



Probabilistic model checking with PRISM

Marta Kwiatkowska

Department of Computer Science, University of Oxford

Gran Sasso Science Institute, L'Aquila, June 2015



What is probabilistic model checking?

- Probabilistic model checking (aka probabilistic/quantitative verification)...
 - is a **formal verification** technique for modelling and analysing systems that exhibit **probabilistic** behaviour
- Formal verification...
 - is the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems

Why formal verification?

- Errors in computerised systems can be costly...



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



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

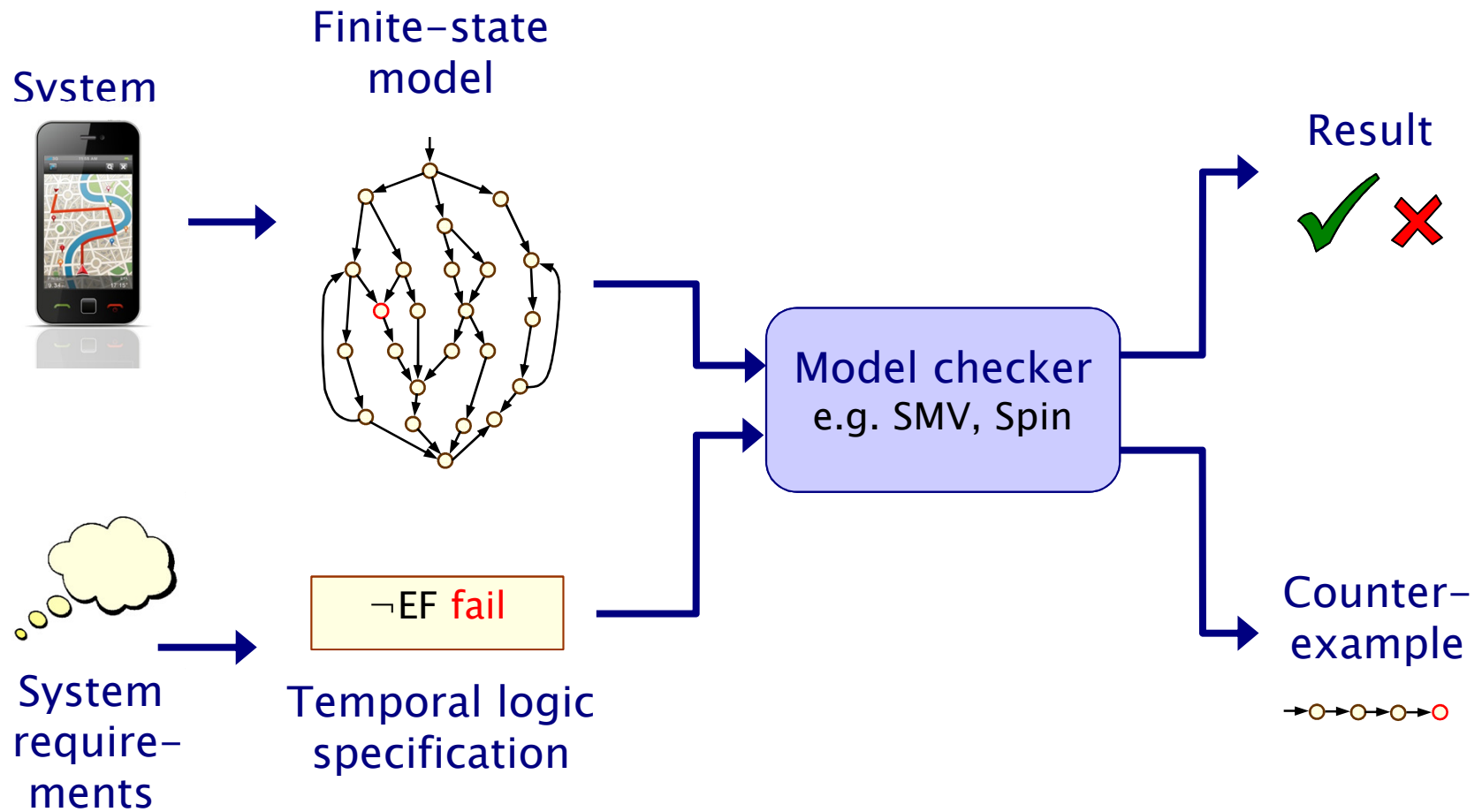


Toyota Prius (2010)
Software “glitch”
found in anti-lock
braking system.
185,000 cars recalled.

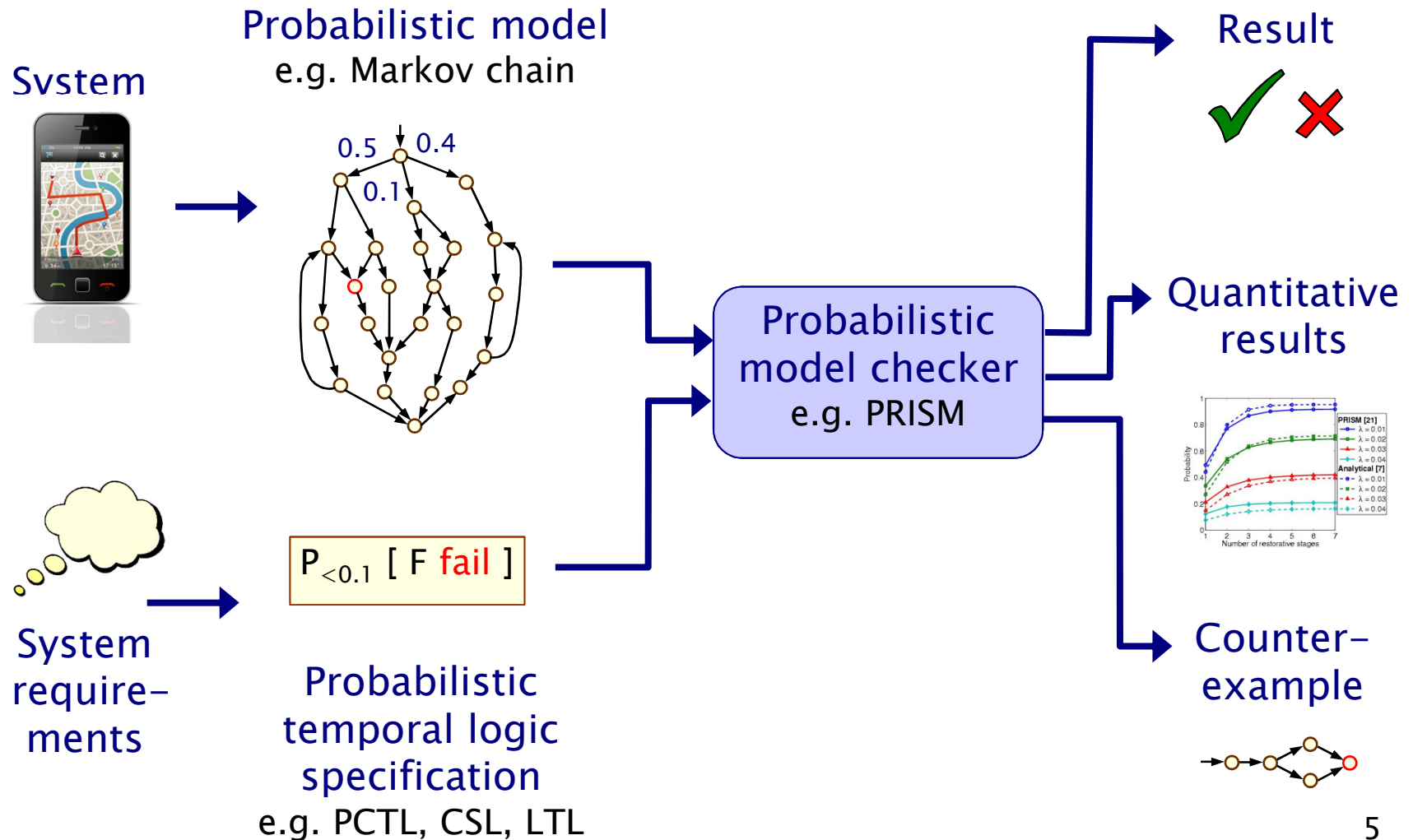
- Why verify?**
 - “Testing can only show the presence of errors, not their absence.” [Edsger Dijkstra]



Model checking



Probabilistic model checking



Why probability?

- Some systems are inherently probabilistic...
- **Randomisation**, e.g. in distributed coordination algorithms
 - as a symmetry breaker, in gossip routing to reduce flooding
- Examples: real-world protocols featuring randomisation:
 - Randomised back-off schemes
 - CSMA protocol, 802.11 Wireless LAN
 - Random choice of waiting time
 - IEEE1394 Firewire (root contention), Bluetooth (device discovery)
 - Random choice over a set of possible addresses
 - IPv4 Zeroconf dynamic configuration (link-local addressing)
 - Randomised algorithms for anonymity, contract signing, ...

Why probability?

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

Why probability?

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



Verifying probabilistic systems

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

Probabilistic models

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

Probabilistic models

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

Course material

- Course slides and lab session
 - <http://www.prismmodelchecker.org/courses/gran-sasso15/>
- Reading
 - [MDPs/LTL] Forejt, Kwiatkowska, Norman and Parker. Automated Verification Techniques for Probabilistic Systems. LNCS vol 6659, p53–113, Springer 2011.
 - [DTMCs/CTMCs] Kwiatkowska, Norman and Parker. Stochastic Model Checking. LNCS vol 4486, p220–270, Springer 2007.
 - [DTMCs/MDPs/LTL] Principles of Model Checking by Baier and Katoen, MIT Press 2008
- See also
 - 20 lecture course taught at Oxford
 - <http://www.prismmodelchecker.org/lectures/pmc/>
- PRISM website www.prismmodelchecker.org



Part 1

Discrete-time Markov chains

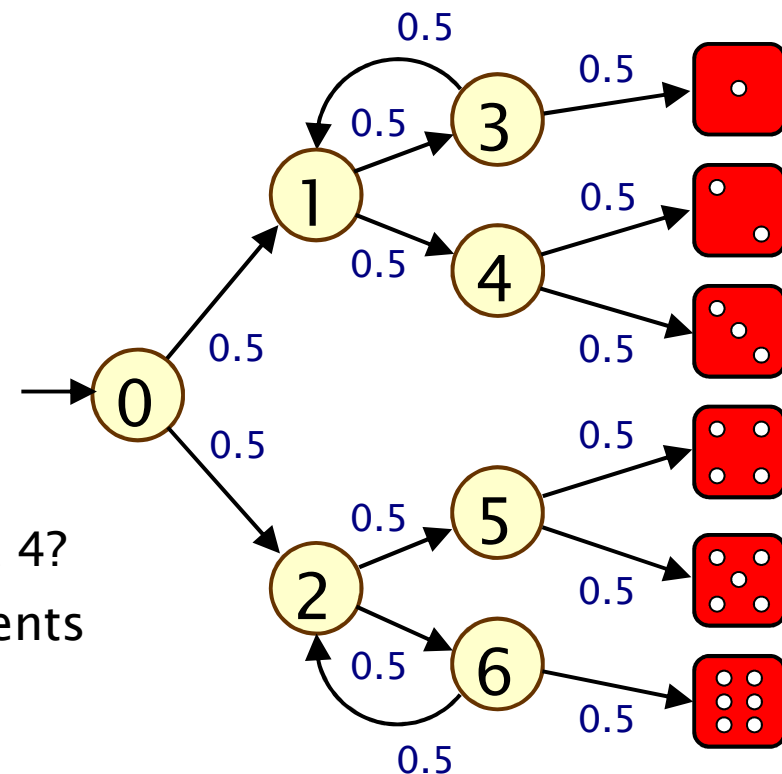
Overview (Part 1)

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

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

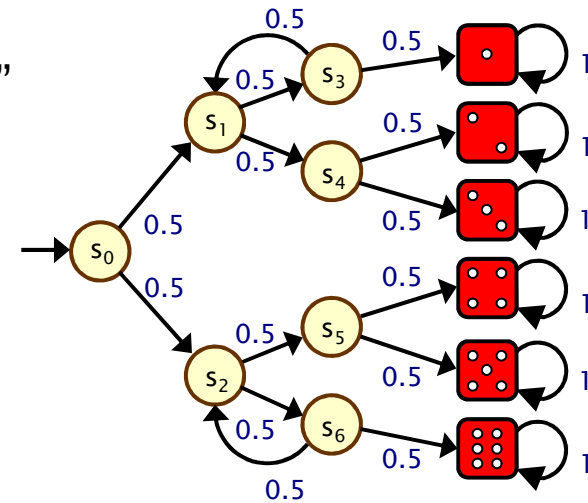


- Is this algorithm correct?

