Probabilistic Model Checking

Marta Kwiatkowska
Dave Parker

Oxford University Computing Laboratory

ESSLLI'10 Summer School, Copenhagen, August 2010

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

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…

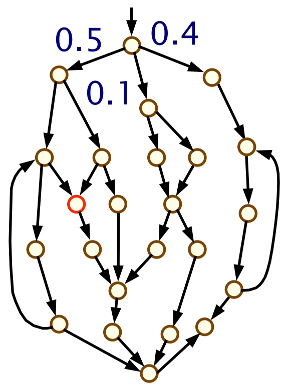
- Quantitative, as well as qualitative requirements:
  - how reliable is my car’s Bluetooth network?
  - how efficient is my phone’s power management policy?
  - is my bank’s web-service secure?
  - what is the expected long-run percentage of protein X?

Probabilistic model checking

System

Probabilistic model

e.g. Markov chain



System require–ments

$P_{<0.1}$ [F fail]

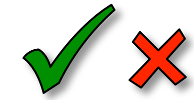
Probabilistic temporal logic specification

e.g. PCTL, CSL, LTL

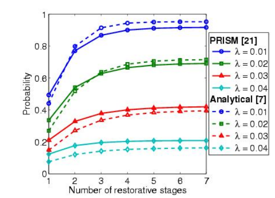
Probabilistic model checker

e.g. PRISM

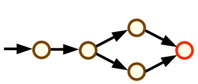
Result



Quantitative results



Counter–example



Probabilistic models

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) (probabilistic automata)
Continuous time	Continuous-time Markov chains (CTMCs)	CTMDPs / IMCs Probabilistic timed automata (PTAs)

we will focus on the red-parts

Part 1

Discrete-time Markov chains

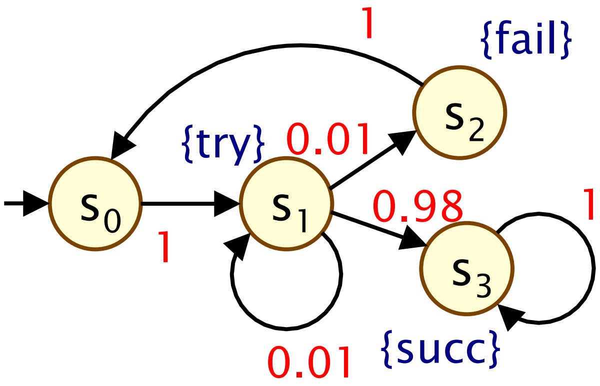
Overview (Part 1)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- LTL model checking
- Costs and rewards

Discrete-time Markov chains

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

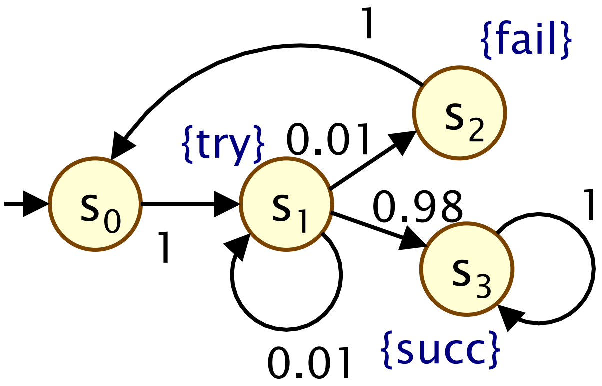
- States
  - discrete set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability of making transitions between states is given by discrete probability distributions



Discrete-time Markov chains

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

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



DTMCs: An alternative definition

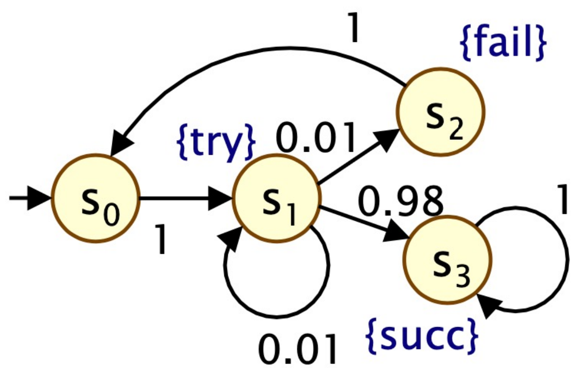
- Alternative definition: a DTMC is:
  - a family of random variables { X(k) | k=0,1,2,… }
  - X(k) are observations at discrete time-steps
  - i.e. X(k) is the state of the system at time-step k

