Probabilistic model checking

Marta Kwiatkowska

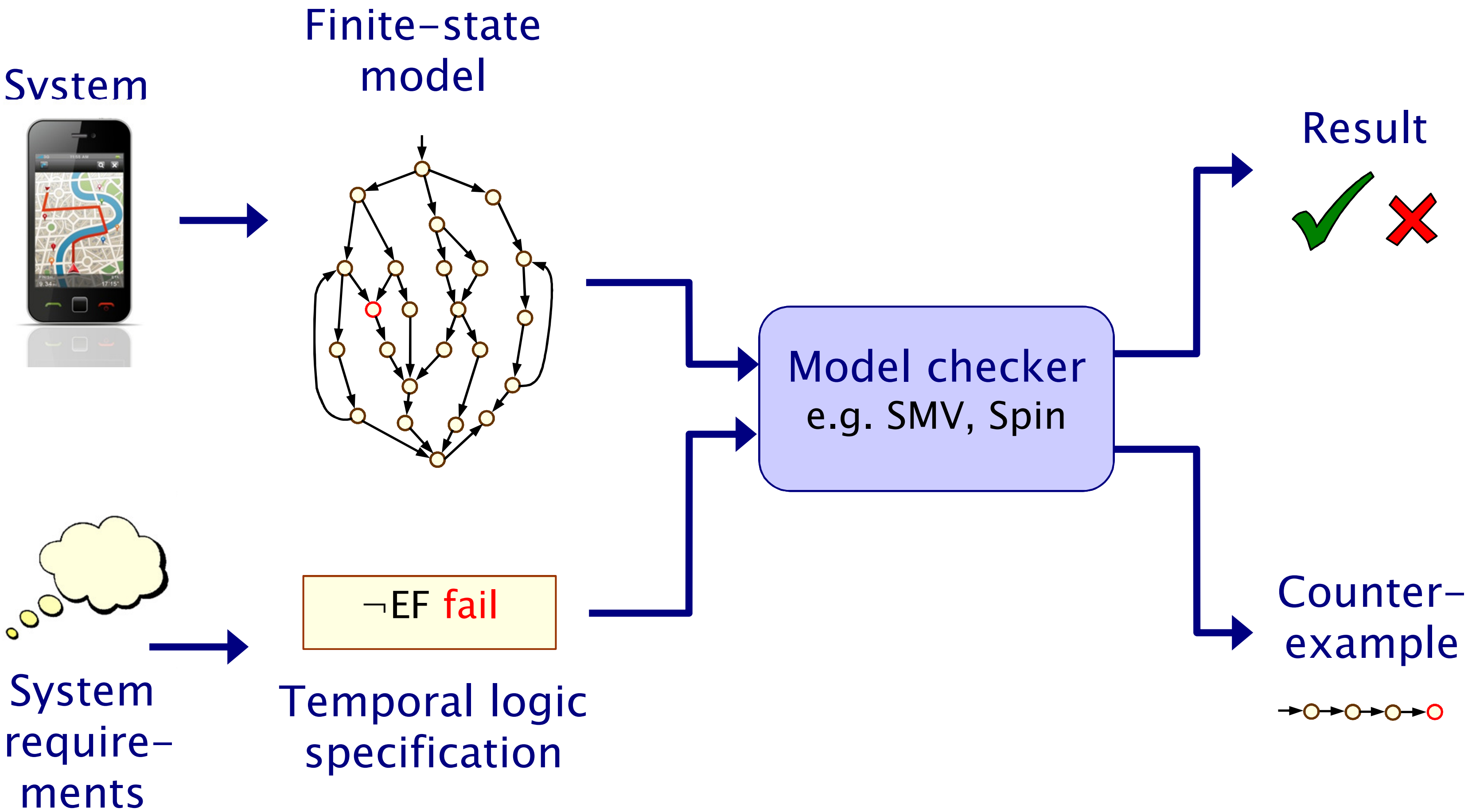
Department of Computer Science, University of Oxford

POPL 2015 tutorial, Mumbai, January 2015

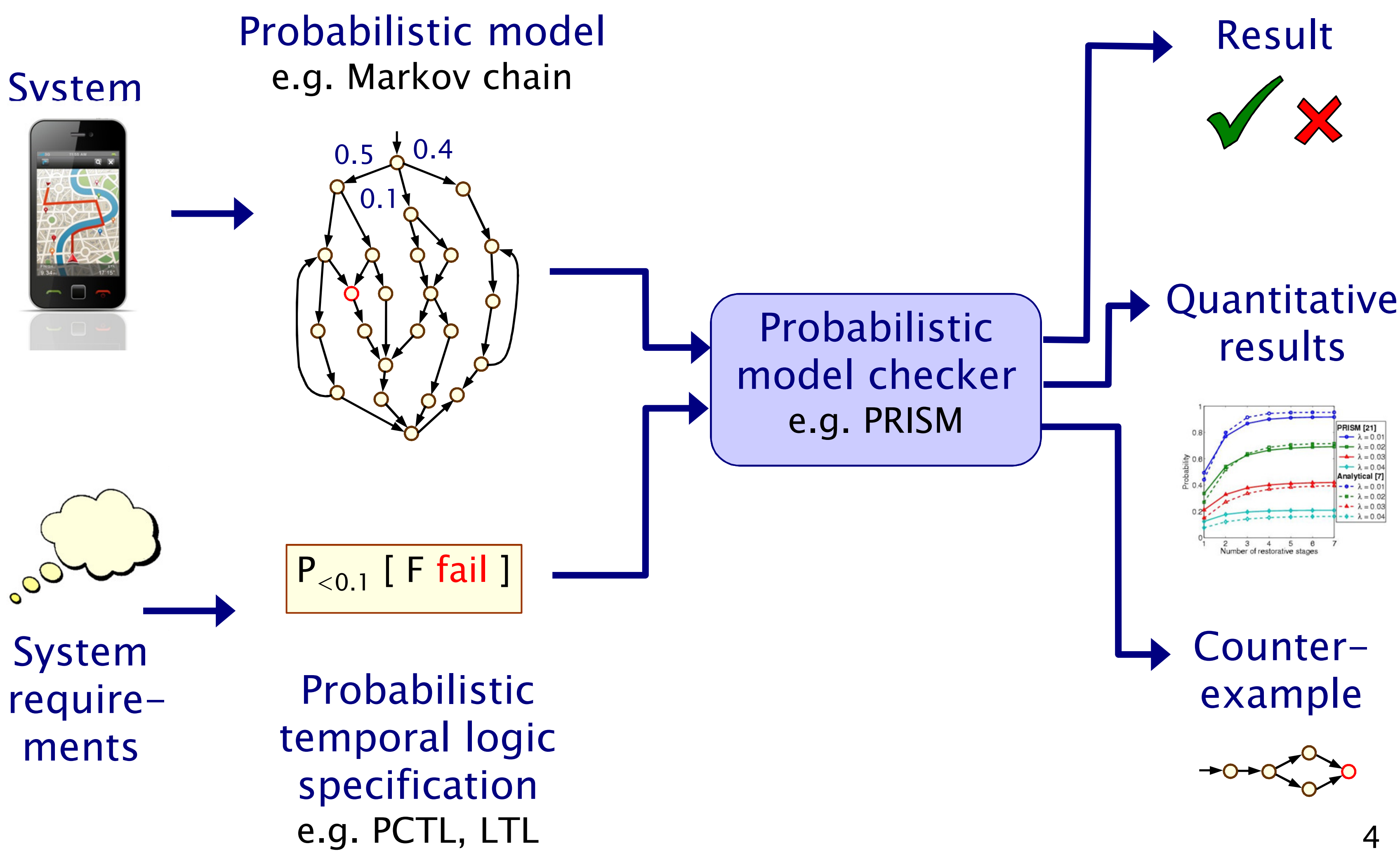
What is probabilistic model checking?

- Probabilistic model checking…
  - is model checking applied to probabilistic models
- Probabilistic models…
  - can be derived from high-level specification or extracted from probabilistic programs

Model checking



Probabilistic model checking



Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in wireless coordination protocols
  - as a symmetry breaker

    bool short_delay = Bernoulli(0.5) // short or long delay
- Modelling uncertainty
  - to quantify rate of failures

    bool fail = Bernoulli(0.001) // success wp 0.999 or failure
- Modelling performance and biological processes
  - reactions occurring between large numbers of molecules are naturally modelled in a stochastic fashion

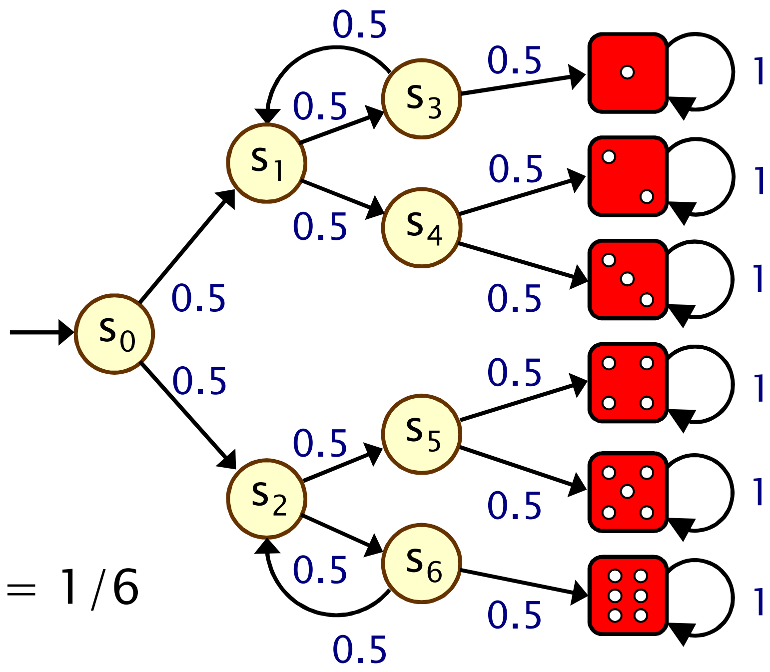
    float binding_rate = exp(2.5) // exponentially distributed

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

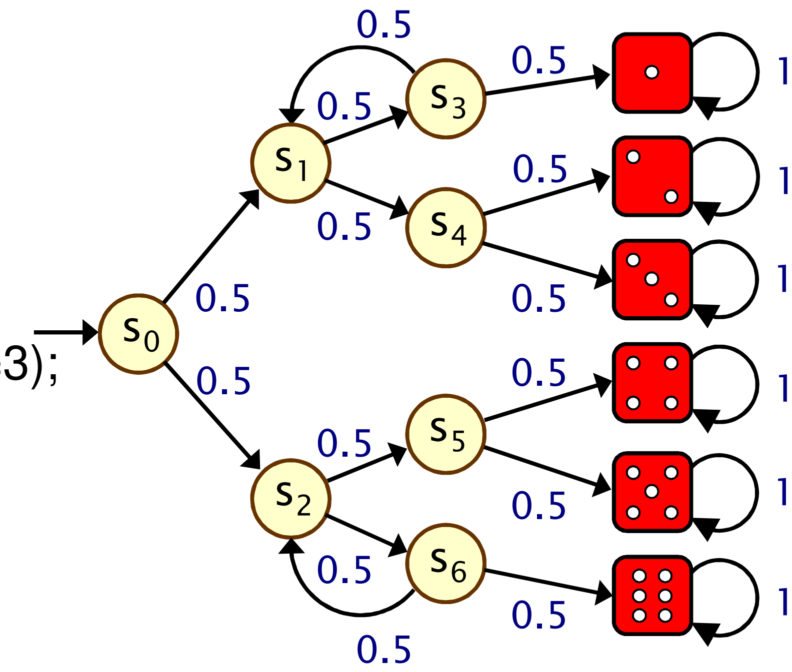
- Probability of obtaining a 4?
  - THH, TTTHH, TTTTTHH, …
  - Pr("eventually 4")

    $= (1/2)^3 + (1/2)^5 + (1/2)^7 + \ldots = 1/6$
  - expected number of coin flips needed = 11/3
  - NB termination guaranteed