- e.g. probability of obtaining a 4?
- obtain as disjoint union of events
- THH, TTTHH, TTTTTHH, ...
- $\Pr(\text{"eventually 4"})$
 $= (1/2)^3 + (1/2)^5 + (1/2)^7 + \dots = 1/6$

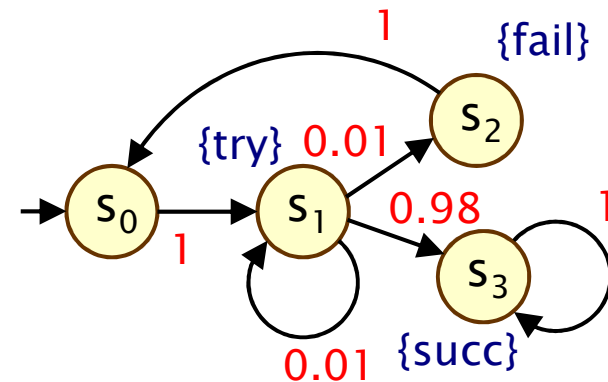
Example...

- 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



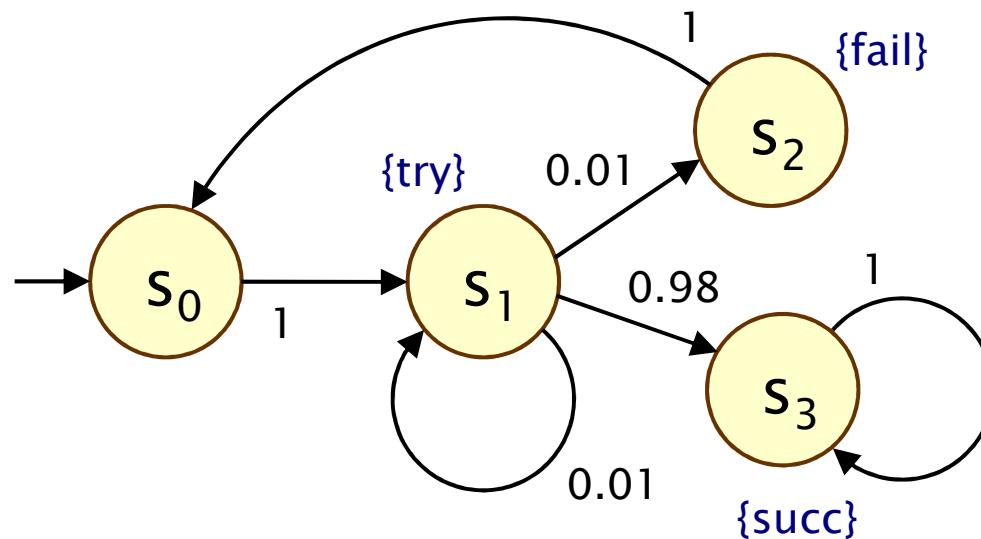
Discrete-time Markov chains

- 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**



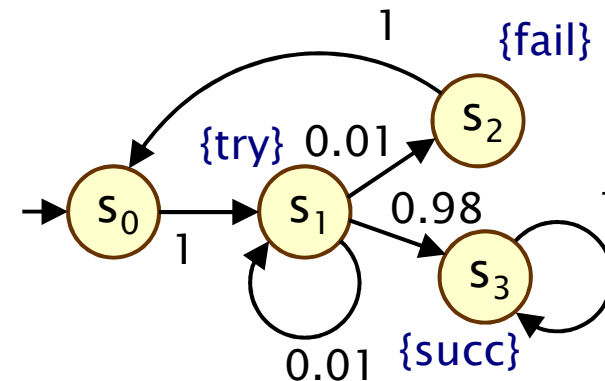
Simple DTMC example

- 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 $(S, s_{\text{init}}, P, L)$ where:
 - S is a finite set of states (“state space”)
 - $s_{\text{init}} \in S$ is the initial state
 - $P : S \times S \rightarrow [0,1]$ is the **transition probability matrix** where $\sum_{s' \in S} P(s, s') = 1$ 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

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

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

$$s_{\text{init}} = s_0$$

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

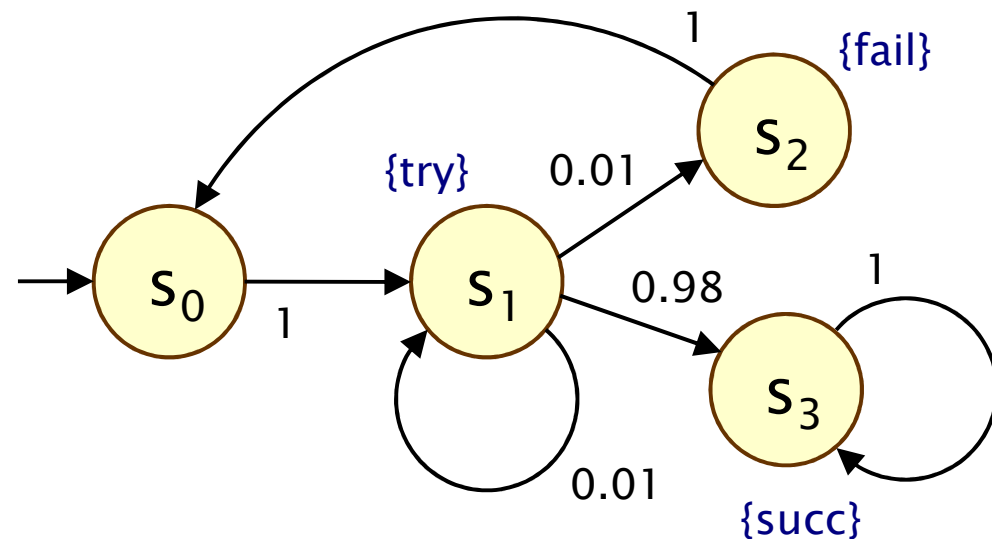
$$L(s_0) = \emptyset,$$

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

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

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

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



Some more terminology

- **P** is a **stochastic matrix**, meaning it satisfies:
 - $P(s,s') \in [0,1]$ for all $s,s' \in S$ and $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$
- A **sub-stochastic matrix** satisfies:
 - $P(s,s') \in [0,1]$ for all $s,s' \in S$ and $\sum_{s' \in S} P(s,s') \leq 1$ 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

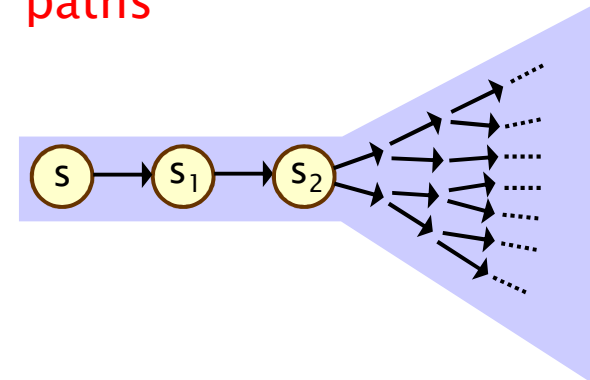
- Alternative definition... a DTMC is:
 - a family of random variables $\{ X(k) \mid k=0,1,2,\dots \}$
 - where $X(k)$ are observations at discrete time-steps
 - i.e. $X(k)$ is the state of the system at time-step k
 - which satisfies...
- The Markov property (“memorylessness”)
 - $\Pr(X(k)=s_k \mid X(k-1)=s_{k-1}, \dots, X(0)=s_0)$
= $\Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$
 - 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 $s_{\text{init}} \in S$ can be generalised...
 - to an initial probability distribution $s_{\text{init}} : S \rightarrow [0,1]$
- 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 $s_0s_1s_2s_3\dots$ 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: $\text{Path}(s)$ = 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 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 \emptyset 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 $(\Omega, \Sigma, \text{Pr})$**
 - Ω is the sample space
 - Σ is the set of events: σ -algebra on Ω
 - $\text{Pr} : \Sigma \rightarrow [0,1]$ is the probability measure:
 $\text{Pr}(\Omega) = 1$ and $\text{Pr}(\cup_i A_i) = \sum_i \text{Pr}(A_i)$ for countable disjoint A_i

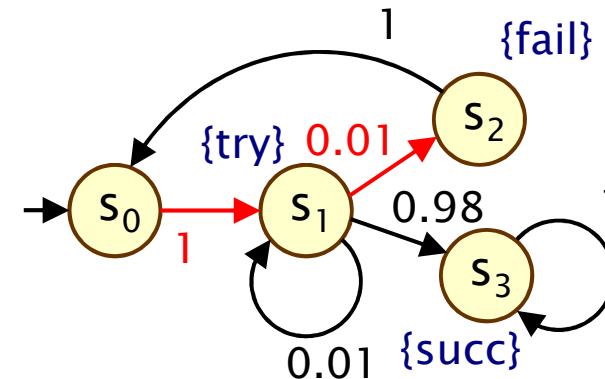
Probability space over paths

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

Probability space – Example

- Paths where sending fails the first time

- $\omega = s_0 s_1 s_2$
- $C(\omega) = \text{all paths starting } s_0 s_1 s_2 \dots$
- $P_{s_0}(\omega) = P(s_0, s_1) \cdot P(s_1, s_2)$
 $= 1 \cdot 0.01 = 0.01$
- $\Pr_{s_0}(C(\omega)) = P_{s_0}(\omega) = 0.01$



- Paths which are eventually successful and with no failures

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

Reachability

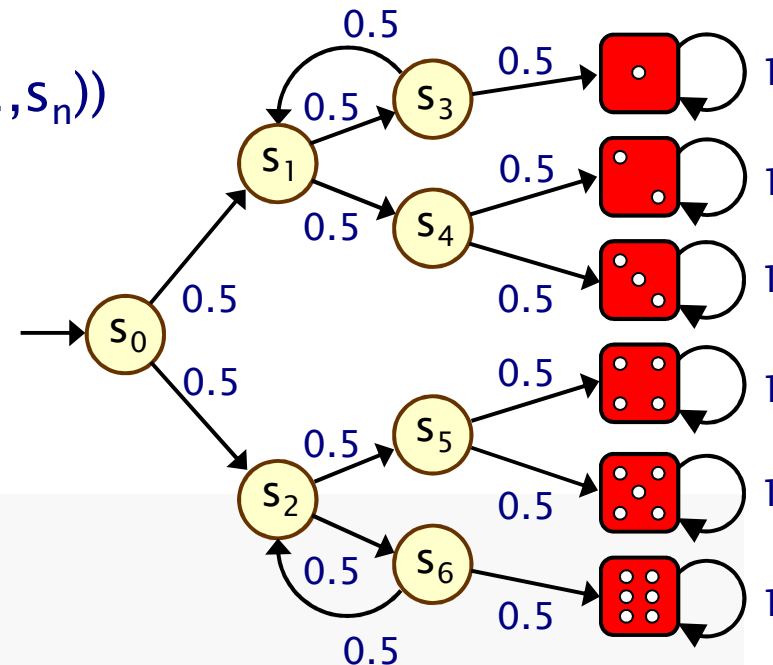
- Key property: **probabilistic reachability**
 - probability of a path reaching a state in some target set $T \subseteq S$
 - e.g. “probability of the algorithm terminating successfully?”
 - e.g. “probability that an error occurs during execution?”
- Dual of reachability: **invariance**
 - probability of remaining within some class of states
 - $\text{Pr}(\text{“remain in set of states } T\text{”}) = 1 - \text{Pr}(\text{“reach set } S \setminus T\text{”})$
 - e.g. “probability that an error never occurs”
- We will also consider other variants of reachability
 - **time-bounded**, constrained (“**until**”), ...

Reachability probabilities

- Formally: $\text{ProbReach}(s, T) = \Pr_s(\text{Reach}(s, T))$
 - where $\text{Reach}(s, T) = \{ s_0 s_1 s_2 \dots \in \text{Path}(s) \mid s_i \in T \text{ for some } i \}$
- Is $\text{Reach}(s, T)$ measurable for any $T \subseteq S$? Yes...
 - $\text{Reach}(s, T)$ is the union of all basic cylinders $\text{Cyl}(s_0 s_1 \dots s_n)$ where $s_0 s_1 \dots s_n \in \text{Reach}_{\text{fin}}(s, T)$
 - $\text{Reach}_{\text{fin}}(s, T)$ contains all finite paths $s_0 s_1 \dots s_n$ such that: $s_0 = s$, $s_0, \dots, s_{n-1} \notin T$, $s_n \in T$ (reaches T **first time**)
 - set of such finite paths $s_0 s_1 \dots s_n$ is countable
- Probability
 - in fact, the above is a disjoint union
 - so probability obtained by simply summing...

