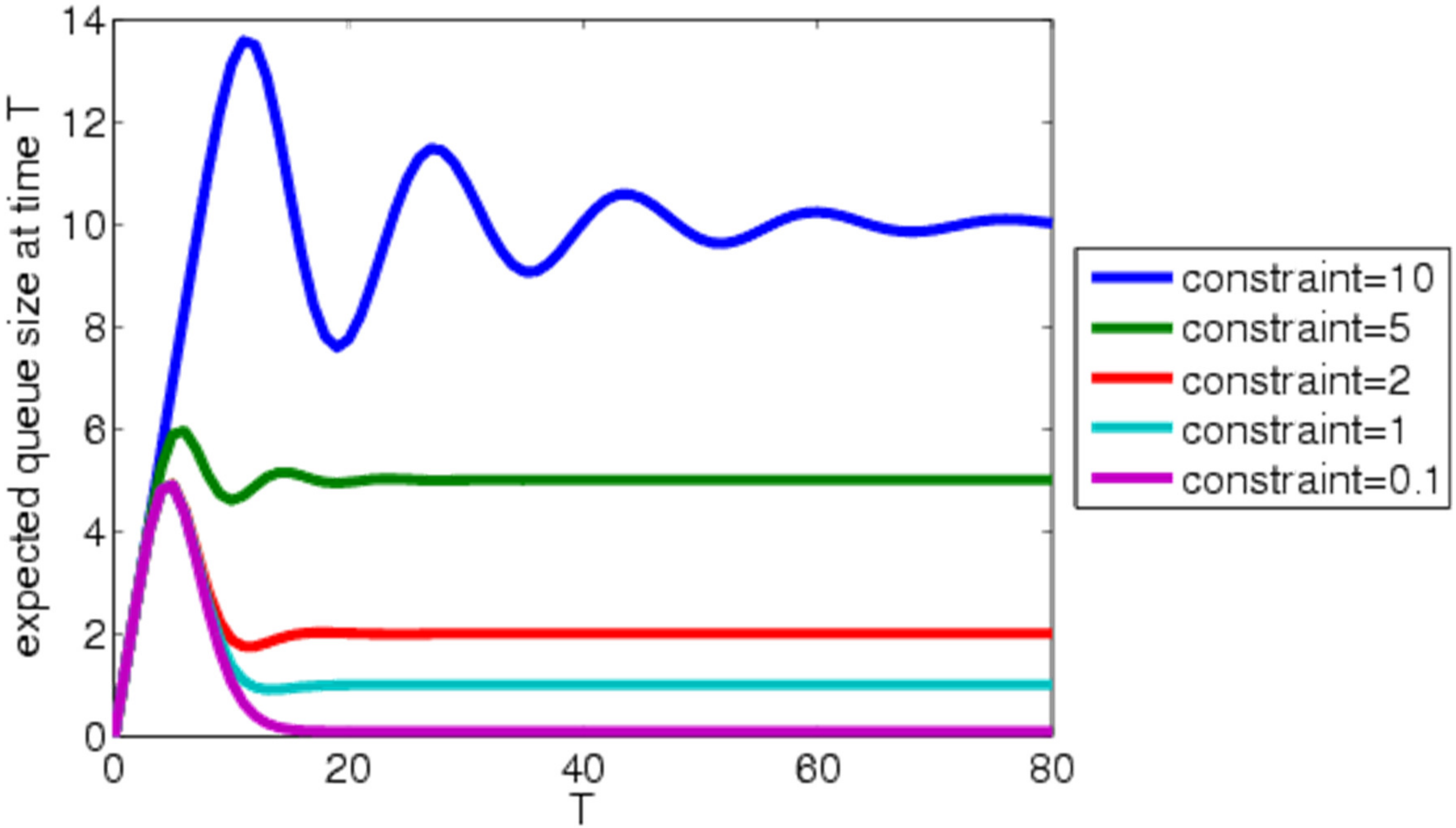
Fujitsu disk drive – PRISM results

- Probability M requests lost by time t $P_{=?}[F^{\leq t}\ (lost \geq M)]$



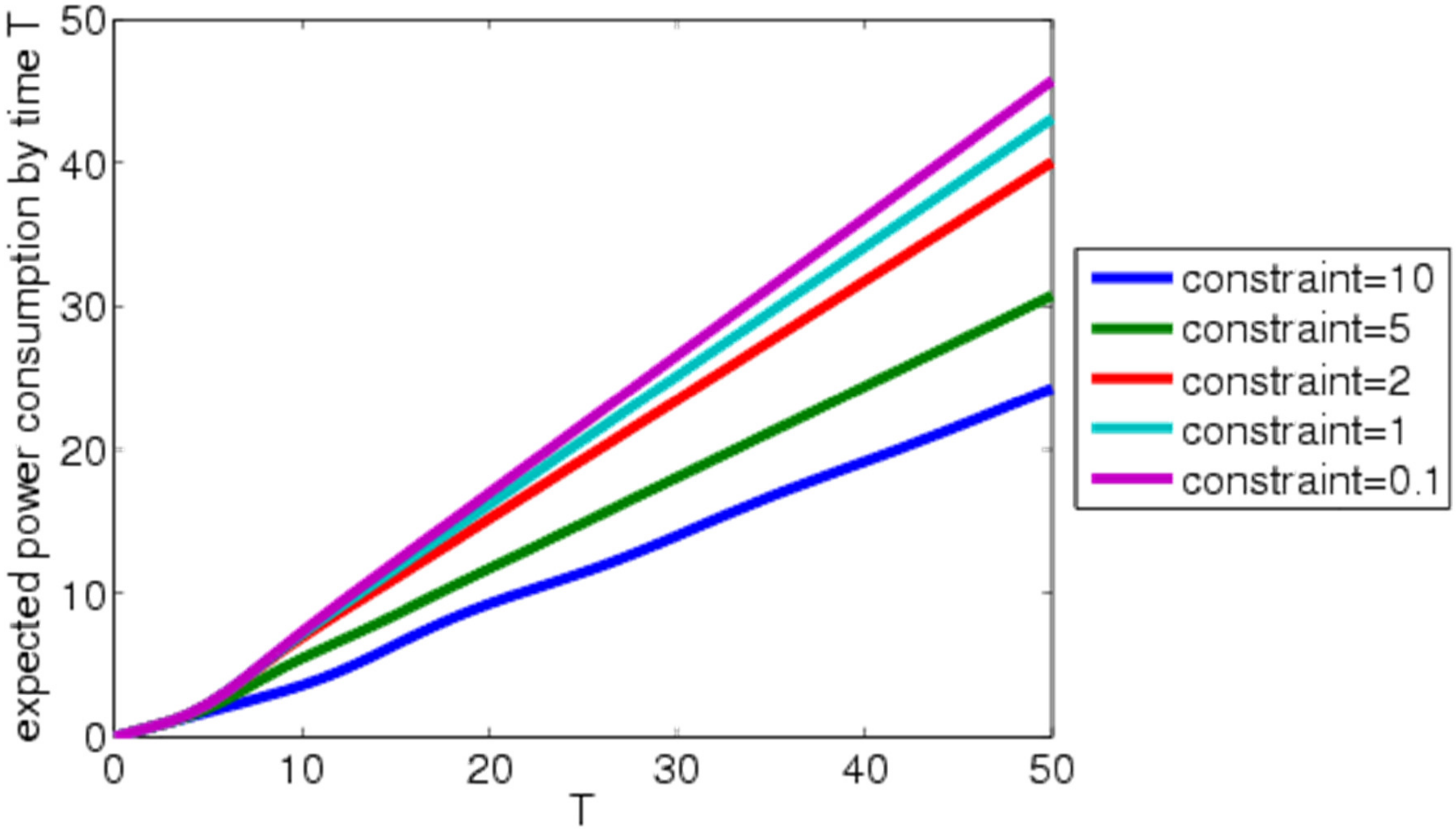
Fujitsu disk drive – PRISM results

- Expected queue size at time t $R_{\{\text{"queue"}\}=?}[I^{=t}]$



Fujitsu disk drive – PRISM results

- Expected power consumption by time t $R_{\{\text{"power"}\}=?}[C^{\leq t}]$



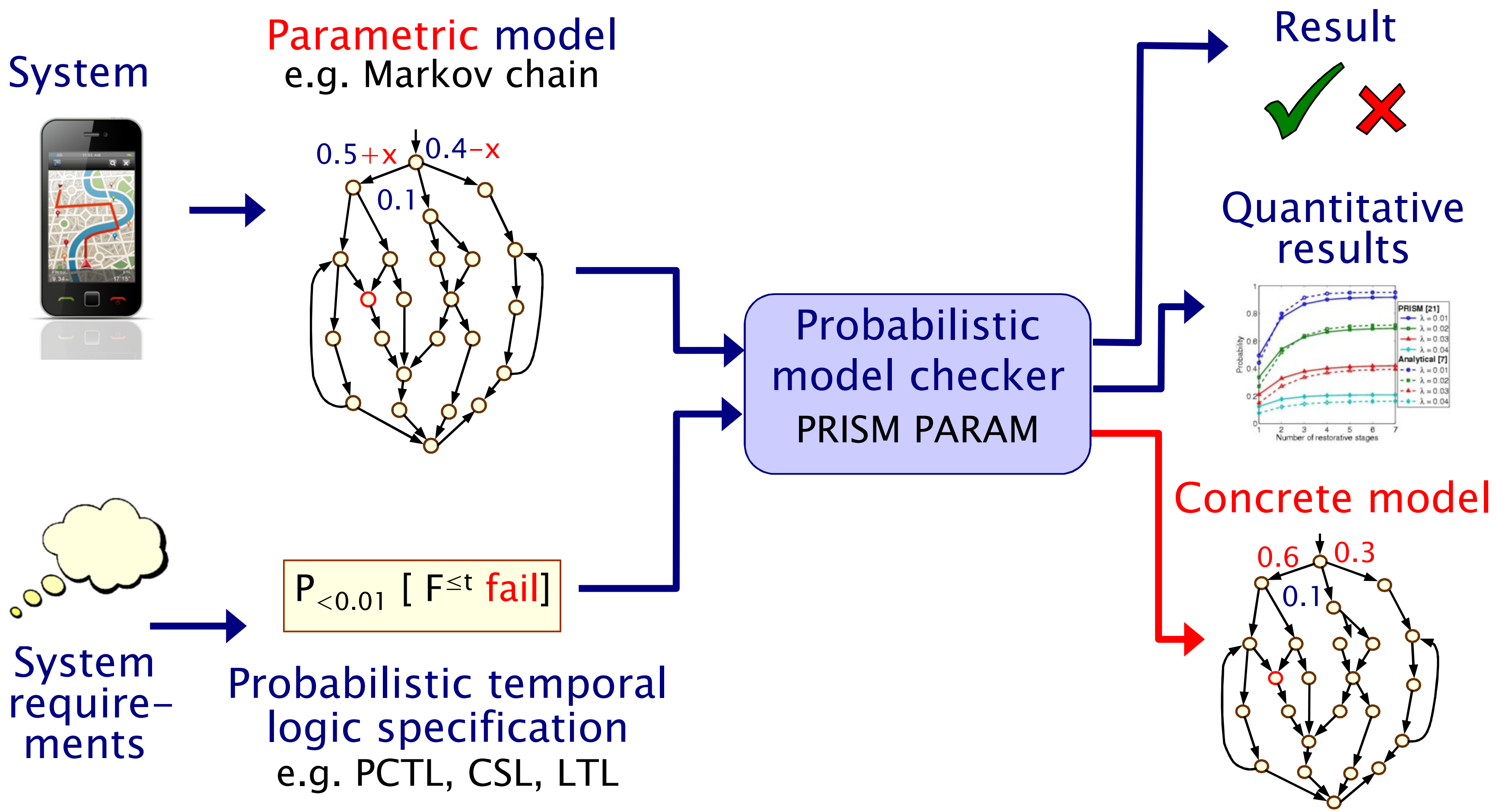
PRISM: Recent & new developments

- New features:
  1. parametric model checking
  2. parameter synthesis
  3. strategy synthesis
  4. stochastic multi-player games
  5. real-time: probabilistic timed automata (PTAs)

- Further new additions:
  - enhanced statistical model checking (approximations + confidence intervals, acceptance sampling)