Probabilistic models

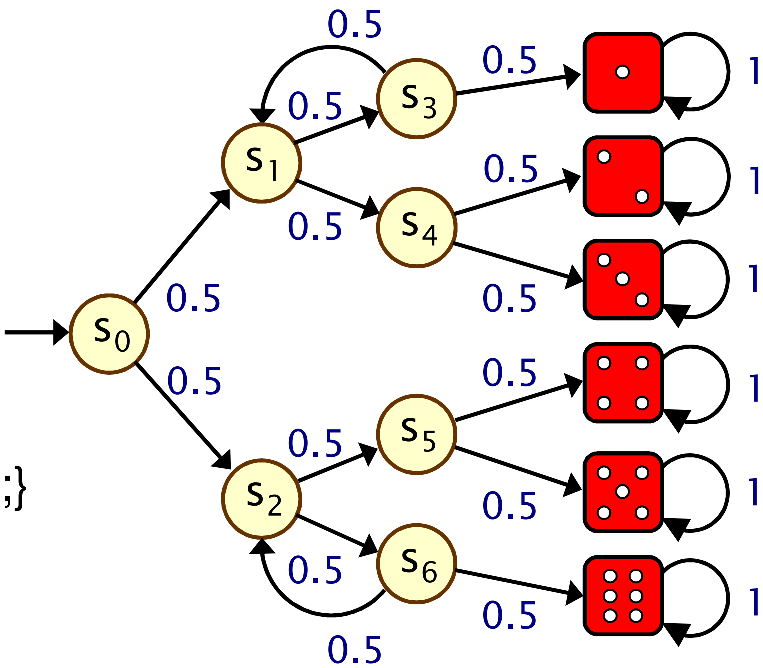
```
dtmc
module die
// local state s : [0..7] init 0;
// value of the dice d : [0..6] init 0;
[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
…
[] s=3 ->
  0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
[] s=4 ->
  0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
…
[] s=7 -> (s'=7);
endmodule
rewards "coin_flips"
[] s<7 : 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 (loopy) probabilistic program

Relation to programming languages

- Probabilistic model checking (PMC)
  - probabilistic models, state based, where transition relation is probabilistic
  - nonterminating behaviour
  - focus on computing probability or expectation of an event, or repeated events, typically via numerical methods
  - considers models with nondeterminism
- Probabilistic programming (PP)
  - imperative or functional programming extended with random assignment, interpreted as distribution transformers
  - terminating behaviour
  - focus on probabilistic inference (computing representation of the denoted probability distribution), typically via sampling
  - no nondeterminism, but conditioning on observations

PMC vs PP

- Excellent potential for cross-fertilisation
  - PMC and PP different communities
  - yet shared models (Markov chains) and methods (symbolic MTBDD/ADD-based solvers)
- PMC: maturing field
  - variety of models, incl. nondeterministic, timed, hybrid, etc
  - good for compact model representations, efficient automata-based and controller synthesis methods
  - can benefit from machine learning, cf ATVA 2014
- PP: emerging field
  - variety of efficient sampling-based MC methods
  - good for representing and computing distributions
  - can benefit from nondeterminism, useful for under-specification and input nondeterminism

Probabilistic programming. Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, Sriram K. Rajamani. *Proc. FOSE 2014, pp 167-181.*

Outline

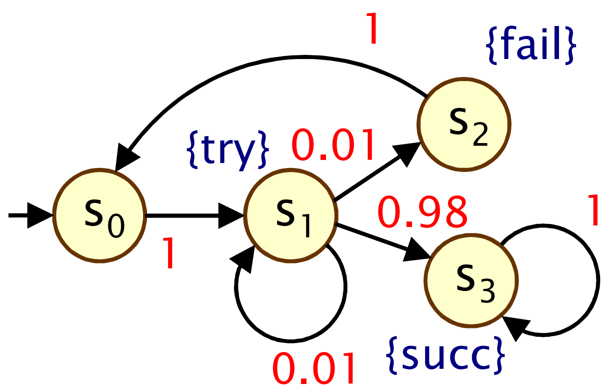
0. Motivation
1. Model checking for discrete-time Markov chains
   - Definition, paths & probability spaces
   - PCTL model checking
   - Costs and rewards
2. Model checking for Markov decision processes
   - Definition & adversaries
   - PCTL model checking
   - Note on LTL model checking
3. Probabilistic programs as Markov decision processes
   - How to verify probabilistic programs
4. PRISM
   - Functionality, supported models and logics
5. Summary and further reading

Part 1

Discrete-time Markov chains

Discrete-time Markov chains

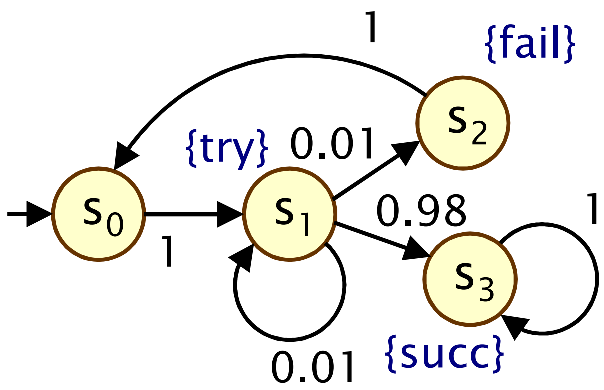
- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities
- States
  - discrete set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability of making transitions between states is given by discrete probability distributions



Discrete-time Markov chains

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

- Note: no deadlock states
  - i.e. every state has at least one outgoing transition
  - terminating behaviour represented by adding self loops



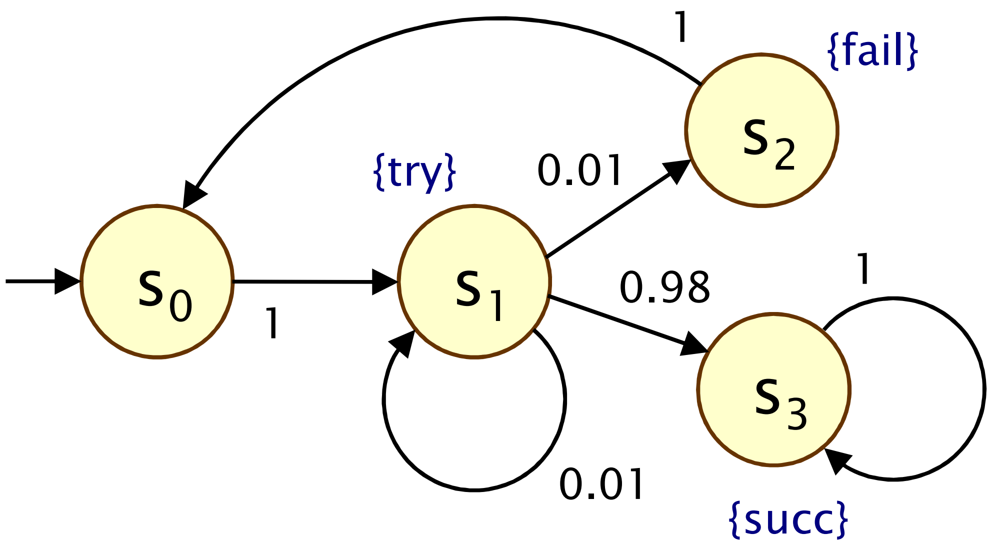
Simple DTMC example

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

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

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

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



DTMCs: An alternative definition

- Alternative definition… a DTMC is:
  - a family of random variables $\{ X(k) \mid k=0,1,2,\ldots \}$
  - 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)

 16

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

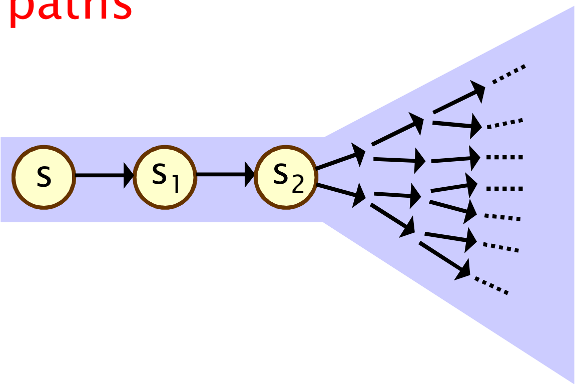
- Initial state $s_{init} \in S$ can be generalised…
  - to an initial probability distribution $s_{init} : S \rightarrow [0,1]$

- Transition probabilities are reals: $P(s,s') \in [0,1]$
  - but for algorithmic purposes, are assumed to be rationals

 17

Paths and probabilities

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



Probability space over paths

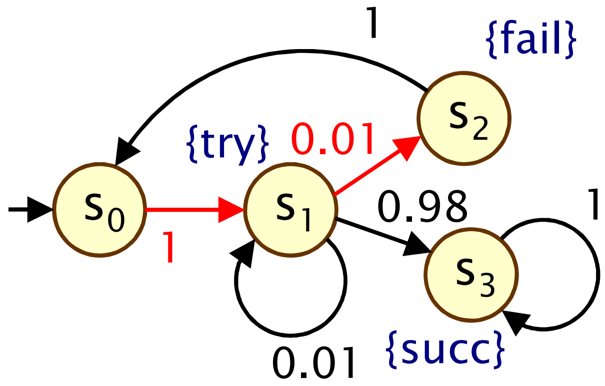
- Sample space Ω = Path(s)

  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
- Can also derive the probability space for finite and infinite sequences

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

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 $\rightarrow$ $P_{\geq 0.95}$ [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

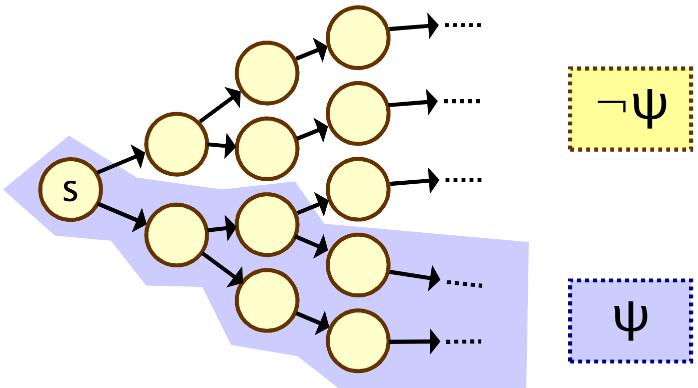
- PCTL formulas interpreted over states of a DTMC
  - $s \models \phi$ denotes ϕ is "true in state s" or "satisfied in state s"

- Semantics of (non-probabilistic) state formulas:
  - for a state s:
  - $s \models a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \models \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \models \phi_1$ and $s \models \phi_2$
  - $s \models \neg\phi \quad \Leftrightarrow \quad s \models \phi$ is false

- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$:
  - $\omega \models X\, \phi \quad \Leftrightarrow \quad s_1 \models \phi$
  - $\omega \models \phi_1\, U\, \phi_2 \quad \Leftrightarrow \quad \exists$ i such that $s_i \models \phi_2$ and $\forall j<i,\ s_j \models \phi_1$

PCTL semantics for DTMCs

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

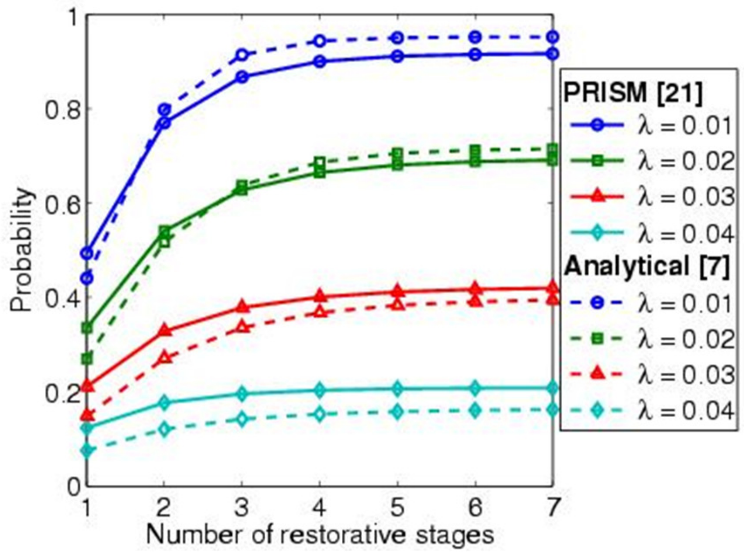


Prob(s, ψ) ~ p ?

Quantitative properties

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

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



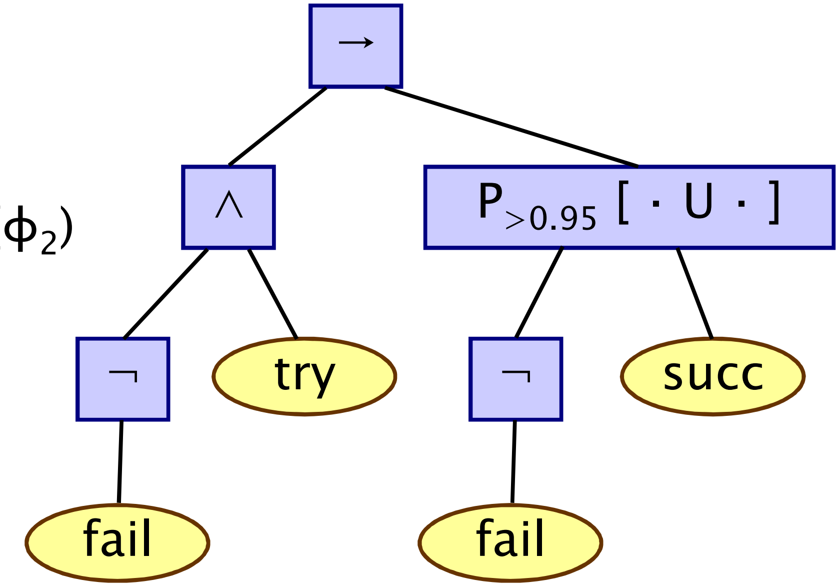
PCTL model checking for DTMCs