Computing reachability probabilities

- Compute as (infinite) sum...
- $\sum_{s_0, \dots, s_n \in \text{Reachfin}(s, T)} \Pr_{s_0}(\text{Cyl}(s_0, \dots, s_n))$
 $= \sum_{s_0, \dots, s_n \in \text{Reachfin}(s, T)} P(s_0, \dots, s_n)$
- Example:
 - ProbReach(s_0 , {4})



Computing reachability probabilities

- Compute as (infinite) sum...

- $\sum_{s_0, \dots, s_n \in \text{Reachfin}(s, T)} \text{Pr}_{s_0}(\text{Cyl}(s_0, \dots, s_n))$

$$= \sum_{s_0, \dots, s_n \in \text{Reachfin}(s, T)} P(s_0, \dots, s_n)$$

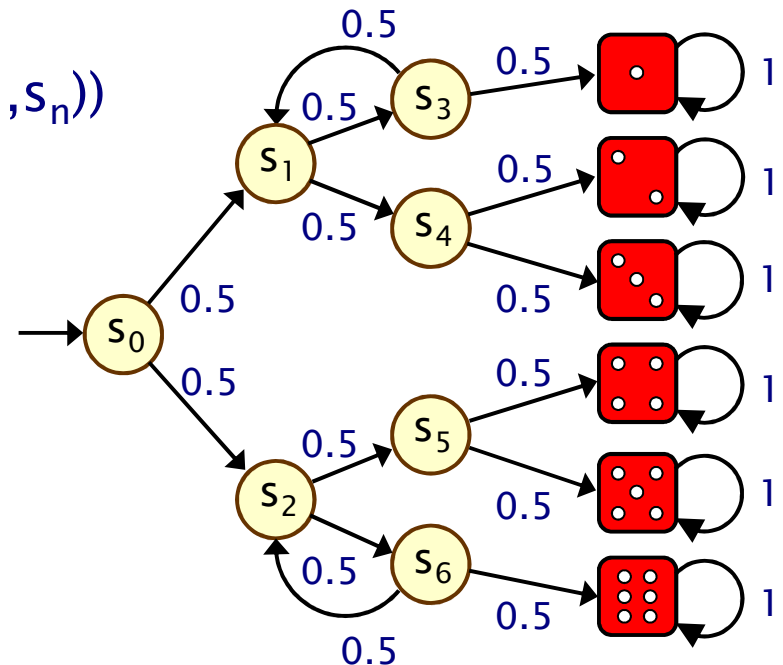
- Example:

- $\text{ProbReach}(s_0, \{4\})$
- $\text{Pr}_{s_0}(\text{Reach}(s_0, \{4\}))$
- Finite path fragments:

- $s_0(s_2s_6)^n s_2s_54$ for $n \geq 0$

- $P_{s_0}(s_0s_2s_54) + P_{s_0}(s_0s_2s_6s_2s_54) + P_{s_0}(s_0s_2s_6s_2s_6s_2s_54) + \dots$

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



Computing reachability probabilities

- Alternative: derive a **linear equation system**

- solve for all states simultaneously
- i.e. compute vector ProbReach(T)

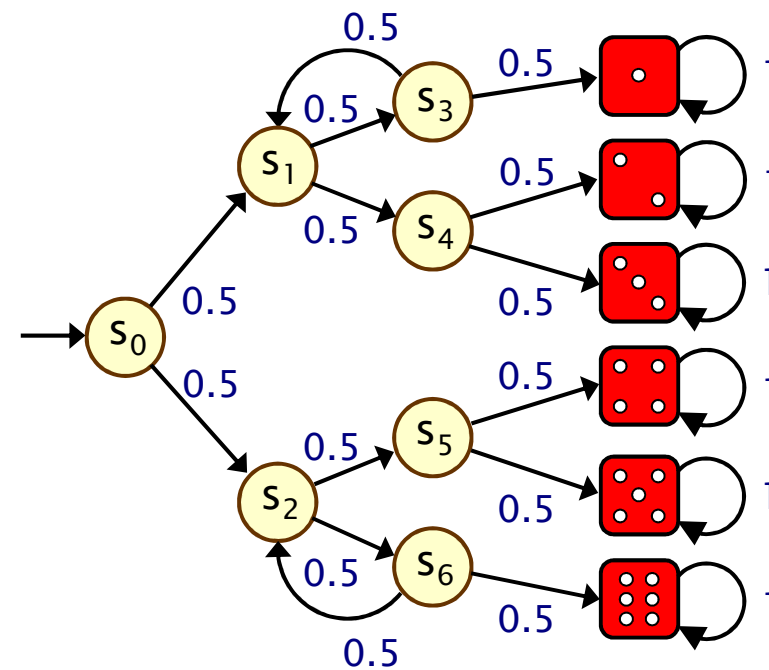
- Let x_s denote ProbReach(s , T)

- Solve:

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

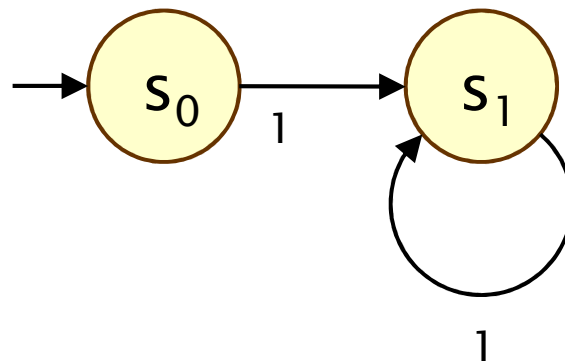
Example

- Compute $\text{ProbReach}(s_0, \{4\})$



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 $p \in [0, 1]$

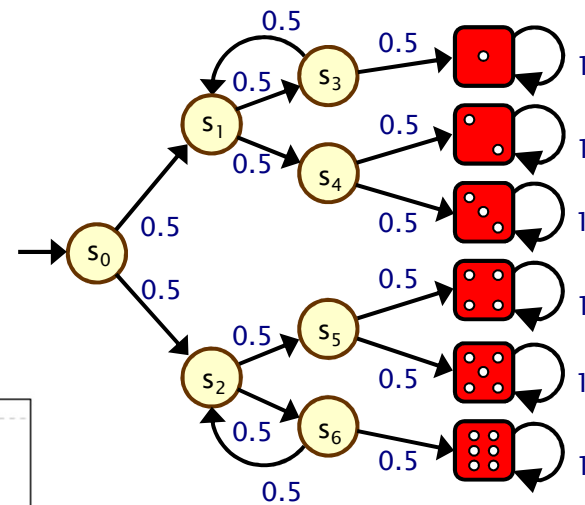
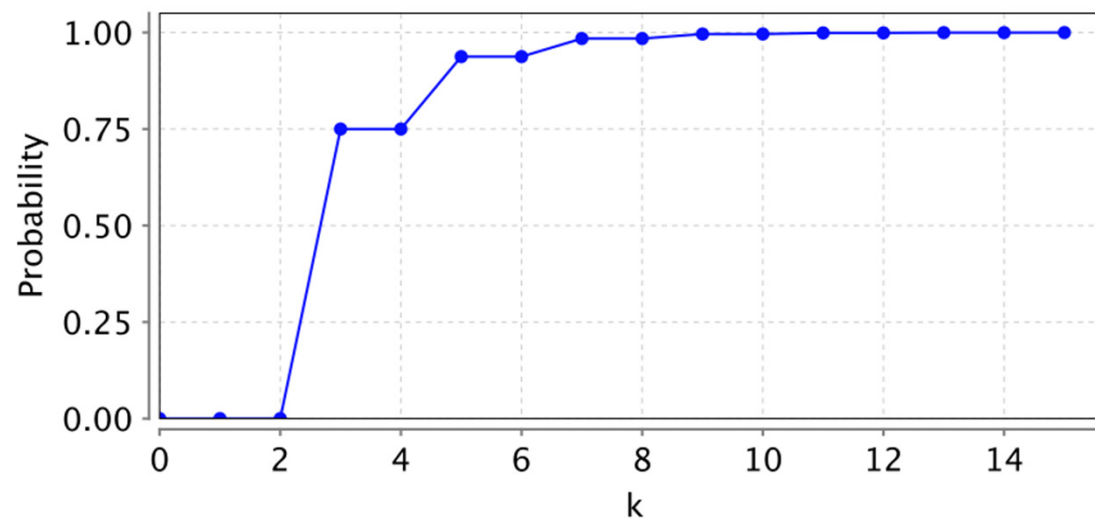
Bounded reachability probabilities

- 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:
 - $\text{Reach}^{\leq k}(s, T) = \{ s_0 s_1 s_2 \dots \in \text{Path}(s) \mid s_i \in T \text{ for some } i \leq k \}$
- $\text{ProbReach}^{\leq k}(T)$ = $\underline{x}^{(k+1)}$ from the previous fixed point
 - which gives us...

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

(Bounded) reachability

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

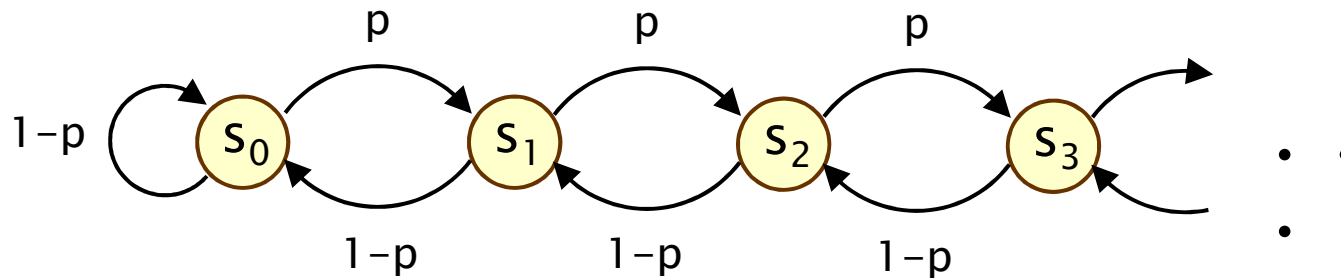


Qualitative properties

- **Quantitative** properties:
 - “what is the probability of event A?”
- **Qualitative** properties:
 - “the probability of event A is 1” (“almost surely A”)
 - or: “the probability of event A is > 0 ” (“possibly A”)
- For finite DTMCs, qualitative 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

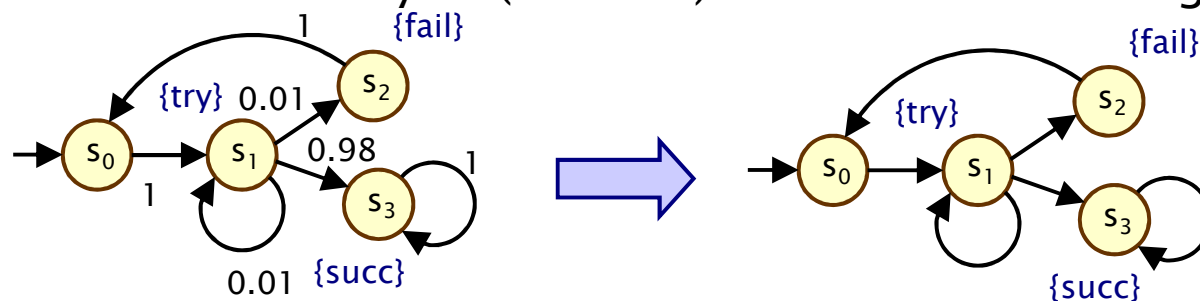
- Infinite-state random walk



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

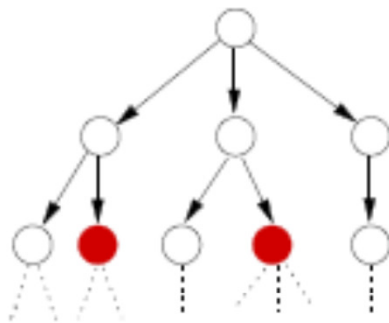
Temporal logic

- 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 $s_0s_1s_2s_3\dots$ 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

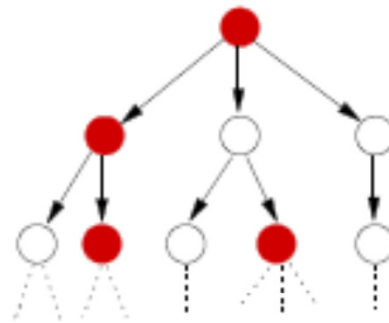


CTL semantics

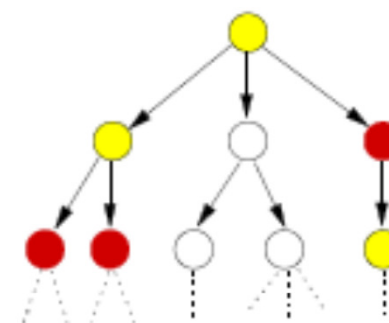
- Intuitive semantics:
 - of quantifiers (A/E) and temporal operators (F/G/U)