  - efficient CTMC model checking (fast adaptive uniformisation)
  - benchmark suite & testing functionality
  - www.prismmodelchecker.org

– Beyond PRISM…

Parametric model checking and synthesis



Parametric model checking in PRISM

- Parametric Markov chain models in PRISM
  - probabilistic parameters expressed as unevaluated constants
  - e.g. const double x;
  - transition probabilities are expressions over parameters, e.g. 0.4 + x
- Properties are given in PCTL, with parameter constants
  - new construct constfilter (min, x1*x2, phi)
  - filters over parameter values, rather than states
- Implemented in ‘explicit’ engine
  - returns mapping from parameter regions (e.g. [0.2,0.3],[−2,0]) to rational functions over the parameters
  - filter properties used to find parameter values that optimise the function
  - reimplementation of PARAM 2.0 [Hahn et al]

Parameter synthesis

- Find optimal parameter value given a parametric model and PCTL/CSL property
  - parametric probabilities and rates

- Techniques
  - discretisation and integer parameters
  - constraint solving, including parametric symbolic constraints
  - iterative refinement to improve accuracy
  - sampling to improve efficiency
  - but scalability is still the biggest challenge

- Implementation
  - using tool combination involving Z3, python, PRISM

Summary (Part 1)

- 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
- Case studies: Bluetooth and Power management

- Next: Markov decision processes (MDPs)