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

PCTL model checking for DTMCs

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

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

PCTL until for DTMCs

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

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

PCTL until – Linear equations

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

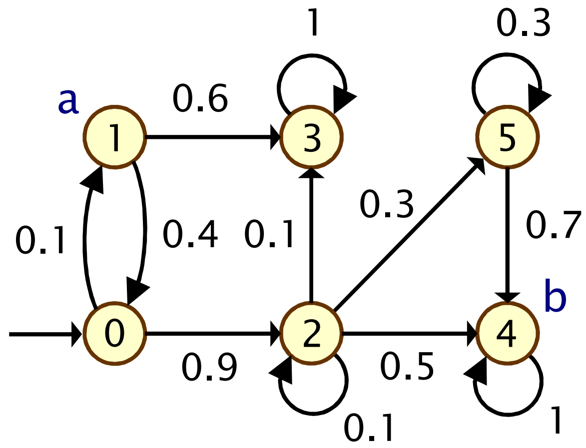
$$Prob(s, \phi_1 \text{ U } \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot Prob(s', \phi_1 \text{ U } \phi_2) & \text{otherwise} \end{cases}$$

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

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

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

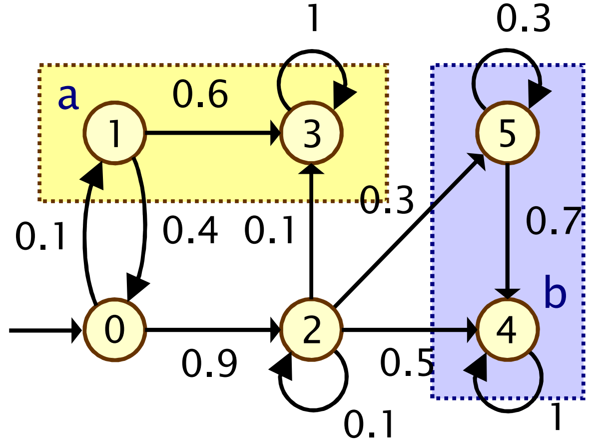


PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

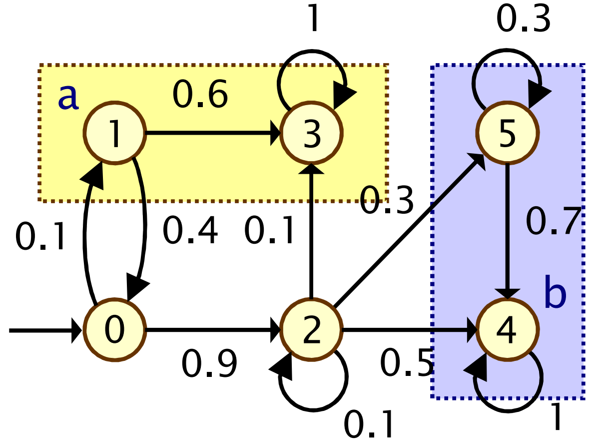
Sat($P_{\geq 1}$ [¬a U b])

PCTL until – Example

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

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

Sat($P_{\geq 1}$ [¬a U b])

$x_4 = x_5 = 1$

$x_1 = x_3 = 0$

$x_0 = 0.1x_1+0.9x_2 = 0.8$

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

<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 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:
  - $X\ \Phi$: 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|

Reward-based properties

- We augment DTMCs with rewards (or, conversely, costs)
  - real-valued quantities assigned to states and/or transitions
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards (or costs)
  - 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

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)

DTMC reward structures

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

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

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

PCTL and rewards

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

expected reward is ~r

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

"instantaneous" "cumulative" "reachability"

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

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

Reward formula semantics

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

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

- 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

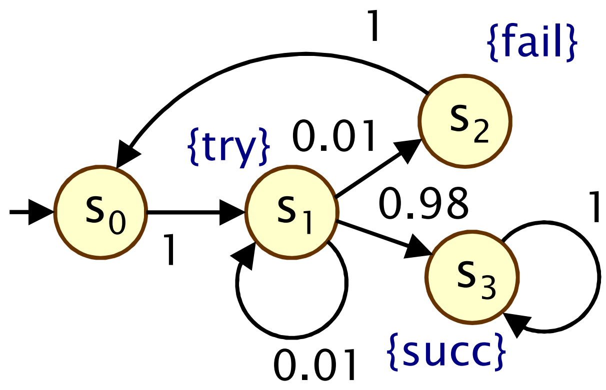
- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - variant of the method for computing bounded until probabilities (not discussed)
  - solution of recursive equations
- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to computing until probabilities
  - precomputation 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

Part 2

Markov decision processes

Recap: Discrete-time Markov chains

- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities
- Formally: DTMC $D = (S, s_{init}, P, L)$ where:
  - S is a set of states and $s_{init} \in S$ is the initial state
  - $P : S \times S \rightarrow [0,1]$ is the transition probability matrix
  - $L : S \rightarrow 2^{AP}$ labels states with atomic propositions
  - define a probability space Pr_s over paths $Path_s$



- Properties of DTMCs
  - can be captured by the logic PCTL
  - e.g. $send \rightarrow P_{\geq 0.95}$ [F deliver]
  - key question: what is the probability of reaching states $T \subseteq S$ from state s?
  - reduces to graph analysis + linear equation system

Nondeterminism

- Some aspects of a system may not be probabilistic and should not be modelled probabilistically; for example:

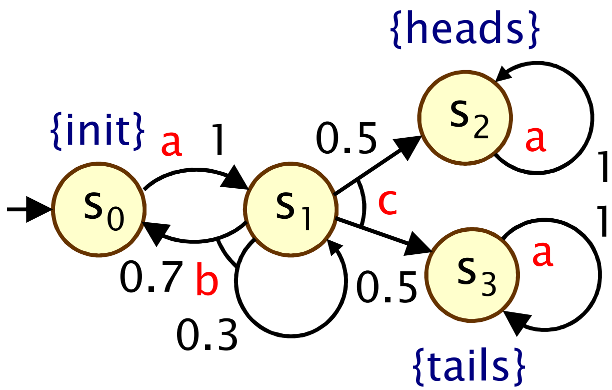
- Concurrency – scheduling of parallel components
  - e.g. randomised distributed algorithms – multiple probabilistic processes operating asynchronously

- Underspecification – unknown model parameters
  - e.g. a probabilistic communication protocol designed for message propagation delays of between d_{min} and d_{max}

- Unknown environments – unknown inputs
  - e.g. probabilistic security protocols – unknown adversary

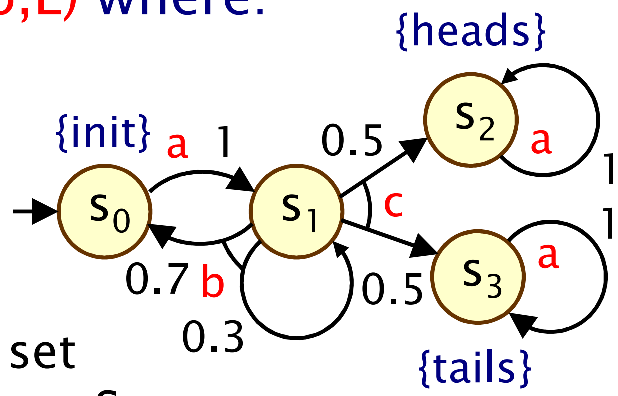
Markov decision processes

- Markov decision processes (MDPs)
  - extension of DTMCs which allow nondeterministic choice
- Like DTMCs:
  - discrete set of states representing possible configurations of the system being modelled
  - transitions between states occur in discrete time-steps
- Probabilities and nondeterminism
  - in each state, a nondeterministic choice between several discrete probability distributions over successor states



Markov decision processes

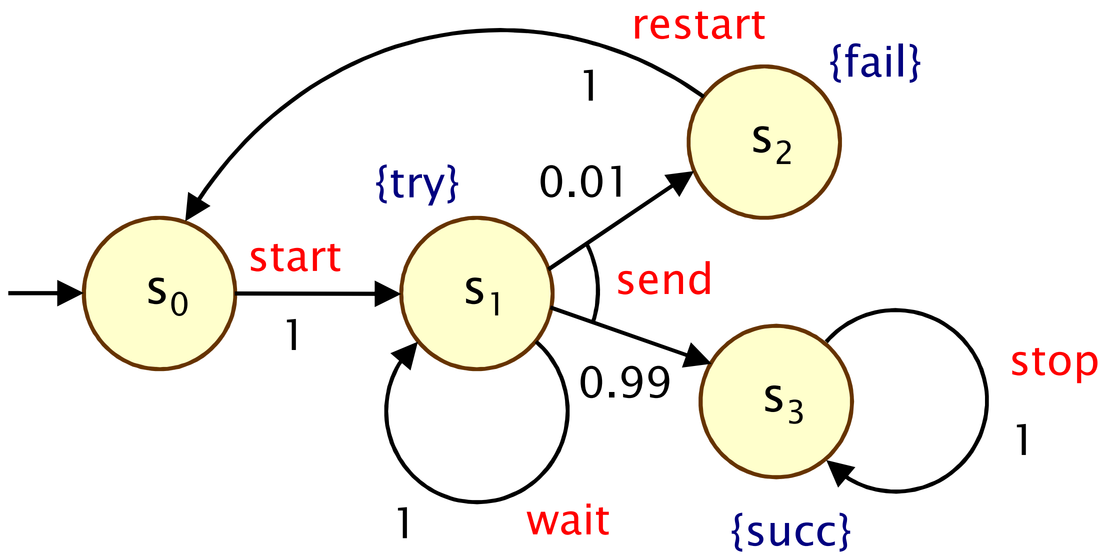
- Formally, an MDP M is a tuple $(S,s_{init},\alpha,\delta,L)$ where:
  - S is a set of states ("state space")
  - $s_{init} \in S$ is the initial state
  - α is an alphabet of action labels
  - $\delta \subseteq S \times \alpha \times Dist(S)$ is the transition probability relation, where Dist(S) is the set of all discrete probability distributions over S
  - $L : S \to 2^{AP}$ is a labelling with atomic propositions



- Notes:
  - we also abuse notation and use δ as a function
  - i.e. $\delta : S \to 2^{\alpha \times Dist(S)}$ where $\delta(s) = \{ (a,\mu) \mid (s,a,\mu) \in \delta \}$
  - we assume $\delta(s)$ is always non-empty, i.e. no deadlocks
  - MDPs, here, are identical to probabilistic automata [Segala]
    - usually, MDPs take the form: $\delta : S \times \alpha \to Dist(S)$

Simple MDP example

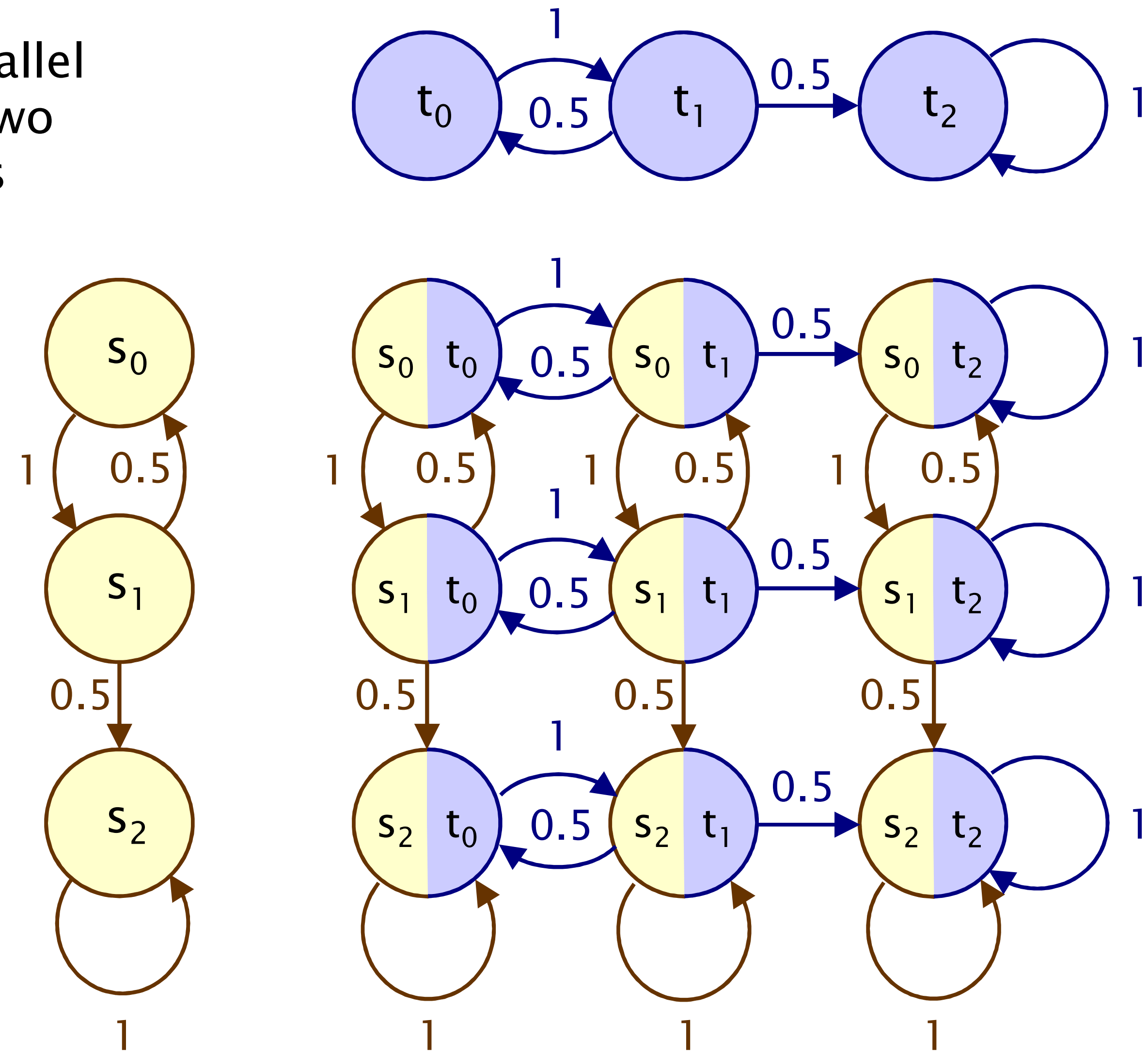