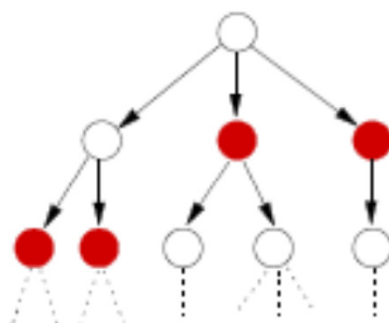
EF red



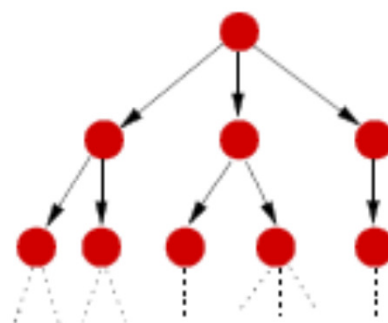
EG red



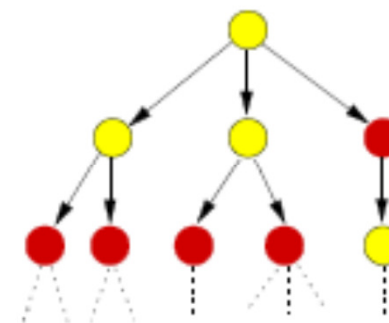
E [yellow U red]



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

PCTL syntax

- PCTL syntax:

– $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg \phi \mid P_{\sim p} [\psi]$ (state formulas)

ψ is true with probability $\sim p$

– $\psi ::= X \phi \mid \phi U^{\leq k} \phi \mid \phi U \phi$ (path formulas)

“next”

“bounded until”

“until”

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

- A PCTL formula is always a state formula

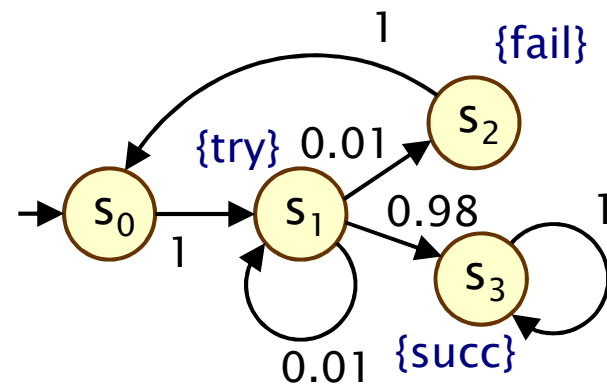
- path formulas only occur inside the P operator

PCTL semantics for DTMCs

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

- Examples

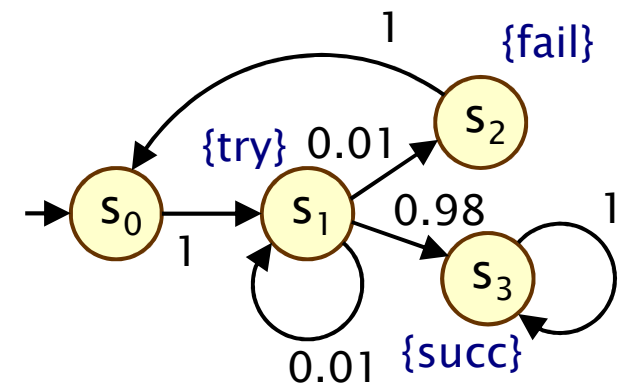
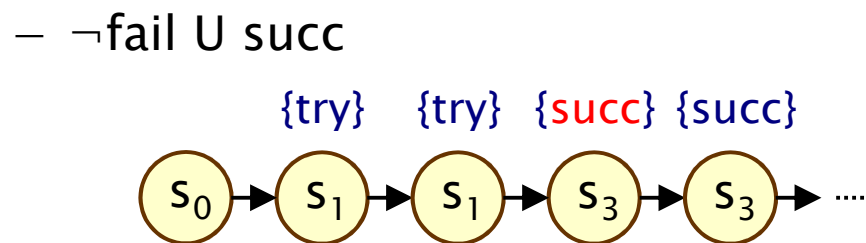
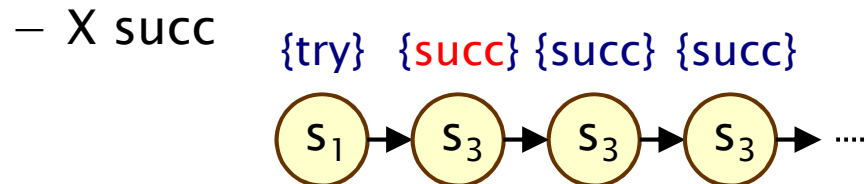
- $s_3 \models \text{succ}$
- $s_1 \models \text{try} \wedge \neg \text{fail}$



PCTL semantics for DTMCs

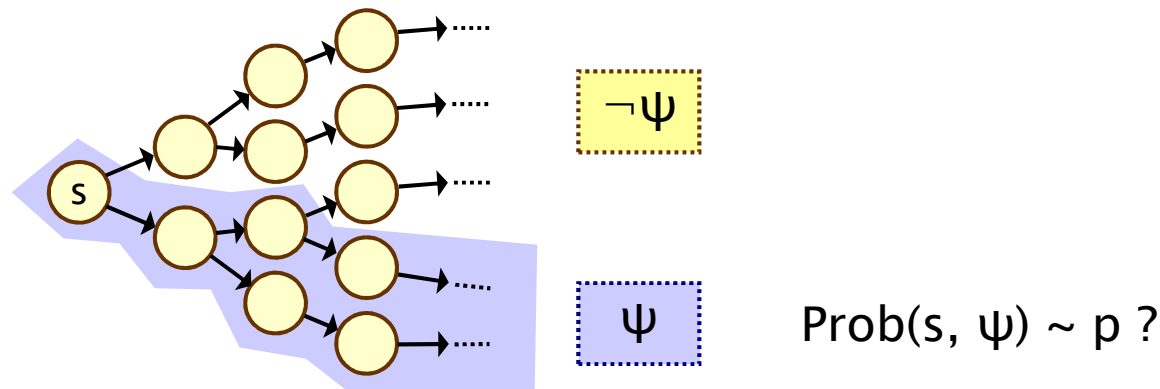
- Semantics of path formulas:
 - for a path $\omega = s_0 s_1 s_2 \dots$ in the DTMC:
 - $\omega \models X \phi \iff s_1 \models \phi$
 - $\omega \models \phi_1 U^{\leq k} \phi_2 \iff \exists i \leq k \text{ such that } s_i \models \phi_2 \text{ and } \forall j < i, s_j \models \phi_1$
 - $\omega \models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

- Some examples of satisfying paths:



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 $\sim p$ ”
 - example: $s \models P_{<0.25} [X \text{ fail}] \Leftrightarrow$ “the probability of atomic proposition fail being true in the next state of outgoing paths from s is less than 0.25”
 - formally: $s \models P_{\sim p} [\psi] \Leftrightarrow \text{Prob}(s, \psi) \sim p$
 - where: $\text{Prob}(s, \psi) = \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \}$
 - (sets of paths satisfying ψ are always measurable [Var85])



More PCTL...

- Usual temporal logic equivalences:

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

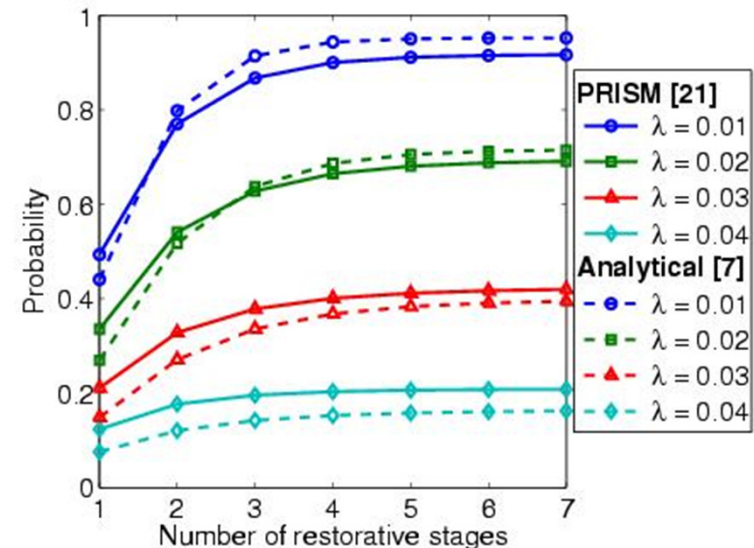
- Negation and probabilities

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

Quantitative properties

- Consider a PCTL formula $P_{\sim p} [\psi]$
 - 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 $P_{=?} [\psi]$
 - “**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
 - $P_{=?} [F \text{ err}/\text{total} > 0.1]$
 - “what is the probability that 10% of the NAND gate outputs are erroneous?”



Reachability and invariance

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

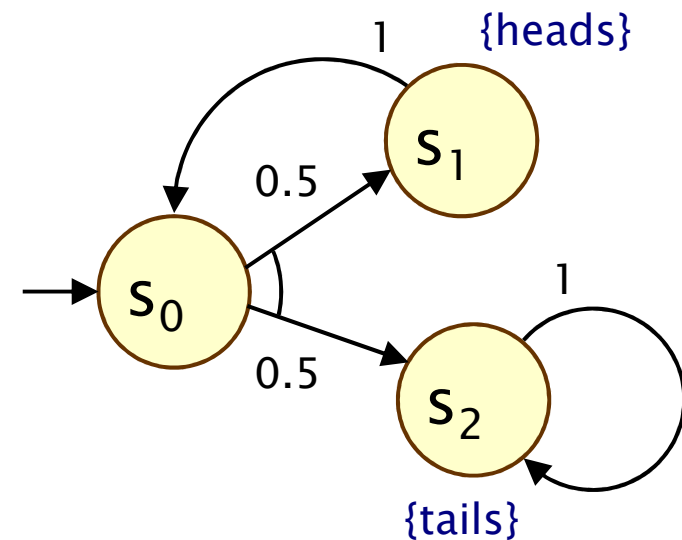
strictly speaking,
 $G \phi$ cannot be
derived from the
PCTL syntax in
this way since
there is no
negation of path
formulae

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
 - $P_{\sim p} [\psi]$ where p is either 0 or 1
- **Quantitative** PCTL properties
 - $P_{\sim p} [\psi]$ where p is in the range (0,1)
- $P_{>0} [F \phi]$ is identical to $EF \phi$
 - there exists a finite path to a ϕ -state
- $P_{\geq 1} [F \phi]$ is (similar to but) weaker than $AF \phi$
 - 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: $s_0s_1s_0s_1s_0s_1\dots$
- Does the probability of eventually throwing “tails” equal one?
 - PCTL: $P_{\geq 1} [F \text{ “tails” }]$
 - Result: **true**
 - Infinite path $s_0s_1s_0s_1s_0s_1\dots$ has **zero probability**



Overview (Part 1)

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

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 $P=? [F \text{ error}]$
 - e.g. compute result of $P=? [F^{\leq k} \text{ error}]$ for $0 \leq k \leq 100$

PCTL model checking for DTMCs

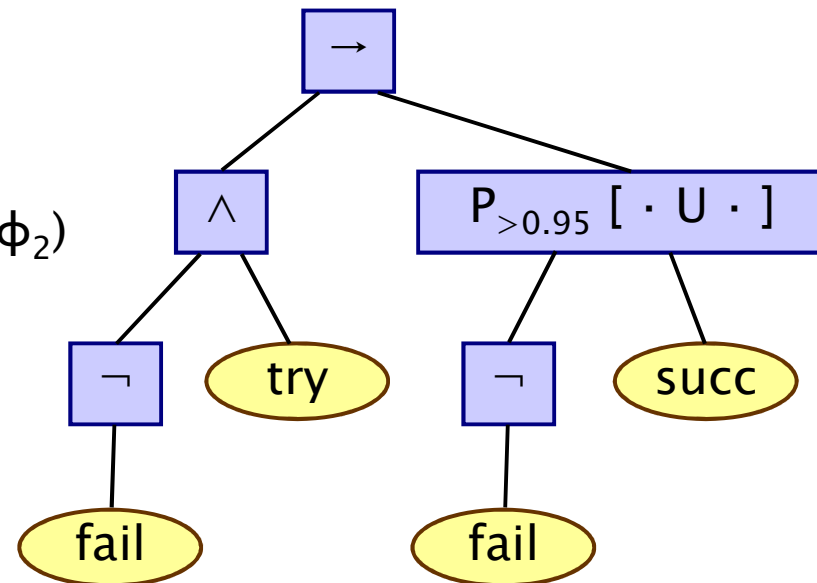
- Basic algorithm proceeds by induction on parse tree of ϕ
 - example: $\phi = (\neg \text{fail} \wedge \text{try}) \rightarrow P_{>0.95} [\neg \text{fail} \cup \text{succ}]$