- Memorylessness (Markov property)
  - $Pr(X(k)=s_k \mid X(k-1)=s_{k-1}, \ldots , X(0)=s_0)$
    $= Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$

- We consider homogenous DTMCs
  - transition probabilities are independent of time
  - $P(s_{k-1},s_k) = Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$

Probability of taking a path or a set of paths

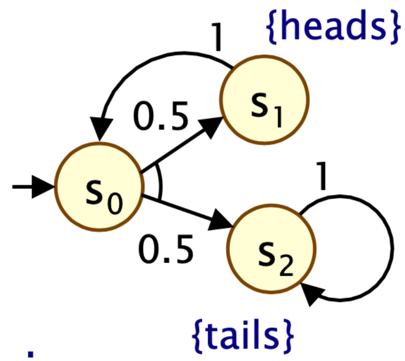
A "DTMC" :



- Consider a path ω e.g. s_0,s_1,s_2,s_0. The probability that the system follows this path when executed with the starting state s_0 is denoted by $P_{s0}(\omega)$. Or simply $P(\omega)$ if it is clear which s_0 is meant. It is the product of the probability of each transition in ω.

  Example: for the above ω, $P(\omega) = 1 * 0.01 * 1 = 0.01$
- For a **set of of paths** U (starting from s0), the probability that the system's execution follows **one of** the paths in U, denoted by P(U), is $\sum_{\omega \in U} P(\omega)$.

Probability of taking a path or a set of paths

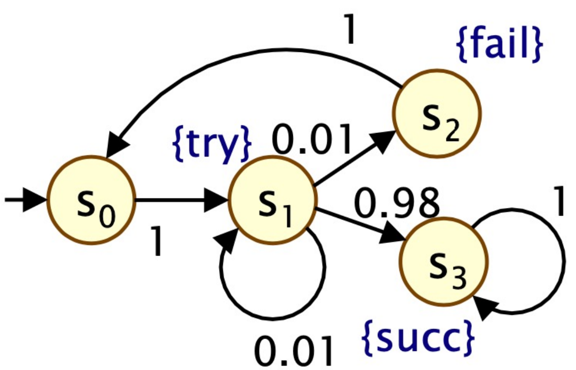


Example: consider U = the set of paths that ends in s_2. Note that U is infinite: U = { 02, 0102, 010102, … }. But we can calculate P(U).

$$P(U) = 0.5 + 0.52 + 0.53 + \ldots = \sum_{k\geq 0} 0.5^k$$

$$= 0.5 * \frac{1}{1 - 0.5} = 1$$

Probability Matrix Representation



P:

	s0	s1	s2	s3
s0	0	1	0	0
s1	0	0.01	0.01	0.98
s2	1	0	0	0
s3	0	0	0	1

$P_{i,k}$ = the value at the i-th row and k-th column. It specifies the probability of taking the transition $s_i \rightarrow s_k$, if we are now at s_i.

For example the circle red value above is $P_{1,2}$, specifying the probability of taking the transition from s_1 to s_2 (check the picture), which is 0.01.

Basic Operations on Probability Matrix

- Multiplying P with itself: P^n
- Multiplying a vector with P: $u \times P$
- Multiplying P with a vector: $P \times v$

P^n

- $P^0 = I$ (identity matrix)
  $P^{n+1} = P \times P^n$
- $P^n_{i,k}$ is the probability of ending up in state s_k in n-steps, given we start in the state s_i.
- For example, wirth the previous P, let's look at P^2:

0	1	0	0
0	0.01	0.01	0.98
1	0	0	0
0	0	0	1

×

0	1	0	0
0	0.01	0.01	0.98
1	0	0	0
0	0	0	1

→

?	?	0.01	?
?	?	?	?
?	?	?	?
?	?	?	?