- A simple communication protocol
  - after one step, process starts trying to send a message
  - then, a nondeterministic choice between: (a) waiting a step because the channel is unready; (b) sending the message
  - if the latter, with probability 0.99 send successfully and stop
  - and with probability 0.01, message sending fails, restart



Example – Parallel composition

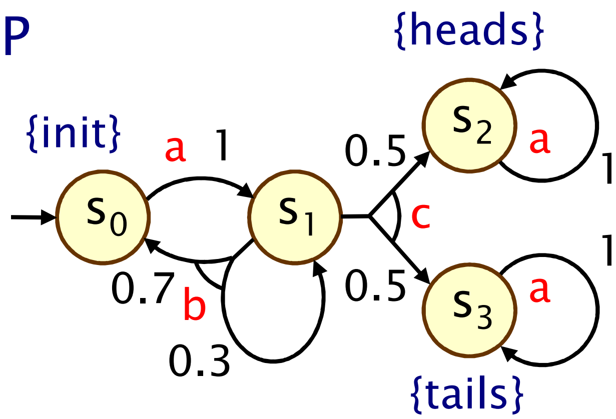
Asynchronous parallel composition of two 3-state DTMCs

Action labels omitted here



Paths and strategies

- A (finite or infinite) path through an MDP
  - is a sequence (s_0...s_n) of (connected) states
  - represents an execution of the system
  - resolves both the probabilistic and nondeterministic choices



- A strategy σ (aka. "adversary" or "policy") of an MDP
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions on action-distribution pairs
  - induces a fully probabilistic model
  - i.e. an (infinite-state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths

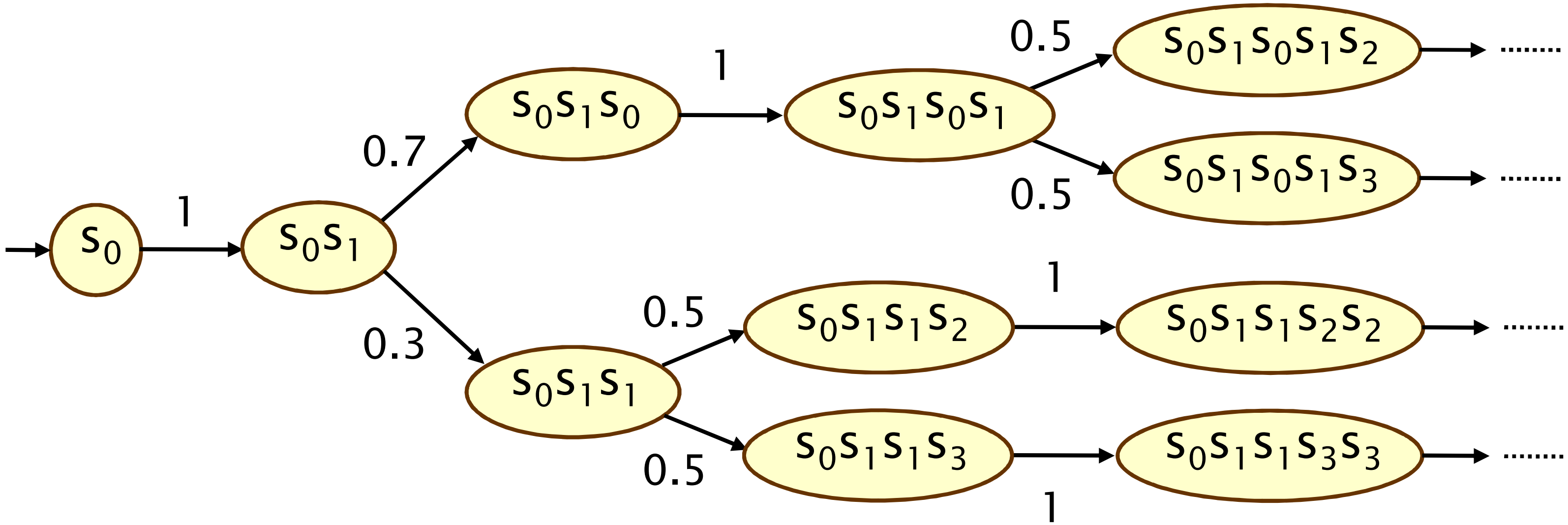
Classification of strategies

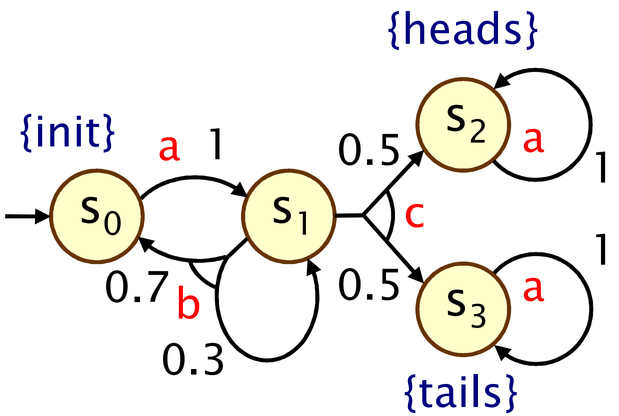
- Strategies are classified according to
- randomisation:
  - σ is deterministic (pure) if $\sigma(s_0 \ldots s_n)$ is a point distribution, and randomised otherwise
- memory:
  - σ is memoryless (simple) if $\sigma(s_0 \ldots s_n) = \sigma(s_n)$ for all $s_0 \ldots s_n$
  - σ is finite memory if there are finitely many modes such as $\sigma(s_0 \ldots s_n)$ depends only on s_n and the current mode, which is updated each time an action is performed
  - otherwise, σ is infinite memory

- A strategy σ induces, for each state s in the MDP:
  - a set of infinite paths $Path^{\sigma}(s)$
  - a probability space Pr^{σ}_{s} over $Path^{\sigma}(s)$

Example strategy

- Fragment of induced Markov chain for strategy which picks b then c in s_1





finite-memory,
deterministic

PCTL

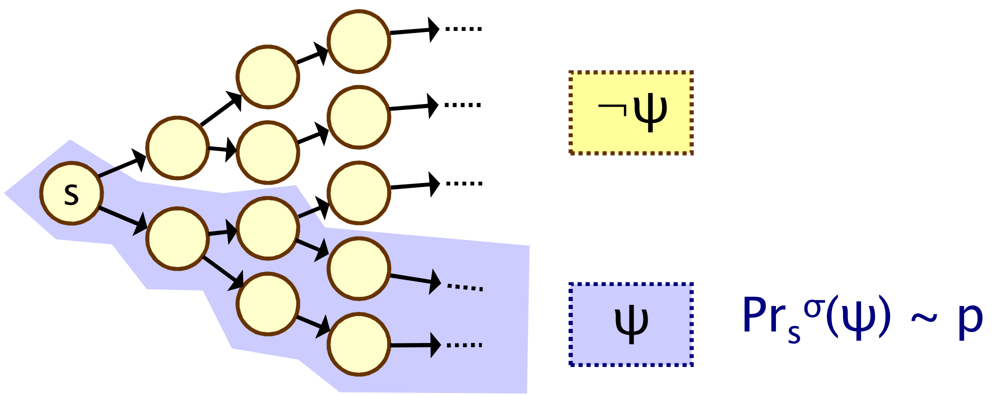
- Temporal logic for properties of MDPs (and DTMCs)
  - extension of (non-probabilistic) temporal logic CTL
  - key addition is probabilistic operator P
  - quantitative extension of CTL's A and E operators

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

- Example: send → $P_{\geq 0.95}$ [true $U^{\leq 10}$ deliver]

PCTL semantics for MDPs

- Semantics of the probabilistic operator P
  - can only define probabilities for a specific strategy σ
  - $s \vDash P_{\sim p} [\psi]$ means “the probability, from state s, that ψ is true for an outgoing path satisfies ~p for all strategies σ”
  - formally $s \vDash P_{\sim p} [\psi] \Leftrightarrow Pr_s^\sigma(\psi) \sim p$ for all strategies σ
  - where we use $Pr_s^\sigma(\psi)$ to denote $Pr_s^\sigma \{ \omega \in Path_s^\sigma \mid \omega \vDash \psi \}$



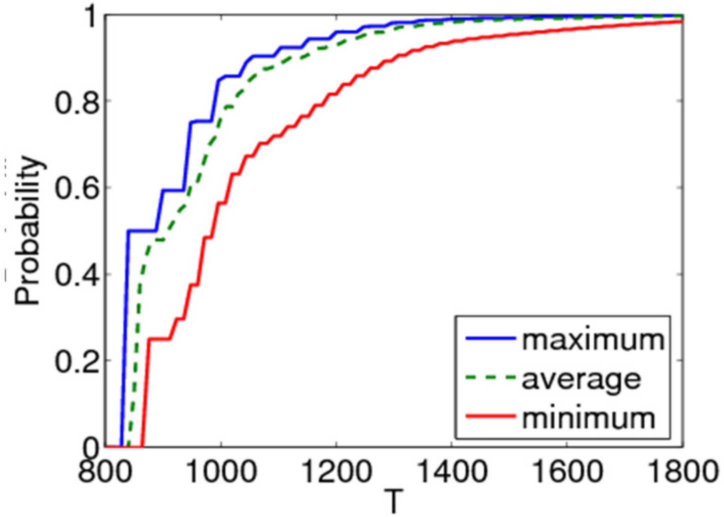
Minimum and maximum probabilities

- Letting:
  - $Pr_s^{max}(\psi) = \sup_\sigma Pr_s^\sigma(\psi)$
  - $Pr_s^{min}(\psi) = \inf_\sigma Pr_s^\sigma(\psi)$
- We have:
  - if $\sim \in \{\geq, >\}$, then $s \models P_{\sim p} [\ \psi\ ] \Leftrightarrow Pr_s^{min}(\psi) \sim p$
  - if $\sim \in \{<, \leq\}$, then $s \models P_{\sim p} [\ \psi\ ] \Leftrightarrow Pr_s^{max}(\psi) \sim p$
- Model checking $P_{\sim p}[\ \psi\ ]$ reduces to the computation over all strategies of either:
  - the minimum probability of ψ holding
  - the maximum probability of ψ holding
- Crucial result for model checking PCTL until on MDPs
  - memoryless strategies suffice, i.e. there are always memoryless strategies σ_{min} and σ_{max} for which:
  - $Pr_s^{\sigma_{min}}(\psi) = Pr_s^{min}(\psi)$ and $Pr_s^{\sigma_{max}}(\psi) = Pr_s^{min}(\psi)$

Quantitative properties

- For PCTL properties with P as the outermost operator
  - quantitative form (two types): $P_{min=?}$ [ψ] and $P_{max=?}$ [ψ]
  - i.e. "what is the minimum/maximum probability (over all adversaries) that path formula ψ is true?"
  - corresponds to an analysis of best-case or worst-case behaviour of the system
  - model checking is no harder since compute the values of $Pr_s^{min}(\psi)$ or $Pr_s^{max}(\psi)$ anyway
  - useful to spot patterns/trends

- Example: CSMA/CD protocol
  - "min/max probability that a message is sent within the deadline"



PCTL model checking for MDPs

- Algorithm for PCTL model checking [BdA95]
  - inputs: MDP $M=(S,s_{init},\alpha,\delta,L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \models \phi \}$ = set of states satisfying ϕ
- Basic algorithm same as PCTL model checking for DTMCs
  - proceeds by induction on parse tree of ϕ
  - non-probabilistic operators (true, a, $\neg$, $\wedge$) straightforward
- Only need to consider $P_{\sim p}$ [ψ] formulas
  - reduces to computation of $Pr_s^{min}(\psi)$ or $Pr_s^{max}(\psi)$ for all $s \in S$
  - dependent on whether $\sim \in \{\geq,>\}$ or $\sim \in \{<,\leq\}$
  - these slides cover the case $Pr_s^{min}(\phi_1 \ U\ \phi_2)$, i.e. $\sim \in \{\geq,>\}$
  - case for maximum probabilities is very similar

PCTL until for MDPs

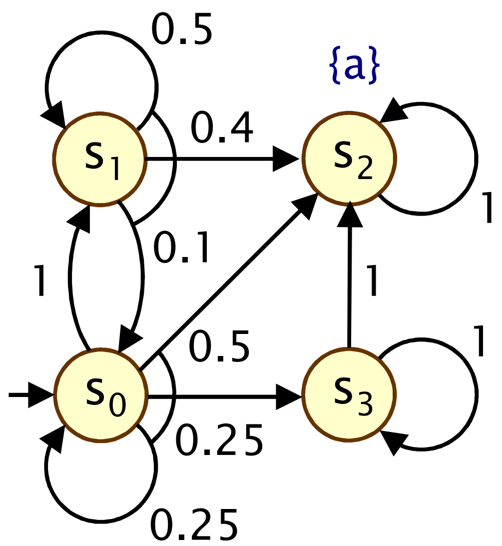
- Computation of probabilities $Pr_s^{min}(\phi_1 \text{ U } \phi_2)$ for all $s \in S$
- First identify all states where the probability is 1 or 0
  - "precomputation" algorithms, yielding sets S^{yes}, S^{no}
- Then compute (min) probabilities for remaining states ($S^?$)
  - either: solve linear programming problem
  - or: approximate with an iterative solution method
  - or: use policy iteration

Example:

$P_{\geq p}$ [F a]

$\equiv$

$P_{\geq p}$ [true U a]