- For the non-probabilistic operators:

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

- For the $P_{\sim p} [\psi]$ operator

- need to compute the probabilities $\text{Prob}(s, \psi)$ for all states $s \in S$
- focus here on “until” case: $\psi = \phi_1 \cup \phi_2$

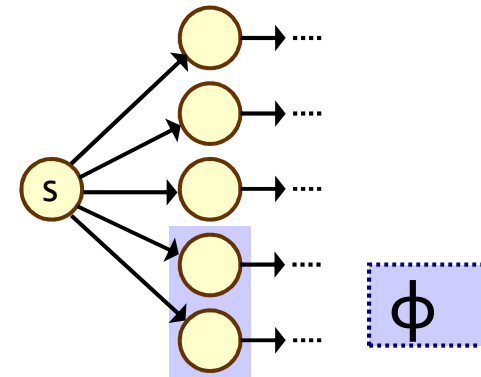


Probability computation

- 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

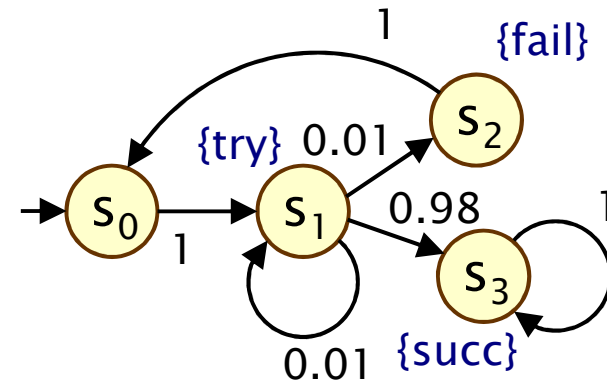
- Computation of probabilities for PCTL next operator
 - $\text{Sat}(P_{\sim p}[X \phi]) = \{ s \in S \mid \text{Prob}(s, X \phi) \sim p \}$
 - need to compute $\text{Prob}(s, X \phi)$ for all $s \in S$
- Sum outgoing probabilities for transitions to ϕ -states
 - $\text{Prob}(s, X \phi) = \sum_{s' \in \text{Sat}(\phi)} P(s, s')$
- Compute vector $\text{Prob}(X \phi)$ of probabilities for all states s
 - $\text{Prob}(X \phi) = P \cdot \underline{\phi}$
 - 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:** $P_{\geq 0.9} [X (\neg \text{try} \vee \text{succ})]$
 - $\text{Sat} (\neg \text{try} \vee \text{succ}) = (S \setminus \text{Sat}(\text{try})) \cup \text{Sat}(\text{succ})$
 $= (\{s_0, s_1, s_2, s_3\} \setminus \{s_1\}) \cup \{s_3\} = \{s_0, s_2, s_3\}$
 - $\text{Prob}(X (\neg \text{try} \vee \text{succ})) = \mathbf{P} \cdot \underline{(\neg \text{try} \vee \text{succ})} = \dots$

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

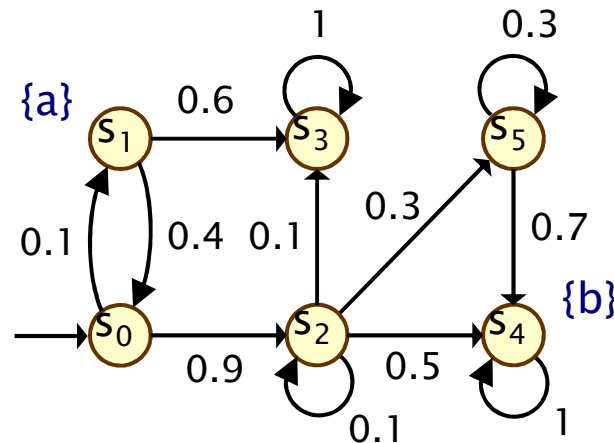


- **Results:**
 - $\text{Prob}(X (\neg \text{try} \vee \text{succ})) = [0, 0.99, 1, 1]$
 - $\text{Sat}(P_{\geq 0.9} [X (\neg \text{try} \vee \text{succ})]) = \{s_1, s_2, s_3\}$

PCTL until for DTMCs

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

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



Precomputation

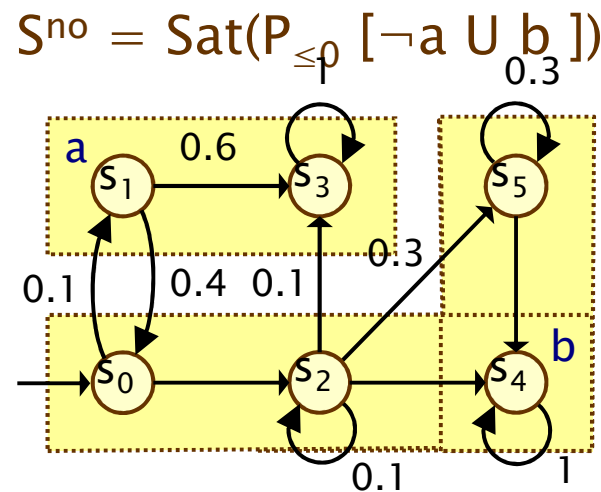
- We refer to the first phase (identifying sets S^{yes} and S^{no}) as “precomputation”
 - two algorithms: Prob0 (for S^{no}) and Prob1 (for S^{yes})
 - algorithms work on underlying graph (probabilities irrelevant)
- Important for several reasons
 - ensures **unique** solution to linear equation system
 - 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 $S^{no} = \text{Sat}(P_{\leq 0} [\phi_1 \cup \phi_2])$:
 - first compute $\text{Sat}(P_{>0} [\phi_1 \cup \phi_2]) \equiv \text{Sat}(E[\phi_1 \cup \phi_2])$
 - 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

Example:

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



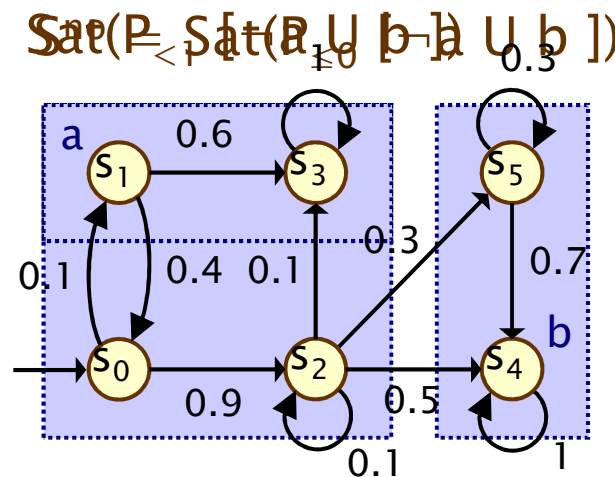
$\text{Sat}(B)_{>0} [\neg a \cup b]$

Precomputation – Prob1

- Prob1 algorithm to compute $S^{\text{yes}} = \text{Sat}(P_{\geq 1} [\phi_1 \cup \phi_2])$:
 - first compute $\text{Sat}(P_{< 1} [\phi_1 \cup \phi_2])$, 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:

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



$S^{\text{yes}} =$

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

PCTL until – linear equations

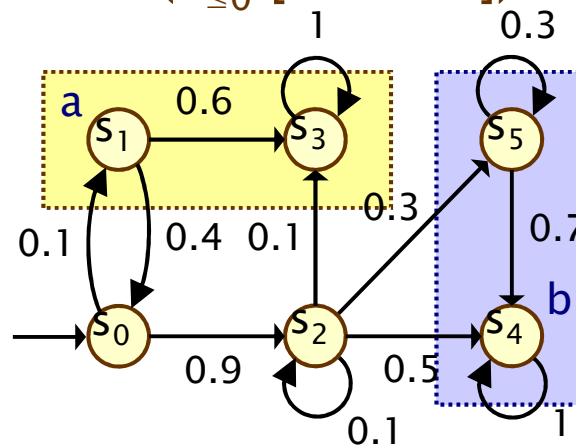
- Probabilities $\text{Prob}(s, \phi_1 \cup \phi_2)$ can now be obtained as the unique solution of the following set of **linear equations**
 - essentially the same as for probabilistic reachability

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

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

PCTL until – linear equations

- Example: $P_{>0.8} [\neg a \text{ U } b]$ $S^{\text{no}} =$
- Let $x_i = \text{Prob}(s_i, \neg a \text{ U } b)$ $\text{Sat}(P_{\leq 0} [\neg a \text{ U } b])$



$S^{\text{yes}} =$

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

$$x_1 = x_3 = 0$$

$$x_4 = x_5 = 1$$

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

$$x_0 = 0.1x_1 + 0.9x_2 = 0.8$$

$$\text{Prob}(\neg a \text{ U } b) = \underline{x} = [0.8, 0, 8/9, 0, 1, 1]$$

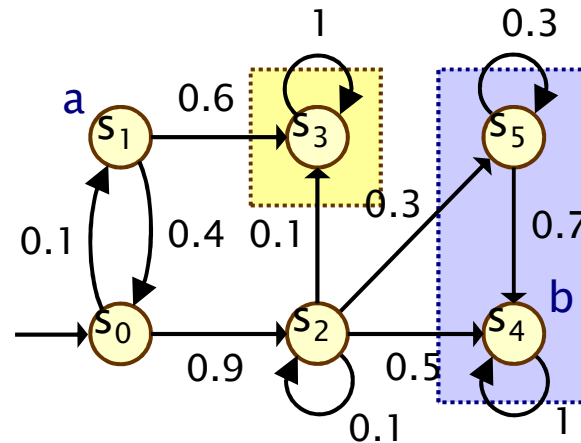
$$\text{Sat}(P_{>0.8} [\neg a \text{ U } b]) = \{s_2, s_4, s_5\}$$

PCTL Until – Example 2

- Example: $P_{>0.5} [G \neg b]$ $S^{\text{no}} = \text{Sat}(P_{\leq 0} [F b])$

- $\text{Prob}(s_i, G \neg b)$
 $= 1 - \text{Prob}(s_i, \neg(G \neg b))$
 $= 1 - \text{Prob}(s_i, F b)$

- Let $x_i = \text{Prob}(s_i, F b)$



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

$$x_3 = 0 \text{ and } x_4 = x_5 = 1$$

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

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

$$x_0 = 0.1x_1 + 0.9x_2 = 5/6 \text{ and } x_1 = 1/3$$

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

$$\text{Sat}(P_{>0.5} [G \neg b]) = \{ s_1, s_3 \}$$

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: $\mathbf{A} \cdot \underline{x} = \underline{b}$

- indexed over integers,
- i.e. assume $S = \{ 0, 1, \dots, |S|-1 \}$

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

PCTL model checking – Summary

- Computation of set $\text{Sat}(\Phi)$ for DTMC D and PCTL formula Φ
 - recursive descent of parse tree
 - combination of graph algorithms, numerical computation
- Probabilistic operator P :
 - $X \Phi$: one matrix–vector multiplication, $O(|S|^2)$
 - $\Phi_1 \cup^{\leq k} \Phi_2$: k matrix–vector multiplications, $O(k|S|^2)$
 - $\Phi_1 \cup \Phi_2$: linear equation system, at most $|S|$ variables, $O(|S|^3)$
- Complexity:
 - linear in $|\Phi|$ 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, $P_{\sim p} [\dots]$ 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