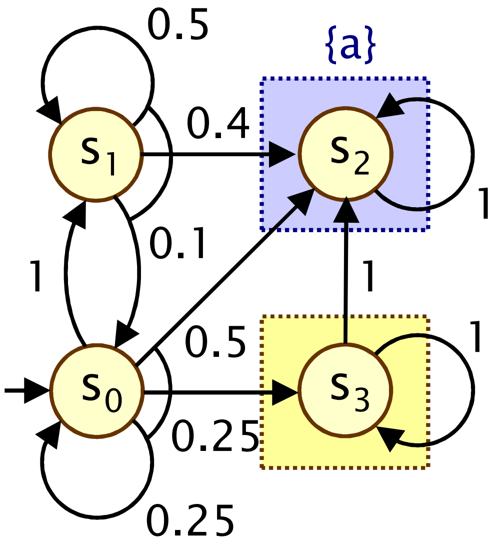


PCTL until – Precomputation

- Identify all states where $Pr_s^{min}(\phi_1 \text{ U } \phi_2)$ is 1 or 0
  - S^{yes} = Sat($P_{\geq 1}$ [ϕ_1 U ϕ_2]), S^{no} = Sat($\neg$ $P_{>0}$ [ϕ_1 U ϕ_2])
- Two graph-based precomputation algorithms:
  - algorithm Prob1A computes S^{yes}
    - for all strategies the probability of satisfying ϕ_1 U ϕ_2 is 1
  - algorithm Prob0E computes S^{no}
    - there exists a strategy for which the probability is 0

Example:

$P_{\geq p}$ [F a]



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

S^{no} = Sat($\neg P_{>0}$ [F a])

Method 1 – Linear programming

- Probabilities $Pr_s^{min}(\phi_1 \text{ U } \phi_2)$ for remaining states in the set $S^? = S \setminus (S^{yes} \cup S^{no})$ can be obtained as the unique solution of the following linear programming (LP) problem:

$$\text{maximize } \sum_{s \in S^?} x_s \text{ subject to the constraints :}$$

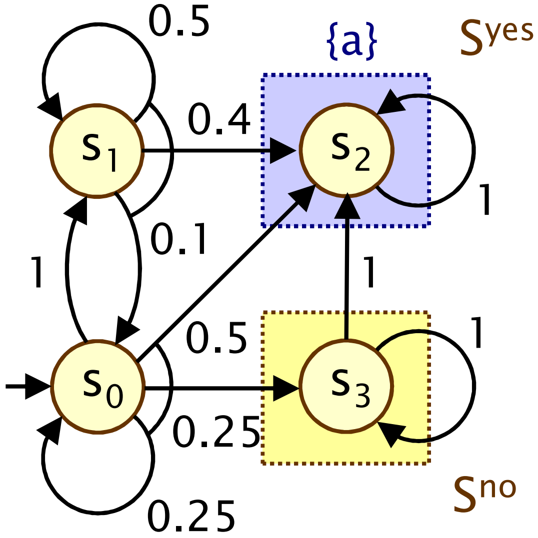
$$x_s \le \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s')$$

$$\text{for all } s \in S^? \text{ and for all } (a, \mu) \in \delta(s)$$

- Simple case of a more general problem known as the stochastic shortest path problem [BT91]

- This can be solved with standard techniques
  - e.g. Simplex, ellipsoid method, branch-and-cut

Example – PCTL until (LP)



Let $x_i = Pr_{s_i}^{min}(F\ a)$

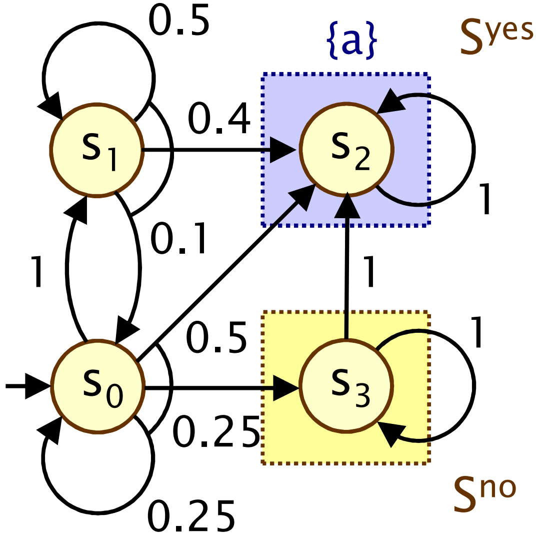
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 0.25 \cdot x_0 + 0.5$
- $x_1 \leq 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

Example – PCTL until (LP)



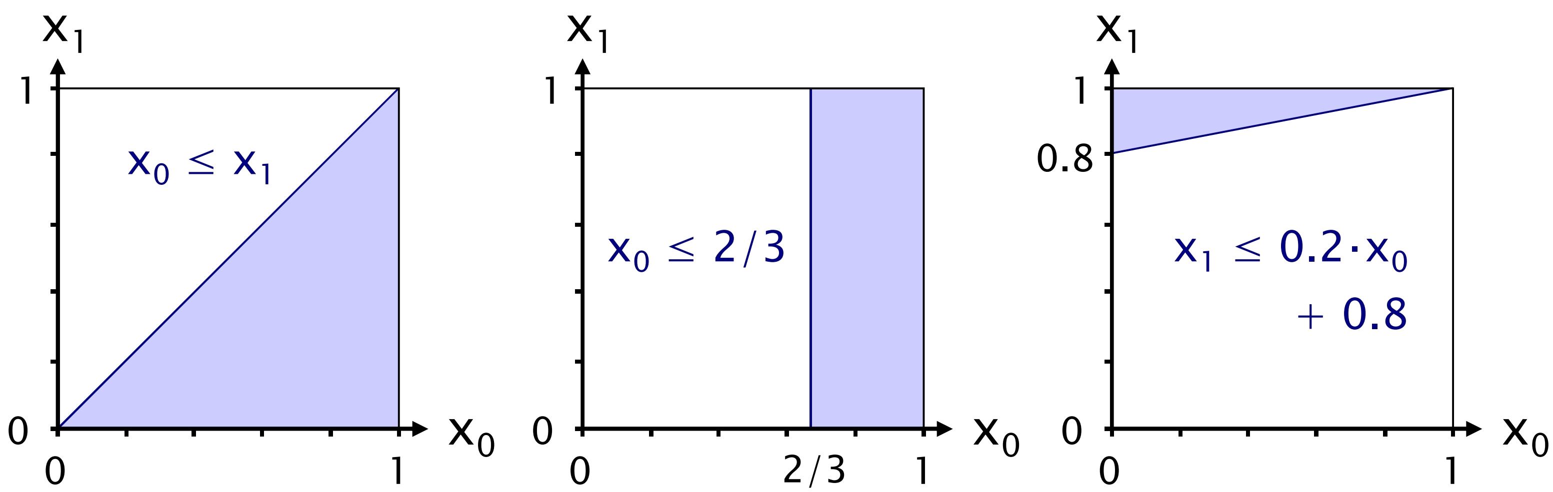
Let $x_i = Pr_{s_i}^{min}(F\ a)$

S^{yes}: $x_2=1$, S^{no}: $x_3=0$

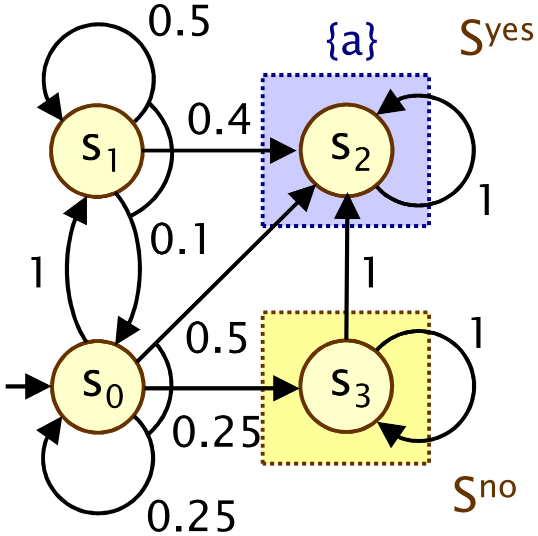
For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$



Example – PCTL until (LP)



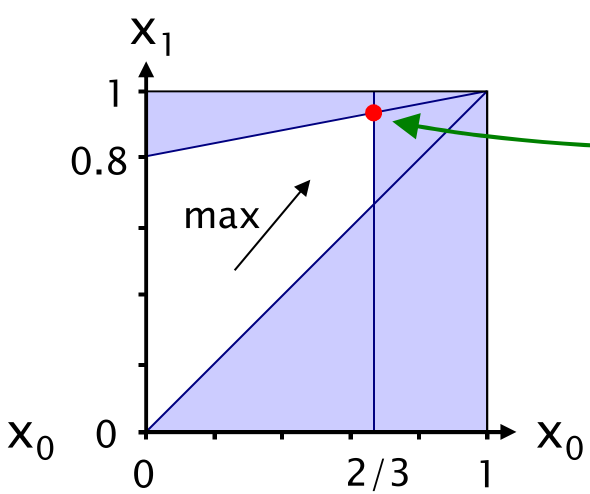
Let $x_i = Pr_{s_i}^{min}(F\ a)$

S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \le x_1$
- $x_0 \le 2/3$
- $x_1 \le 0.2 \cdot x_0 + 0.8$



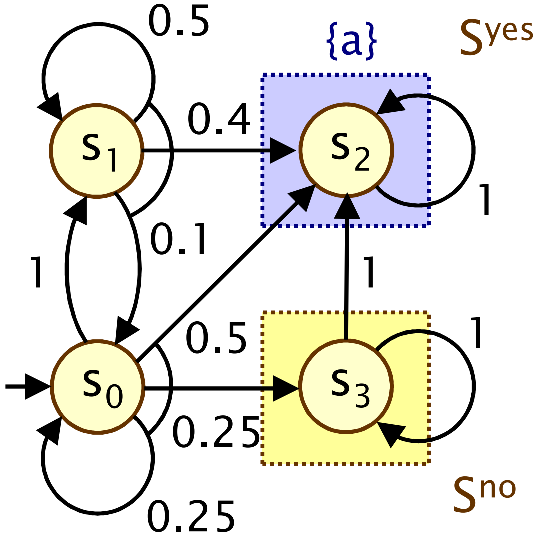
Solution:

(x_0, x_1)

$=$

$(2/3, 14/15)$

Example – PCTL until (LP)



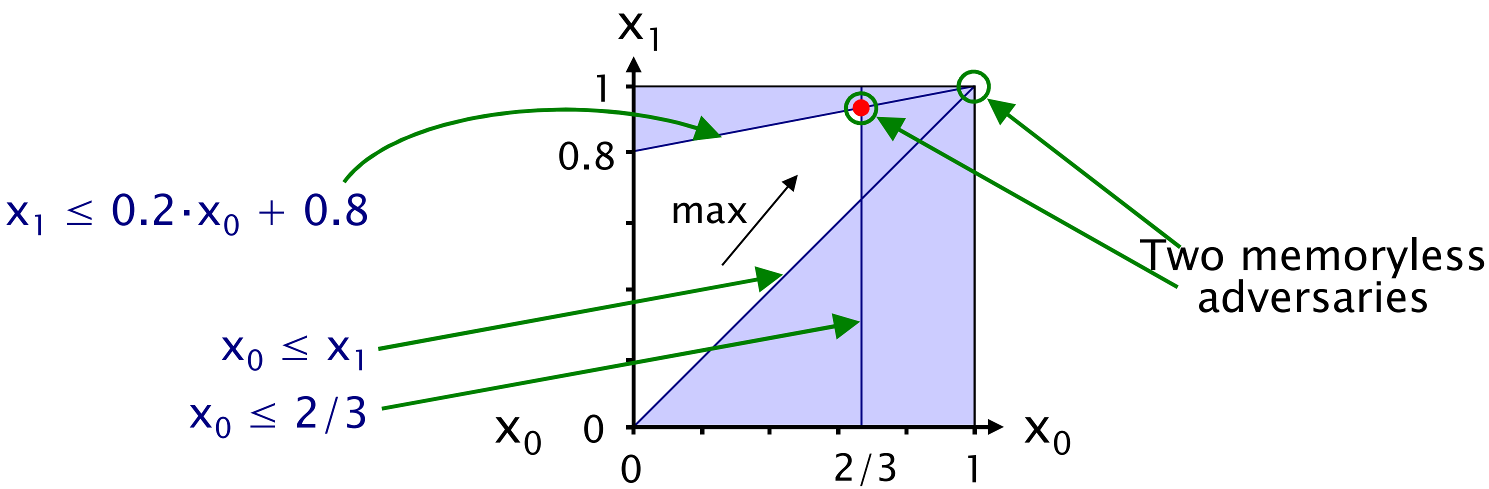
Let $x_i = Pr_{s_i}^{min}(F\ a)$

S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$



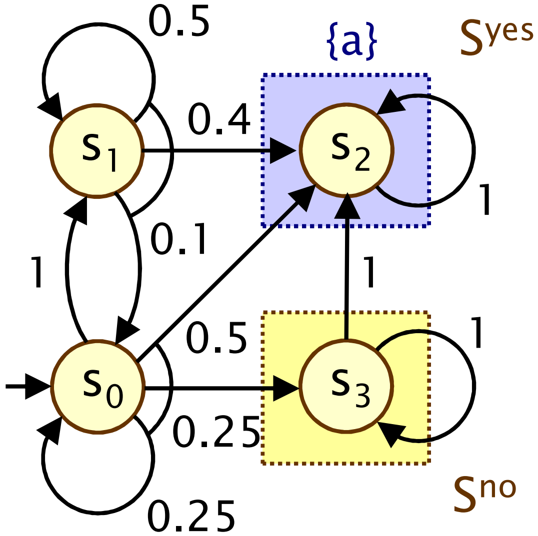
Method 2 – Value iteration

- For probabilities $Pr_s^{min}(\phi_1 \text{ U } \phi_2)$ it can be shown that:
  - $Pr_s^{min}(\phi_1 \text{ U } \phi_2) = \lim_{n\to\infty} x_s^{(n)}$ where:

$$x_s^{(n)} = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ 0 & \text{if } s \in S^{?} \text{ and } n = 0 \\ \min_{(a,\mu)\in Steps(s)} \left(\sum_{s' \in S} \mu(s') \cdot x_{s'}^{(n-1)} \right) & \text{if } s \in S^{?} \text{ and } n > 0 \end{cases}$$

- This forms the basis for an (approximate) iterative solution
  - iterations terminated when solution converges sufficiently

Example – PCTL until (value iteration)



Compute: $Pr_{s_i}^{min}(F\ a)$

$S^{yes} = \{x_2\}$, $S^{no} = \{x_3\}$, $S^{?} = \{x_0, x_1\}$

$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$

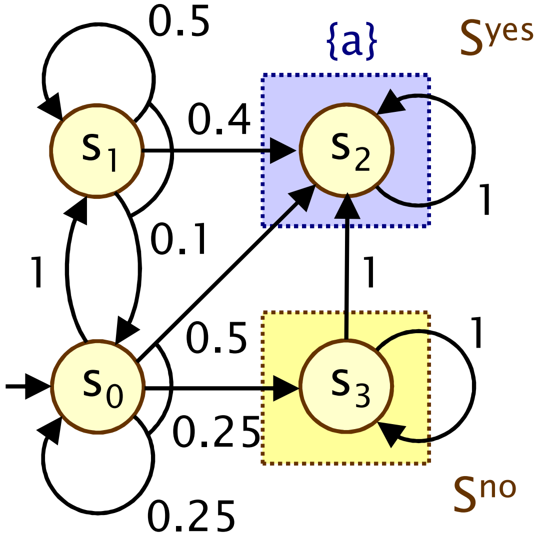
n=0: $[\ 0, 0, 1, 0\ ]$

n=1: $[\ \min(0, 0.25\cdot0+0.5),$
$0.1\cdot0+0.5\cdot0+0.4, 1, 0\ ]$
$= [\ 0, 0.4, 1, 0\ ]$

n=2: $[\ \min(0.4, 0.25\cdot0+0.5),$
$0.1\cdot0+0.5\cdot0.4+0.4, 1, 0\ ]$
$= [\ 0.4, 0.6, 1, 0\ ]$

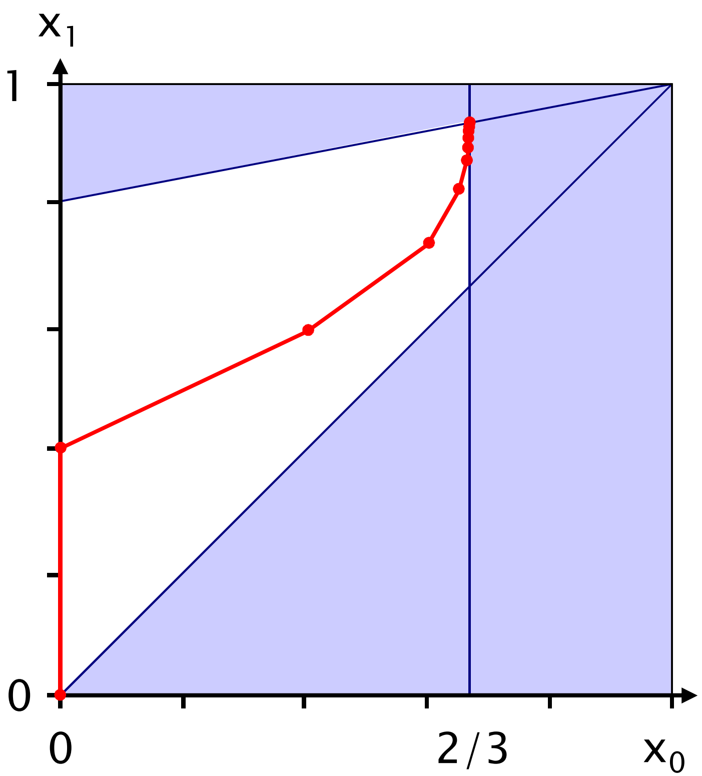
n=3: …

Example – PCTL until (value iteration)



	$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$
n=0:	[0.000000, 0.000000, 1, 0]
n=1:	[0.000000, 0.400000, 1, 0]
n=2:	[0.400000, 0.600000, 1, 0]
n=3:	[0.600000, 0.740000, 1, 0]
n=4:	[0.650000, 0.830000, 1, 0]
n=5:	[0.662500, 0.880000, 1, 0]
n=6:	[0.665625, 0.906250, 1, 0]
n=7:	[0.666406, 0.919688, 1, 0]
n=8:	[0.666602, 0.926484, 1, 0]
n=9:	[0.666650, 0.929902, 1, 0]
	…
n=20:	[0.666667, 0.933332, 1, 0]
n=21:	[0.666667, 0.933332, 1, 0]
	≈ [2/3, 14/15, 1, 0]

Example – Value iteration + LP



	$[\, x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)} \,]$
n=0:	[0.000000, 0.000000, 1, 0]
n=1:	[0.000000, 0.400000, 1, 0]
n=2:	[0.400000, 0.600000, 1, 0]
n=3:	[0.600000, 0.740000, 1, 0]
n=4:	[0.650000, 0.830000, 1, 0]
n=5:	[0.662500, 0.880000, 1, 0]
n=6:	[0.665625, 0.906250, 1, 0]
n=7:	[0.666406, 0.919688, 1, 0]
n=8:	[0.666602, 0.926484, 1, 0]
n=9:	[0.666650, 0.929902, 1, 0]
	…
n=20:	[0.666667, 0.933332, 1, 0]
n=21:	[0.666667, 0.933332, 1, 0]
	≈ [2/3, 14/15, 1, 0]

Method 3 – Policy iteration

- Value iteration:
  - iterates over (vectors of) probabilities
- Policy iteration:
  - iterates over strategies ("policies")

- 1. Start with an arbitrary (memoryless) strategy σ
- 2. Compute the reachability probabilities $\underline{Pr}^{\sigma}$(F a) for σ
- 3. Improve the strategy in each state
- 4. Repeat 2/3 until no change in strategy

- Termination:
  - finite number of memoryless strategies
  - improvement in (minimum) probabilities each time

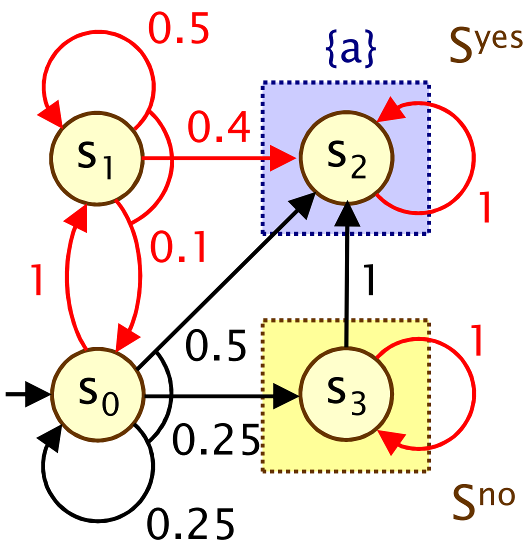
Method 3 – Policy iteration

- 1. Start with an arbitrary (memoryless) strategy σ
  - pick an element of $\delta(s)$ for each state $s \in S$
- 2. Compute the reachability probabilities $\underline{Pr}^{\sigma}(F\ a)$ for σ
  - probabilistic reachability on a DTMC
  - i.e. solve linear equation system
- 3. Improve the strategy in each state

$$\sigma'(s) = \text{arg}\min \left\{ \sum_{s' \in S} \mu(s') \cdot Pr_{s'}^{\sigma}(F\ a) \mid (a, \mu) \in \delta(s) \right\}$$

- 4. Repeat 2/3 until no change in strategy

Example – Policy iteration



Arbitrary strategy σ:

Compute: $\underline{Pr}^{\sigma}(F\ a)$

Let $x_i = Pr_{s_i}{}^{\sigma}(F\ a)$

$x_2=1$, $x_3=0$ and:

- $x_0 = x_1$
- $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

Solution:

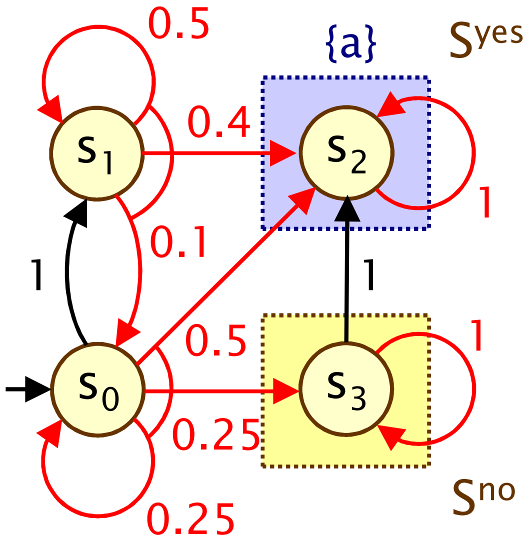
$\underline{Pr}^{\sigma}(F\ a) = [\ 1, 1, 1, 0\ ]$

Refine σ in state s_0:

min{1(1), 0.5(1)+0.25(0)+0.25(1)}

= min{1, 0.75} = 0.75

Example – Policy iteration



Refined strategy σ':

Compute: $\underline{Pr}^{\sigma'}(F\ a)$

Let $x_i = Pr_{s_i}^{\sigma'}(F\ a)$

$x_2=1$, $x_3=0$ and:

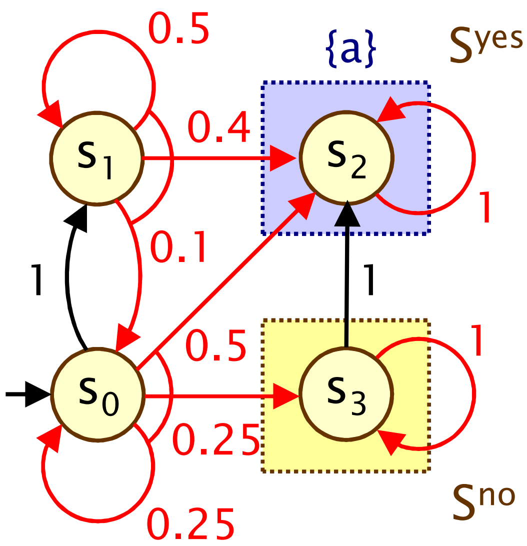
- $x_0 = 0.25 \cdot x_0 + 0.5$
- $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

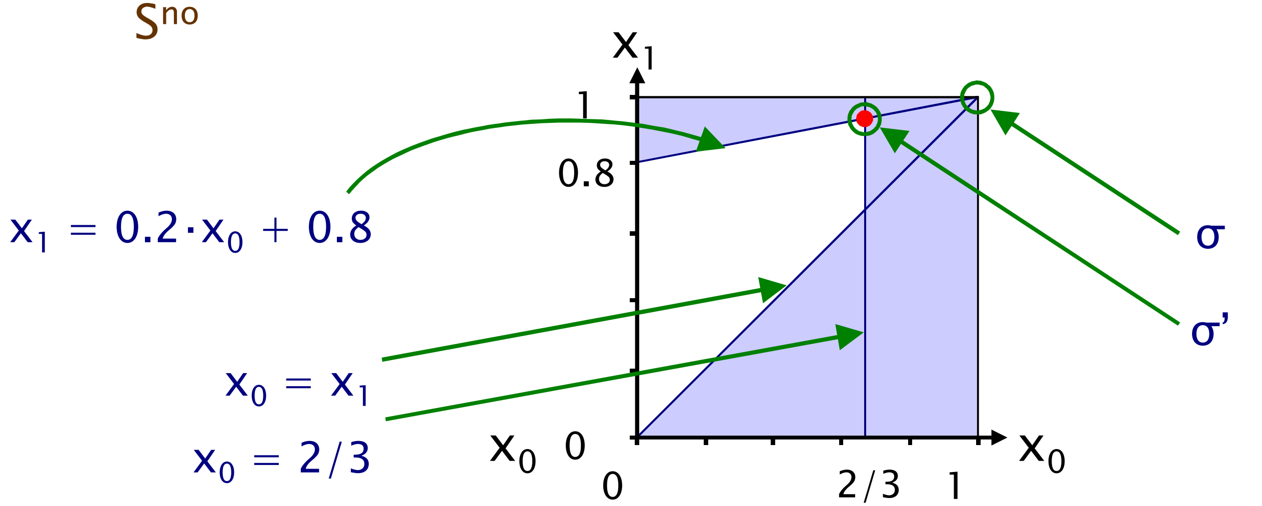
Solution:

$\underline{Pr}^{\sigma'}(F\ a) = [\ 2/3,\ 14/15,\ 1,\ 0\ ]$

This is optimal

Example – Policy iteration





PCTL model checking - Summary

- Computation of set Sat(Φ) for MDP M 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 programming problem, polynomial in |S| (assuming use of linear programming)
- Complexity:
  - linear in |Φ| and polynomial in |S|
  - S is states in MDP, assume |δ(s)| is constant

Costs and rewards for MDPs

- We can augment MDPs 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

- Extend logic PCTL with R operator, for “expected reward”
  - as for PCTL, either $R_{\sim r}$ […], $R_{min=?}$ […] or $R_{max=?}$ […]
- Some examples:
  - $R_{min=?}$ [$I^{=90}$], $R_{max=?}$ [$C^{\leq 60}$], $R_{max=?}$ [F “end”]
  - “the minimum expected queue size after exactly 90 seconds”
  - “the maximum expected power consumption over one hour”
  - the maximum expected time for the algorithm to terminate

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] – the 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, temporal operators always appear inside $P_{\sim p}$ […]
  - (and, in CTL, they always appear inside A or E)
  - in LTL (and PCTL*), temporal operators can be combined

LTL + probabilities

- Same idea as PCTL: probabilities of sets of path formulae
  - for a state s of a DTMC and an LTL formula ψ:
  - $Prob(s, \psi) = Pr_s \{ \omega \in Path(s) \mid \omega \models \psi \}$
  - all such path sets are measurable (see later)
- For MDPs, we can again consider lower/upper bounds
  - $p_{min}(s, \psi) = \inf_{\sigma \in Adv} Prob^{\sigma}(s, \psi)$
  - $p_{max}(s, \psi) = \sup_{\sigma \in Adv} Prob^{\sigma}(s, \psi)$
  - (for LTL formula ψ)
- For DTMCs or MDPs, an LTL specification often comprises an LTL (path) formula and a probability bound
  - e.g. $P_{>0.99}$ [F (req ∧ X ack)]

 79

LTL model checking for DTMCs

- Model check LTL specification $P_{\sim p}$ [ψ] against DTMC D

- 1. Generate a deterministic Rabin automaton (DRA) for ψ
  - build nondeterministic Büchi automaton (NBA) for ψ [VW94]
  - convert the NBA to a DRA [Saf88]
- 2. Construct product DTMC D⊗A
- 3. Identify accepting BSCCs of D⊗A
- 4. Compute probability of reaching accepting BSCCs
  - from all states of the D⊗A
- 5. Compare probability for (s, q_s) against p for each s

- Qualitative LTL model checking – no probabilities needed

PCTL* model checking

- PCTL* syntax:
  - ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $P_{\sim p}$ [ψ]
  - ψ ::= ϕ | $\psi \wedge \psi$ | $\neg\psi$ | X ψ | ψ U ψ
- Example:
  - $P_{>p}$ [GF (send → $P_{>0}$ [F ack])]

- PCTL* model checking algorithm
  - bottom-up traversal of parse tree for formula (like PCTL)
  - to model check $P_{\sim p}$ [ψ]:
    - replace maximal state subformulae with atomic propositions
    - (state subformulae already model checked recursively)
    - modified formula ψ is now an LTL formula
    - which can be model checked as for LTL

 81

LTL model checking for MDPs

- Model check LTL specification $P_{\sim p}$ [ψ] against MDP M
- 1. Convert problem to one needing maximum probabilities
  - e.g. convert $P_{>p}$ [ψ] to $P_{<1-p}$ [¬ψ]
- 2. Generate a DRA for ψ (or ¬ψ)
  - build nondeterministic Büchi automaton (NBA) for ψ [VW94]
  - convert the NBA to a DRA [Saf88]
- 3. Construct product MDP M⊗A
- 4. Identify accepting end components (ECs) of M⊗A
- 5. Compute max. probability of reaching accepting ECs
  - from all states of the D⊗A
- 6. Compare probability for (s, q_s) against p for each s

 82

Complexity

- Complexity of model checking LTL formula ψ on DTMC D
  - is doubly exponential in |ψ| and polynomial in |D|
- Converting LTL formula ψ to DRA A
  - for some LTL formulae of size n, size of smallest DRA is 2^{2^n}
- In total: O(poly(|D|,|A|))
- In practice: |ψ| is small and |D| is large
- Can be reduced to single exponential in |ψ|
  - see e.g. [CY88,CY95]

- Complexity of model checking LTL formula ψ on MDP M
  - is doubly exponential in |ψ| and polynomial in |M|
  - unlike DTMCs, this cannot be improved upon

 83

Part 3

Probabilistic programs as MDPs

Probabilistic software

- Consider sequential ANSI C programs
  - support functions, pointers, arrays, but not dynamic memory allocation, unbounded recursion, floating point operations
- Add function bool coin(double p) for probabilistic choice
  - for modelling e.g. failures, randomisation
- Add function int ndet(int n) for nondeterministic choice
  - for modelling e.g. user input, unspecified function calls
- Aim: verify software with failures, e.g. wireless protocols
  - extract models as Markov decision processes
  - properties: maximum probability of unsuccessful data transmission, minimum expected number of packets sent
- Develop abstraction-refinement framework [VMCAI09]

Example – sample target program