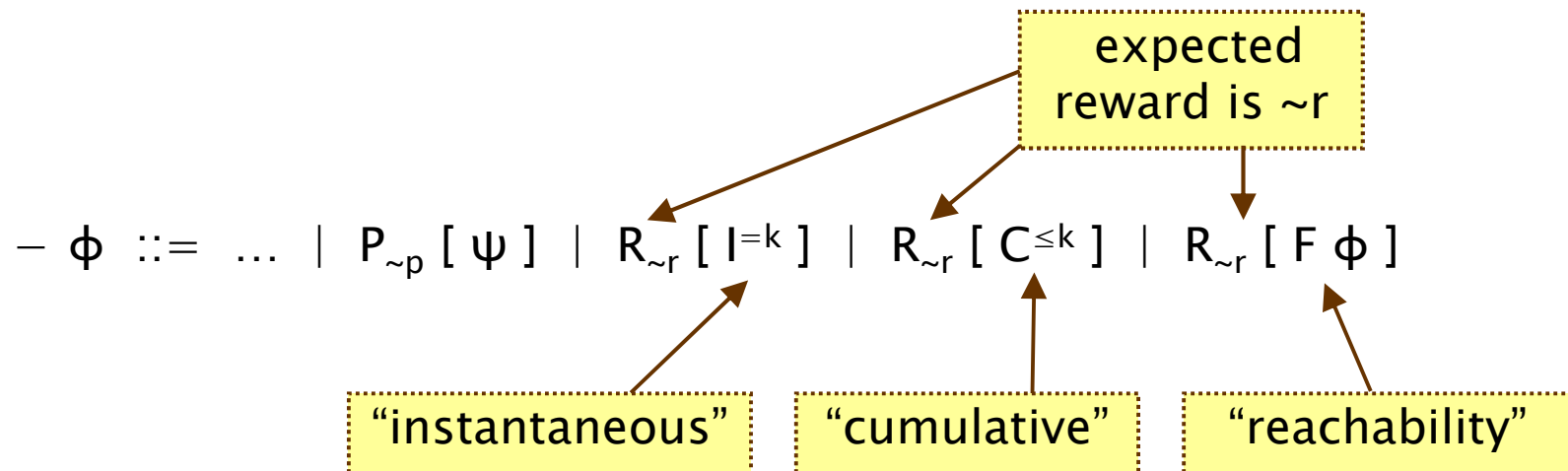
- Properties of DTMCs augmented with rewards
 - allow a wide range of quantitative measures of the system
 - basic notion: expected value of rewards
 - formal property specifications will be in an extension of PCTL
- More precisely, we use two distinct classes of property...
- **Instantaneous** properties
 - 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 $(S, s_{\text{init}}, P, L)$, a reward structure is a pair $(\underline{r}, \underline{t})$
 - $\underline{r} : S \rightarrow \mathbb{R}_{\geq 0}$ is the **state reward function** (vector)
 - $\underline{t} : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the **transition reward function** (matrix)
- Example (for use with instantaneous properties)
 - “size of message queue”: \underline{r} maps each state to the number of jobs in the queue in that state, \underline{t} is not used
- Examples (for use with cumulative properties)
 - “**time-steps**”: \underline{r} returns 1 for all states and \underline{t} is zero (equivalently, \underline{r} is zero and \underline{t} returns 1 for all transitions)
 - “**number of messages lost**”: \underline{r} is zero and \underline{t} maps transitions corresponding to a message loss to 1
 - “**power consumption**”: \underline{r} is defined as the per-time-step energy consumption in each state and \underline{t} 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



– 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 $\sim r$ ”

Types of reward formulas

- **Instantaneous:** $R_{\sim r} [I^k]$
 - “the expected value of the state reward at time-step k is $\sim r$ ”
 - 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 $\sim r$ ”
 - e.g. “the expected power consumption over one hour”
- **Reachability:** $R_{\sim r} [F \phi]$
 - “the expected reward cumulated before reaching a state satisfying ϕ is $\sim r$ ”
 - 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:
 - $s \models P_{\sim p} [\psi] \Leftrightarrow \Pr_s \{ \omega \in \text{Path}(s) \mid \omega \models \psi \} \sim p$
- For a state s in the DTMC (see [KNP07a] for full definition):
 - $s \models R_{\sim r} [I^k] \Leftrightarrow \text{Exp}(s, X_{I^k}) \sim r$
 - $s \models R_{\sim r} [C^{\leq k}] \Leftrightarrow \text{Exp}(s, X_{C^{\leq k}}) \sim r$
 - $s \models R_{\sim r} [F \Phi] \Leftrightarrow \text{Exp}(s, X_{F\Phi}) \sim r$

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

Reward formula semantics

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

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

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

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

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

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: $R_{\sim r} [F \phi]$
 - 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)

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

PRISM

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



PRISM...

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



PRISM modelling language

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

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



Simple example

dtmc

module M1

 x : [0..3] init 0;

 [a] x=0 -> (x' = 1);

 [b] x=1 -> 0.5 : (x' = 2) + 0.5 : (x' = 3);

endmodule

module M2

 y : [0..3] init 0;

 [a] y=0 -> (y' = 1);

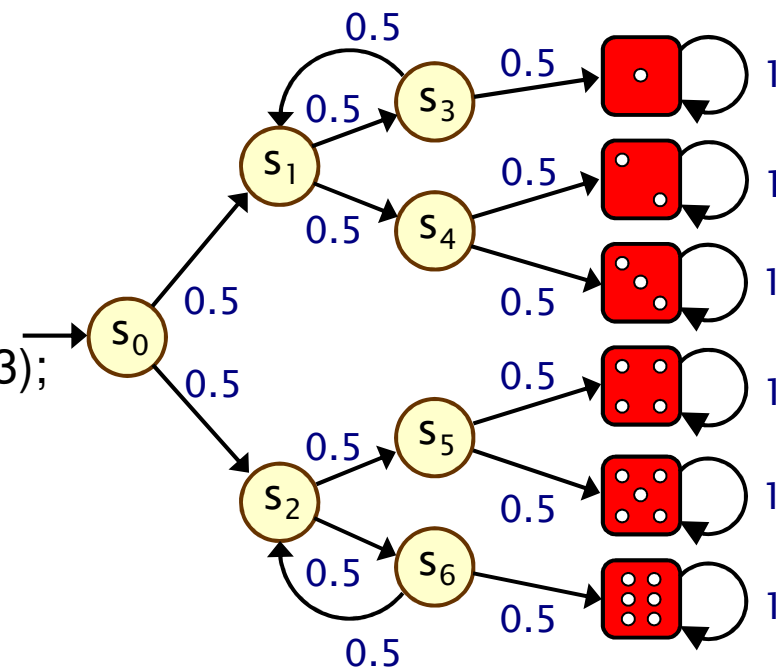
 [b] y=1 -> 0.4 : (y' = 2) + 0.6 : (y' = 3);

endmodule

Probabilistic models

```

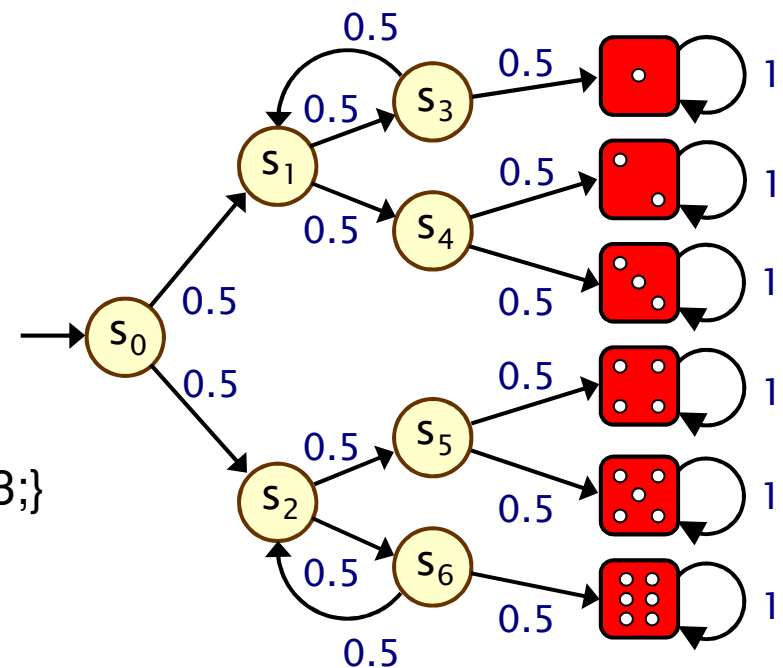
dtmc
module die
// local state  $s$  : [0..7] init 0;
// value of the dice  $d$  : [0..6] init 0;
[]  $s=0 \rightarrow 0.5 : (s'=1) + 0.5 : (s'=2)$ ;
...
[]  $s=3 \rightarrow$ 
     $0.5 : (s'=1) + 0.5 : (s'=7) \& (d'=1)$ ;
[]  $s=4 \rightarrow$ 
     $0.5 : (s'=7) \& (d'=2) + 0.5 : (s'=7) \& (d'=3)$ ;
...
[]  $s=7 \rightarrow (s'=7)$ ;
endmodule
rewards "coin_flips"
[]  $s < 7 : 1$ ;
endrewards
    
```



Given in PRISM's guarded commands modelling notation

Probabilistic models

```
int s, d;  
s = 0; d = 0;  
while (s < 7) {  
  bool coin = Bernoulli(0.5);  
  if (s = 0)  
    if (coin) s = 1 else s = 2;  
  ...  
  else if (s = 3)  
    if (coin) s = 1 else {s = 7; d = 1;}  
  else if (s = 4)  
    if (coin) {s = 7; d = 2} else {s = 7; d = 3;}  
  ...  
}  
return (d)
```



Given as a probabilistic program

Costs and rewards

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

Rewards in the PRISM language

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

(instantaneous, state rewards)

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

(cumulative, state rewards)

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

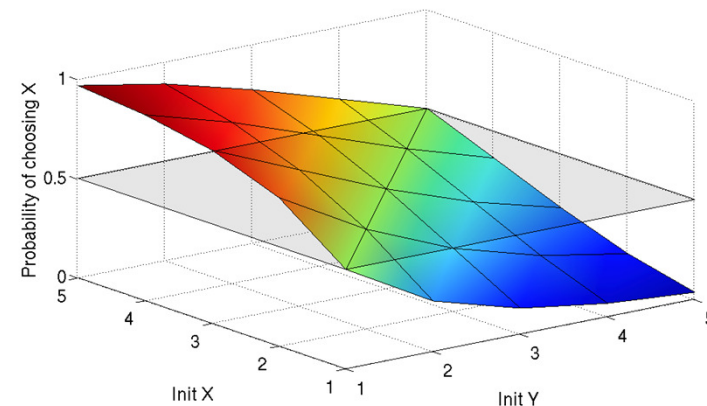
(cumulative, transition rewards)
(q = queue size, q_max = max.
queue size, receive = action label)

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

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

PRISM – Property specification

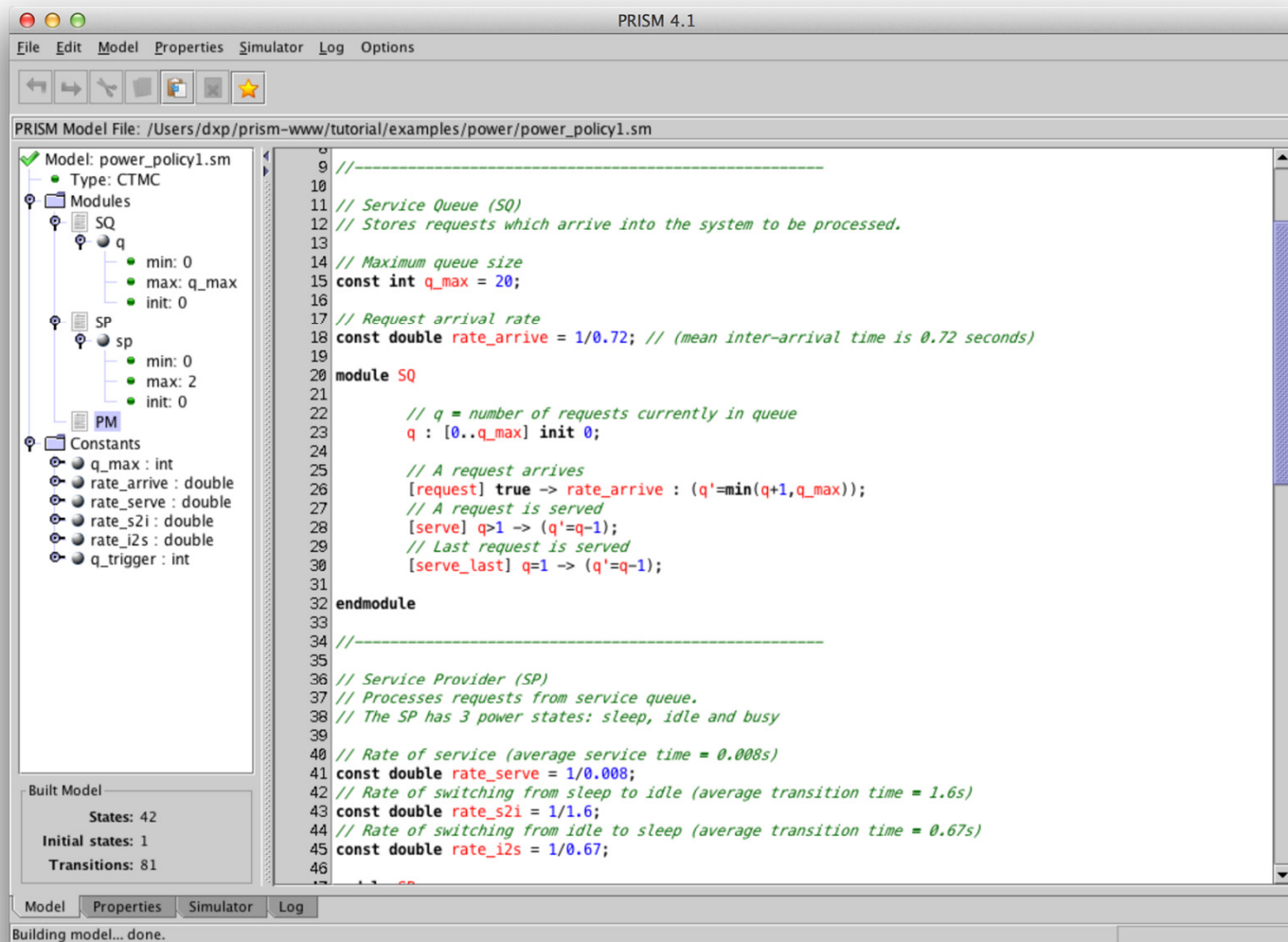
- **Temporal logic**–based property specification language
 - subsumes PCTL, CSL, probabilistic LTL, PCTL*, ...
- Simple examples:
 - $P_{\leq 0.01} [F \text{ “crash” }]$ – “the probability of a crash is at most 0.01”
 - $S_{>0.999} [\text{“up”}]$ – “long-run probability of availability is >0.999 ”
-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