```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = num_to_send ();
    while (! fail && c > 0)
    {
        // probabilistic
        fail = send_msg ();
        c --;
    }
}
```

Φ: “what is the minimum/maximum probability of the program terminating with fail being true?”

Example – simplified

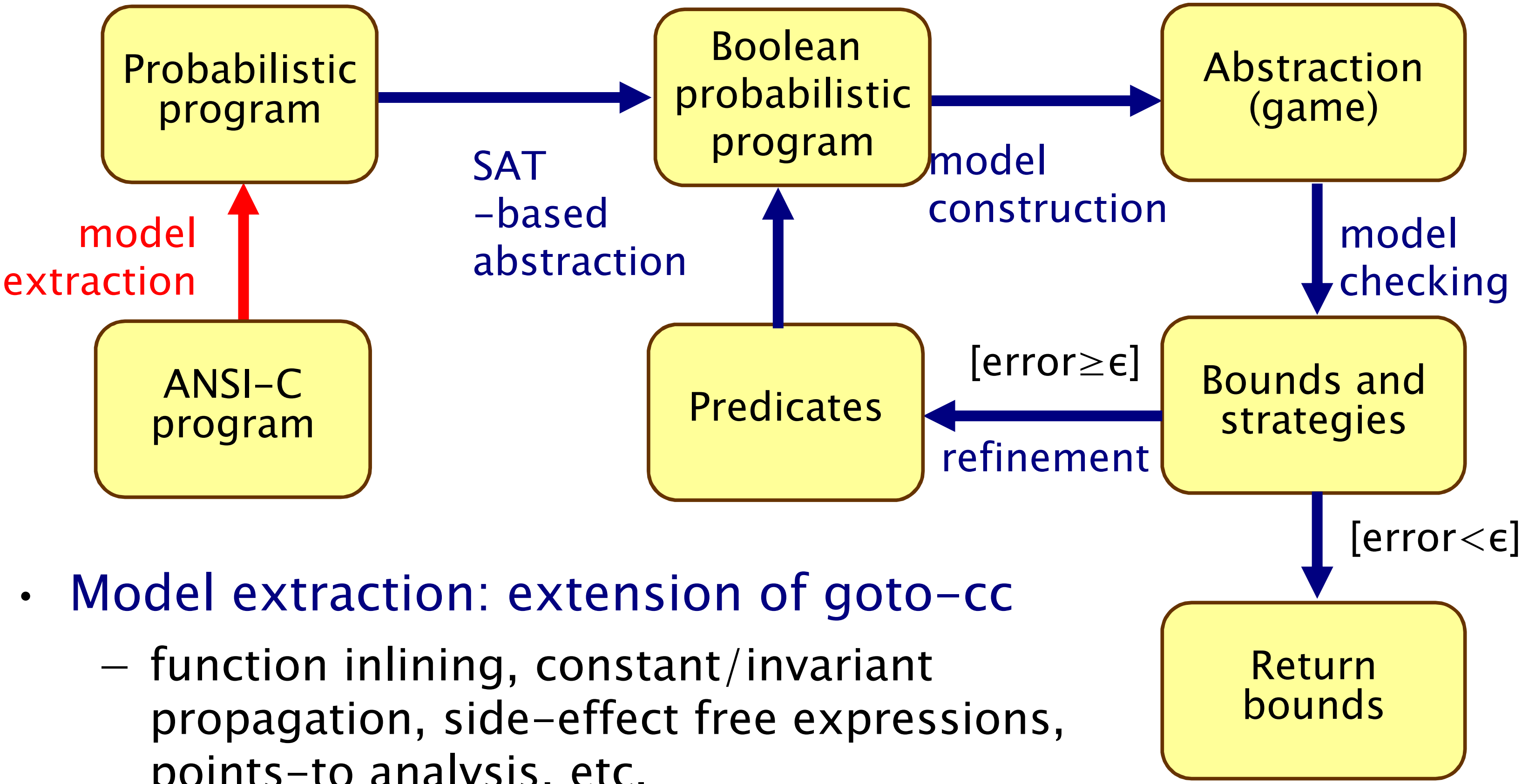
```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
```

input nondeterminism

Φ: "what is the minimum/maximum probability of the program terminating with fail being true?"

Bernoulli distribution

Abstraction-refinement loop

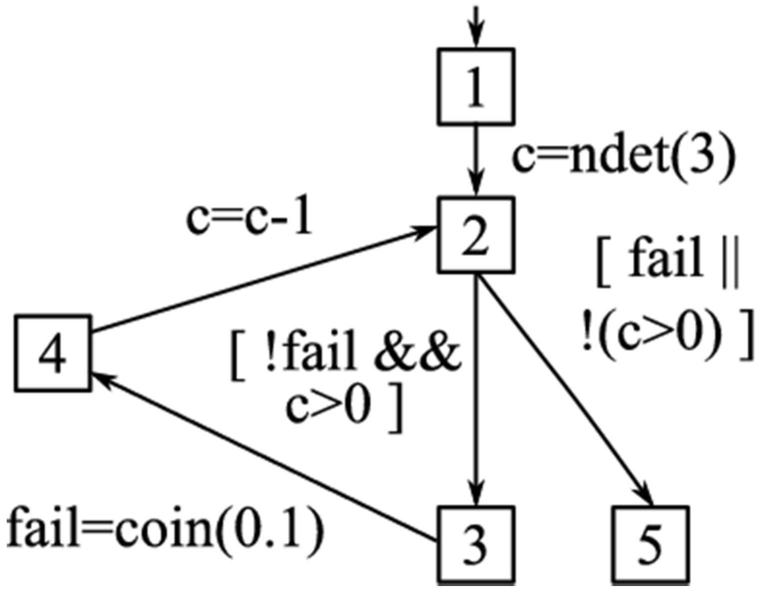


- Model extraction: extension of goto-cc
  - function inlining, constant/invariant propagation, side-effect free expressions, points-to analysis, etc.
- Probabilistic program
  - probabilistic control flow graph
  - Markov decision process (MDP) semantics

Back to example

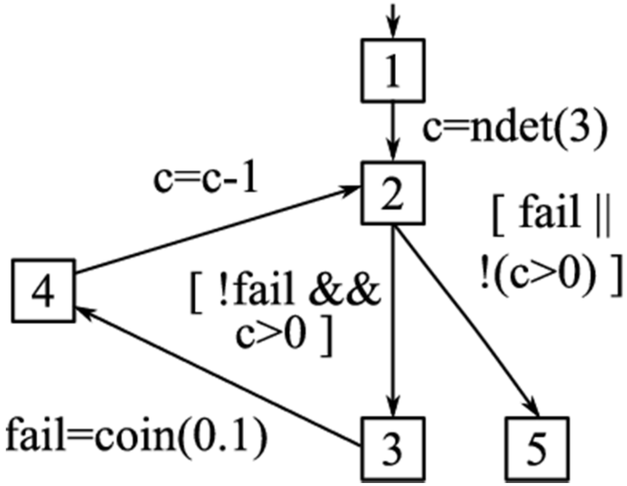
```
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = ndet (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = coin (0.1);
        c --;
    }
}
```

Probabilistic program

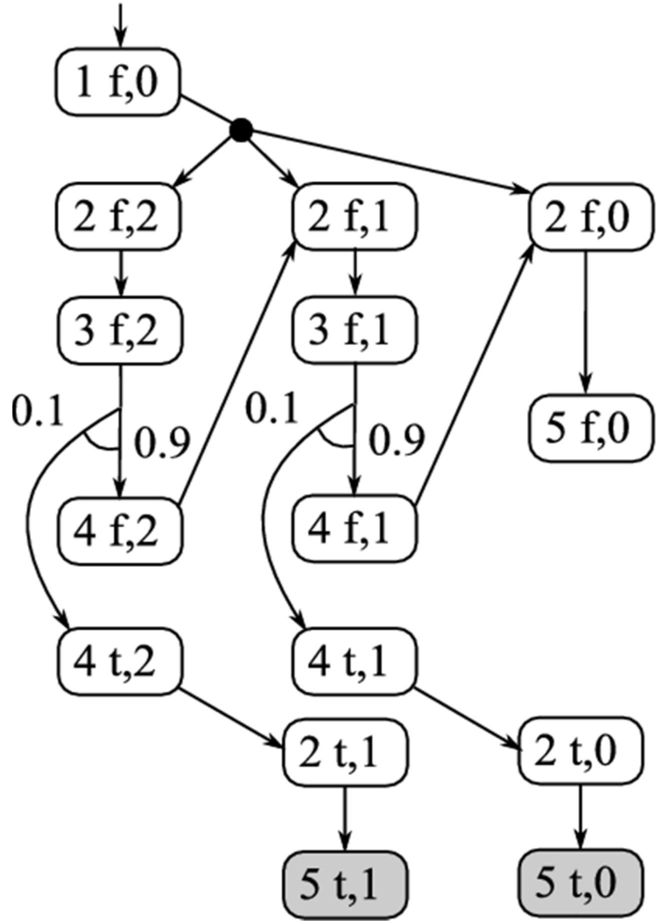


Probabilistic program as MDP

Probabilistic program



MDP semantics



minimum/maximum probability of the program terminating with fail being true is 0 and 0.19, respectively

Experimental results

- Successfully applied to several Linux network utilities:
  - TFTP (file-transfer protocol client)
  - 1 KLOC of non-trivial ANSI-C code
  - Loss of packets modelled by probabilistic choice
  - Linux kernel calls modelled by nondeterministic choice

- Example properties
  - "maximum probability of establishing a write request"
  - "maximum expected amount of data that is sent before timeout"
  - "maximum expected number of echo requests required to establish connectivity"

- Implemented through extension of CProver and PRISM

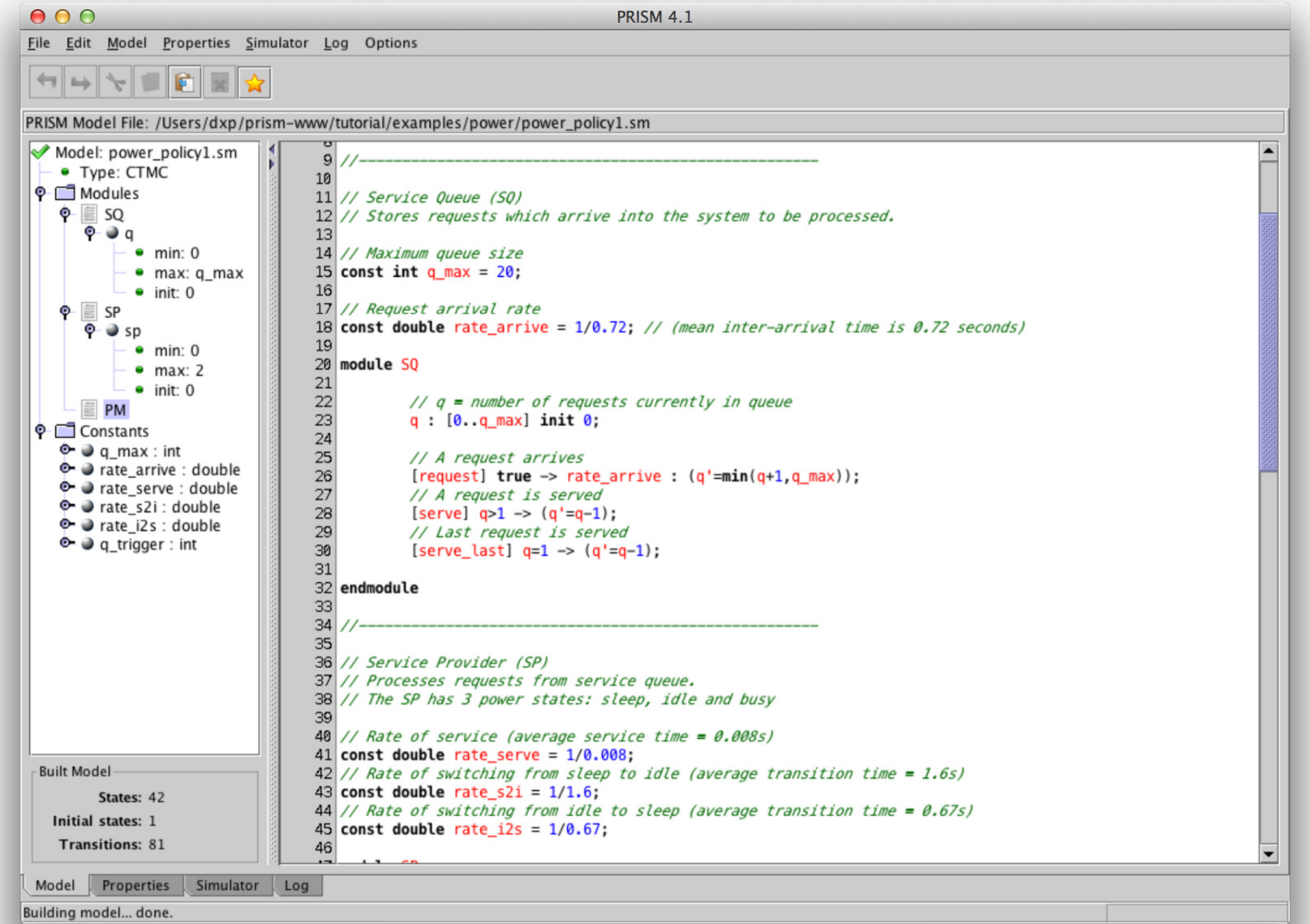
Part 4

PRISM

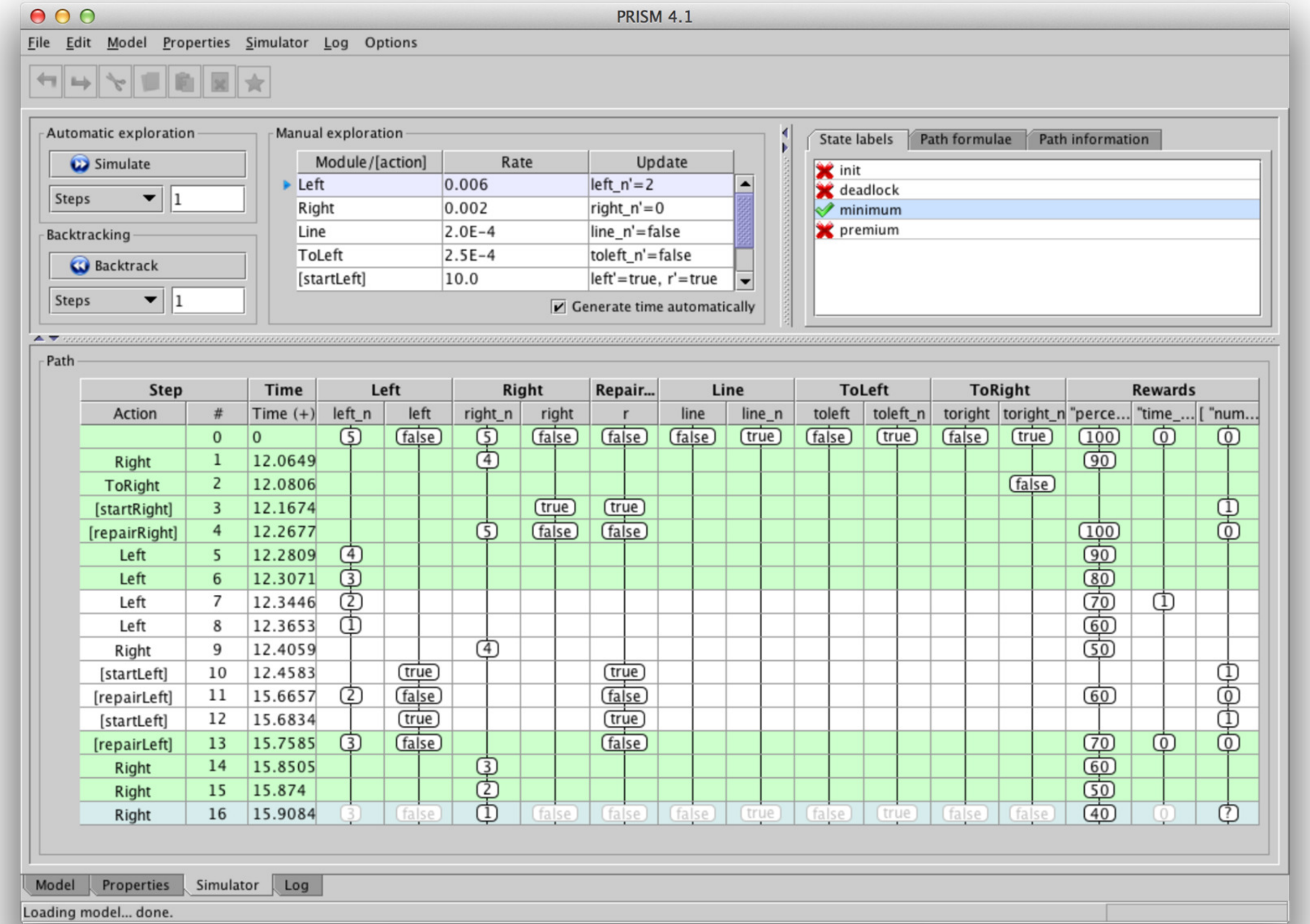
Tool support: PRISM

- PRISM: Probabilistic symbolic model checker [CAV11]
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
- Support for:
  - models: DTMCs, CTMCs, MDPs, PTAs, SMGs, …
  - properties: PCTL, CSL, LTL, PCTL*, costs/rewards, rPATL, …
- Features:
  - simple but flexible high-level modelling language
  - user interface: editors, simulator, experiments, graph plotting
  - multiple efficient model checking engines (e.g. symbolic)
  - New! strategy synthesis, stochastic game models (SMGs) , multiobjective verification, parametric models
- See: http://www.prismmodelchecker.org/

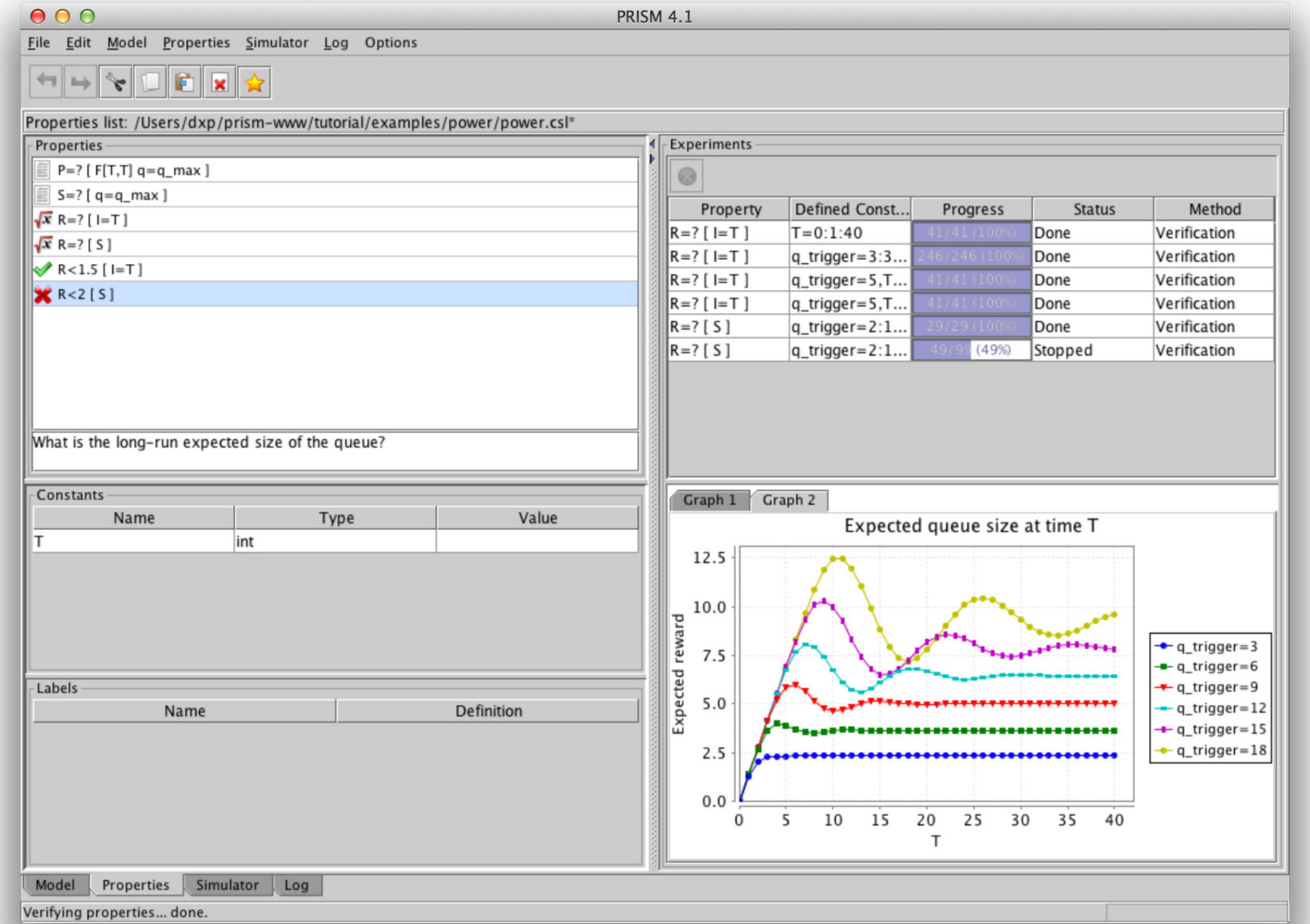
PRISM GUI: Editing a model



PRISM GUI: The Simulator

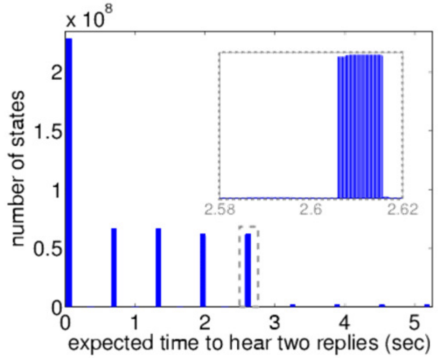


PRISM GUI: Model checking and graphs

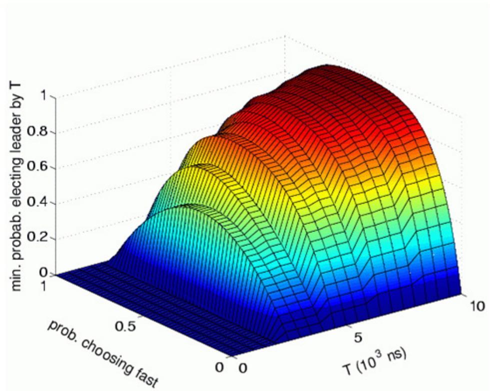


Probabilistic verification in action

- Bluetooth device discovery protocol
  - frequency hopping, randomised delays
  - low-level model in PRISM, based on detailed Bluetooth reference documentation
  - numerical solution of 32 Markov chains, each approximately 3 billion states
  - identified worst-case time to hear one message, 2.5 seconds

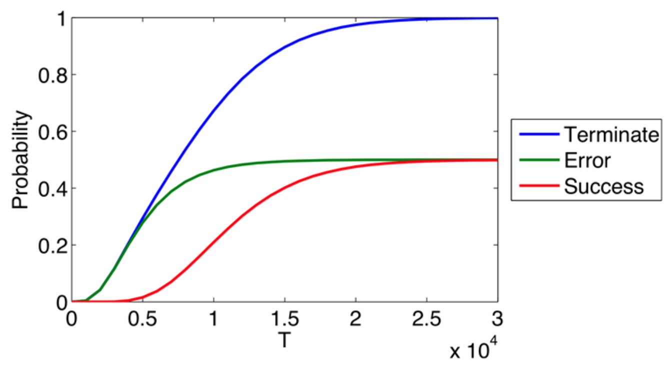


- FireWire root contention
  - wired protocol, uses randomisation
  - model checking using PRISM
  - optimum probability of leader election by time T for various coin biases
  - demonstrated that a biased coin can improve performance

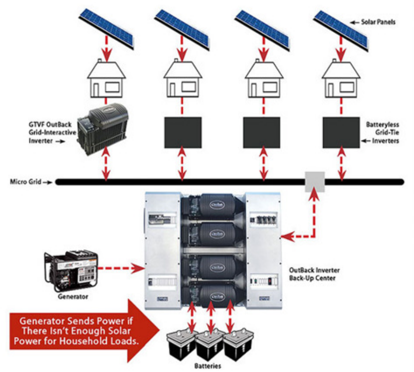


Probabilistic verification in action

- DNA transducer gate [Lakin et al, 2012]
  - DNA computing with a restricted class of DNA strand displacement structures
  - transducer design due to Cardelli
  - automatically found and fixed design error, using Microsoft's DSD and PRISM



- Microgrid demand management protocol [TACAS12,FMSD13]
  - designed for households to actively manage demand while accessing a variety of energy sources
  - found and fixed a flaw in the protocol, due to lack of punishment for selfish behaviour
  - implemented in PRISM-games



Summary

- Overview of probabilistic model checking
  - discrete-time Markov chains and Markov decision processes
  - property specifications in temporal logics
  - model checking methods combine graph-theoretic techniques, automata-based methods, numerical equation solving and optimisation
- Ongoing work (not discussed)
  - further models (stochastic games, probabilistic timed/hybrid automata)
  - controller/strategy synthesis
  - runtime verification
  - multiobjective verification and synthesis
  - sampling-based exploration
- Potential for connections to probabilistic programming
  - integrate with probabilistic inference

Further material

- 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