PRISM – Property specification

- Properties can combine **numerical** + **exhaustive** aspects
 - $P_{\max=?} [F^{\leq 10} \text{“fail”}]$ – “worst-case probability of a failure occurring within 10 seconds, for any possible scheduling of system components”
 - $P_{=?} [G^{\leq 0.02} !\text{“deploy”} \{ \text{“crash”} \}^{\max}]$ – “the maximum probability of an airbag failing to deploy within 0.02s, from any possible crash scenario”
- **Reward**-based properties (**rewards** = **costs** = **prices**)
 - $R_{\{\text{“time”}\}=?} [F \text{“end”}]$ – “expected algorithm execution time”
 - $R_{\{\text{“energy”}\}^{\max=?} [C^{\leq 7200}]$ – “worst-case expected energy consumption during the first 2 hours”
- Properties can be combined with e.g. **arithmetic** operators
 - e.g. $P_{=?} [F \text{fail}_1] / P_{=?} [F \text{fail}_{\text{any}}]$ – “conditional failure prob.”

PRISM GUI: Editing a model



PRISM GUI: The Simulator

PRISM 4.1

File Edit Model Properties Simulator Log Options

Automatic exploration: Simulate Steps: 1

Backtracking: Backtrack Steps: 1

Manual exploration:

Module/[action]	Rate	Update
Left	0.006	left_n' = 2
Right	0.002	right_n' = 0
Line	2.0E-4	line_n' = false
ToLeft	2.5E-4	toleft_n' = false
[startLeft]	10.0	left' = true, r' = true

☒ Generate time automatically

State labels: Path formulae: Path information

- init
- deadlock
- minimum
- premium

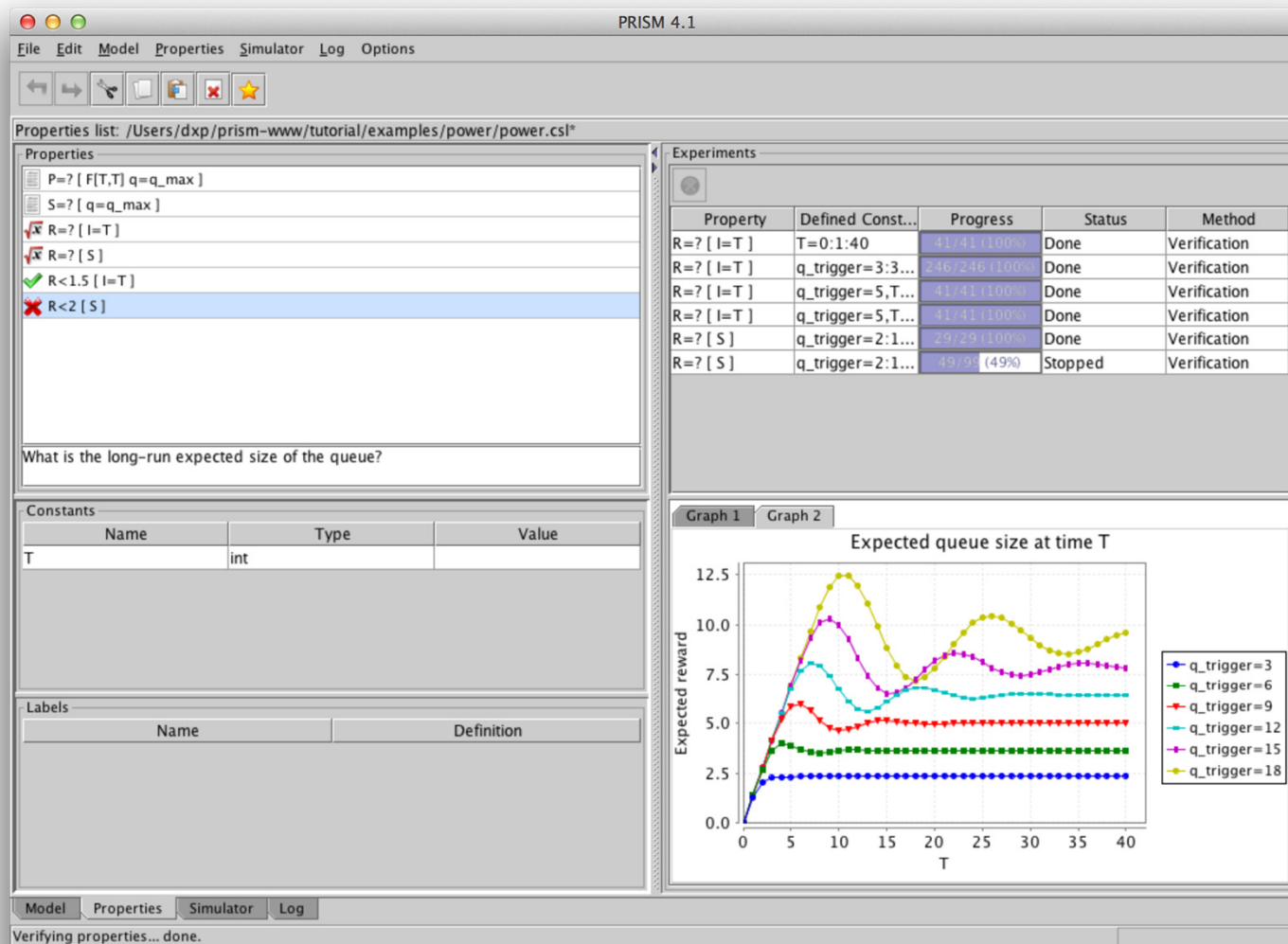
Path

Step	Time	Left	Right	Repair...	Line	ToLeft	ToRight	Rewards								
Action	#	Time (+)	left_n	left	right_n	right	r	line	line_n	toleft	toleft_n	toright	toright_n	perce...	time...	num...
	0	0	5	false	5	false	false	false	true	false	true	false	true	100	0	0
Right	1	12.0649			4									90		
ToRight	2	12.0806											false			
[startRight]	3	12.1674				true	true									1
[repairRight]	4	12.2677			5	false	false							100		0
Left	5	12.2809	4											90		
Left	6	12.3071	3											80		
Left	7	12.3446	2											70	1	
Left	8	12.3653	1											60		
Right	9	12.4059			4									50		
[startLeft]	10	12.4583		true			true									1
[repairLeft]	11	15.6657	2	false			false							60		0
[startLeft]	12	15.6834		true			true									1
[repairLeft]	13	15.7585	3	false			false							70	0	0
Right	14	15.8505			3									60		
Right	15	15.874			2									50		
Right	16	15.9084	3	false	1	false	false	false	true	false	true	false	false	40	0	7

Model Properties Simulator Log

Loading model... done.

PRISM GUI: Model checking and graphs



PRISM – Case studies

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

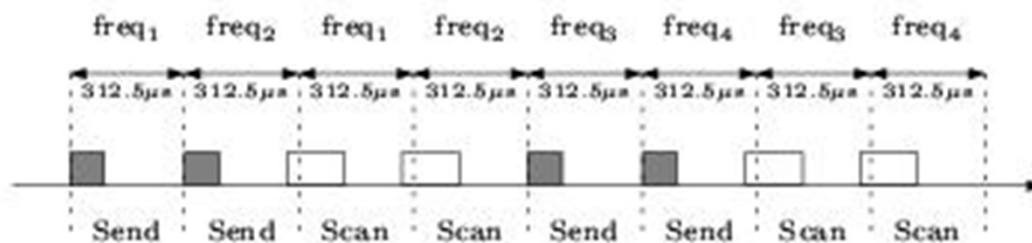
Case study: Bluetooth device discovery

- **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

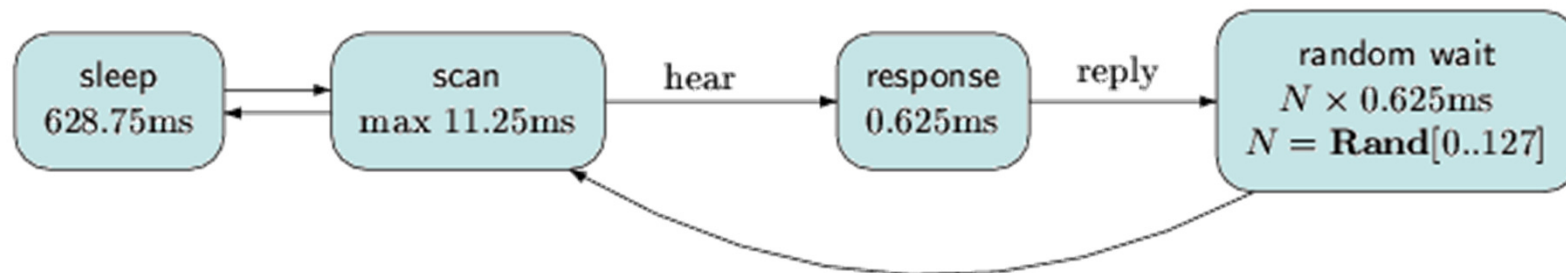
- 28 bit free-running clock **CLK**, ticks every 312.5µs
- Frequency hopping sequence determined by clock:
 - $\text{freq} = [\text{CLK}_{16-12} + k + (\text{CLK}_{4-2,0} - \text{CLK}_{16-12}) \bmod 16] \bmod 32$
 - 2 trains of 16 frequencies (determined by offset **k**), 128 times each, swap between 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	9	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	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

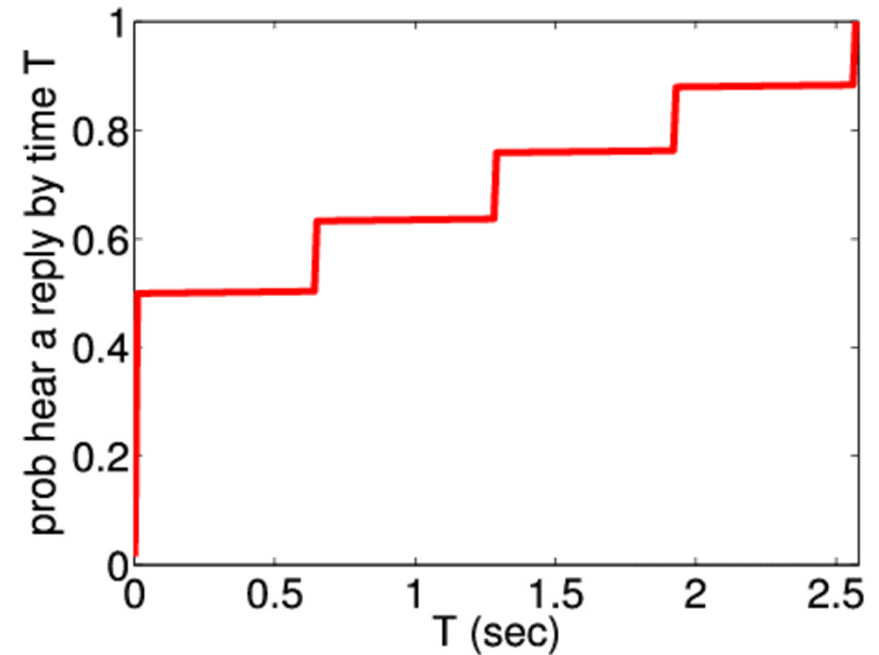
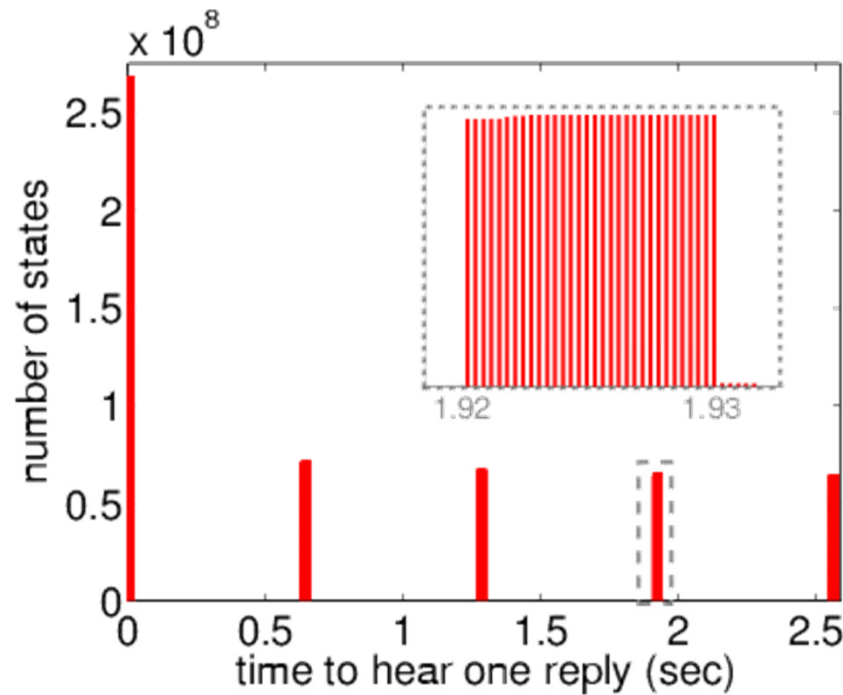
Bluetooth – PRISM model

- 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

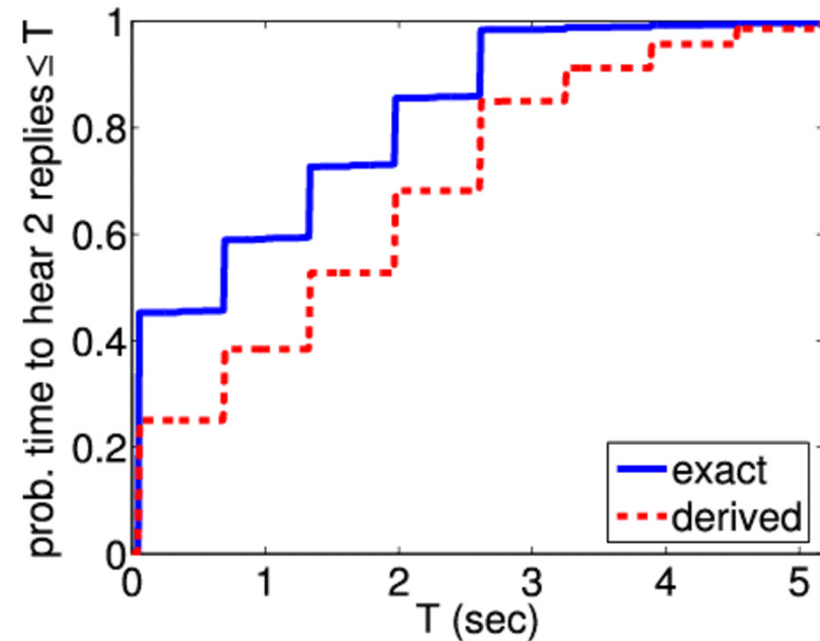
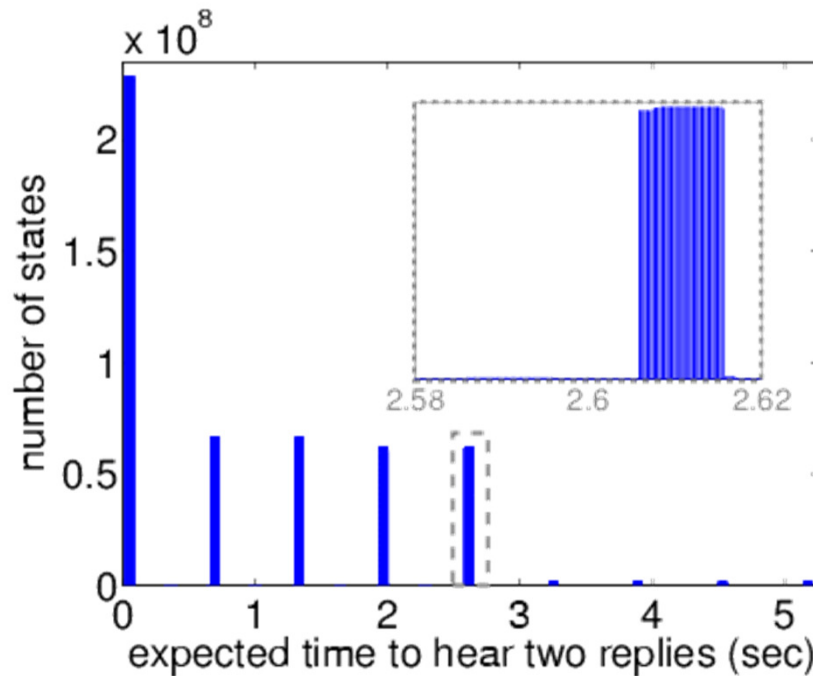
- Huge DTMC – initially, model checking infeasible
 - partition into 32 scenarios, i.e. 32 separate DTMCs
 - on average, approx. 3.4×10^9 states (536,870,912 initial)
 - can be built/analysed with PRISM's MTBDD engine
- We compute:
 - $R=? [F \text{ replies}=K \{ \text{"init"} \} \{ \text{max} \}]$
 - “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 μ s

Bluetooth – Time to hear 2 replies

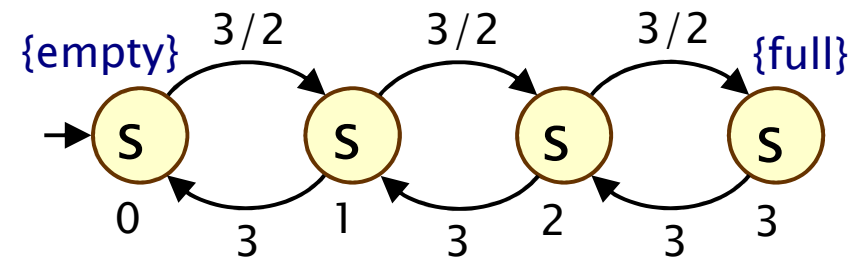


- Worst-case expected time = 5.177 sec
 - in 444 possible initial states
 - compare actual CDF with derived version which assumes times to reply to first/second messages are independent

Beyond DTMCs

- **Continuous-time Markov chains**

- transitions taken with real-valued rate (parameter of exponential distribution)
- suitable for reliability, availability, performance modelling

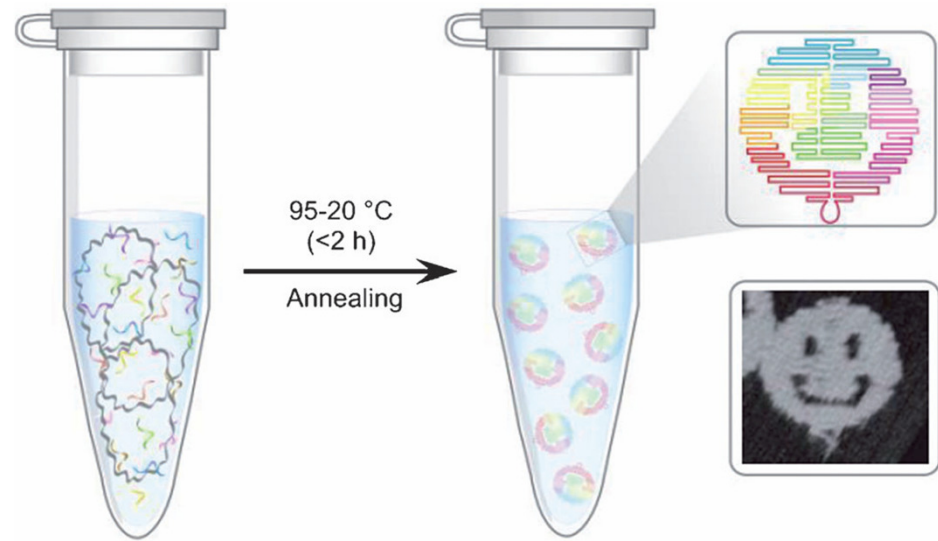


- **Temporal logic CSL – similar to PCTL, except real-valued time**
 - $P_{=?} [F^{[4,5.6]} \text{ outOfPower}]$ – the (transient) probability of being out of power in time interval of 4 to 5.6 time units
 - $S_{=?} [\text{minQoS}]$ – the steady-state probability of satisfying minimum QoS
 - $R_{<10} [C^{\leq 5}]$ – cumulated reward up to time 5 is less than 10
- **Model checking via discretisation (uniformisation)**

DNA programming



2nm

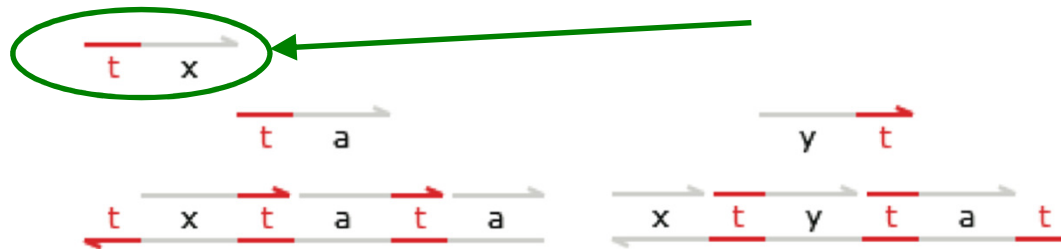


DNA origami

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

Case study: DNA programming

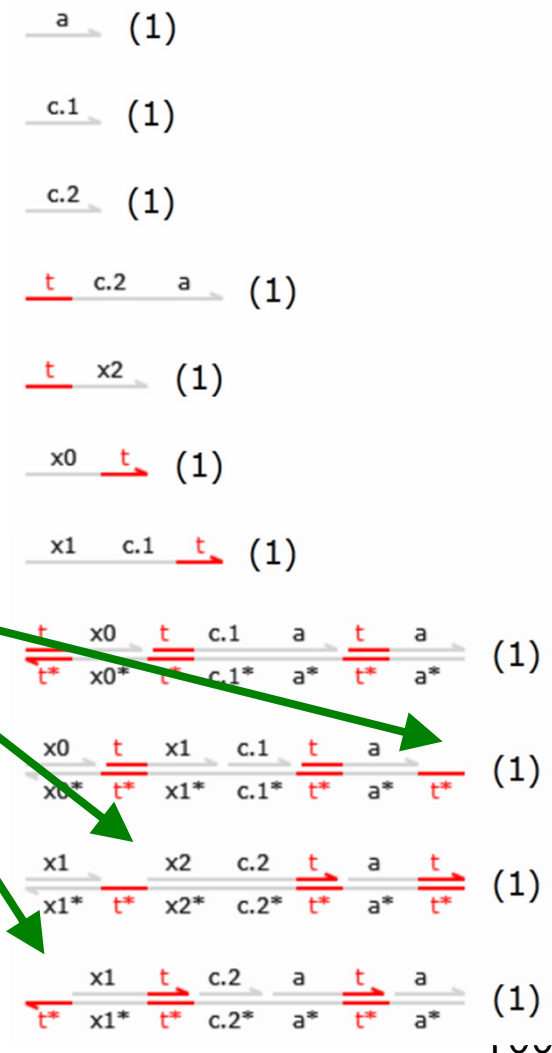
- DNA: easily accessible, cheap to synthesise information processing material
- DNA Strand Displacement language, induces CTMC models
 - for designing DNA circuits [Cardelli, Phillips, et al.]
 - accompanying software tool for analysis/simulation
 - now extended to include auto-generation of PRISM models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



- Formalising correctness: does it finish successfully?...
 - $A [G \text{"deadlock"} \Rightarrow \text{"all_done"}]$
 - $E [F \text{"all_done"}]$ (CTL, but probabilistic also...)

- ### Counterexample:

reactive gates



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 DNA computing
- Next: Markov decision processes (MDPs)