$$P^2_{0,2} = (P \times P)_{0,2}$$
$$= P_{0,0} * P_{0,2} + P_{0,1} * P_{1,2} + P_{0,2} * P_{2,2} + P_{0,3} * P_{3,2}$$

Probabity distribution of the next state, given the current distribution

- The probability of currently being in various states ("*probability distribution*" of the current state) can be given by a vector of size K, if K is the number of possible states. E.g. if $u = [\,0\,,\,0.5\,,\,0.5,\,0\,]$ is the probability distribution of the current state, it says e.g. that there is 0.5 probability that currently we are in the state s_1, but 0 probability that we are in the state s_0.
- The product u × P (we often simply write it as uP) gives a new vector u' of size K, that gives us the probability distribution of the next state.

$$\overset{u}{\begin{bmatrix} 0 & 0.5 & 0.5 & 0 \end{bmatrix}} \times \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \overset{u'}{\begin{bmatrix} 0.5 & 0.005 & 0.005 & 0.49 \end{bmatrix}}$$

e.g. $u'_1 = u \bullet$ the green collumn (dot product)

$$= u_0 * P_{0,1} + u_1 * P_{1,1} + u_2 * P_{2,1} + u_3 * P_{3,1}$$

Probabilty vector

- Sometimes we also want to know what the probability to end up in state, say, s_1 or s_2 as the **next** state, if we start in the state s1.
- We can represent "end up in either s_1 or s_2" with a vector $v = [0,1,1,0]$.
- Let v^t is the *transpose* of v. The product $P \times v^t$ gives a w such that w is a (transposed) vector, where w_i is the probabilty to end up in one of the states specified in v, if we start in s_i.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \underbrace{\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}}_{v^t} \Rightarrow \underbrace{\begin{bmatrix} 1 \\ 0.02 \\ 0 \\ 0 \end{bmatrix}}_{w}$$

e.g. w_1 = the green row • v^t (dot product)

$= P_{1,0} * v_0 + P_{1,1} * v_1 + P_{1,2} * v_2 + P_{1,3} * v_3$

Probability vector

- Prob(φ) (notice the underscore) is a probility vector e.g. w =

1
0.02
0
0

  such that the i-th element tells us what the probability that the system would behave as φ if executed in state s_i.

- Example: the above w (blue) happens to be equal to Prob(**X**(try $\vee$ fail)).

- This notation Prob will be used later when we discuss model checking of probabilistic-CTL.

Overview (Part 1)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- LTL model checking
- Costs and rewards

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:

  ψ is true with probability ~p

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

    "next" "bounded until" "until"

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

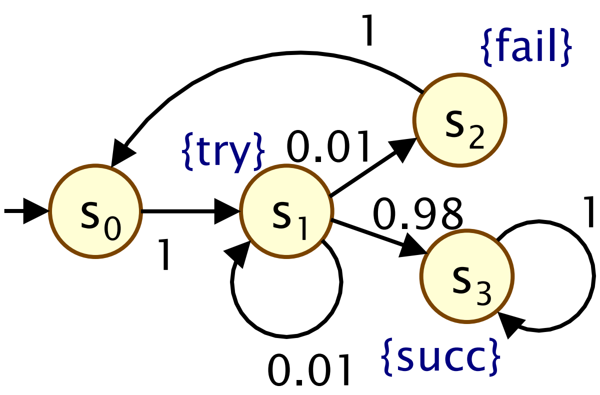
- A PCTL formula is always a state formula
  - path formulas only occur inside the P operator

PCTL semantics for DTMCs

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

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

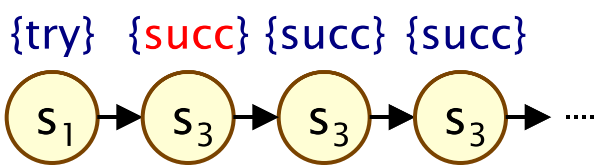
- Examples
  - $s_3 \vDash succ$
  - $s_1 \vDash try \wedge \neg fail$



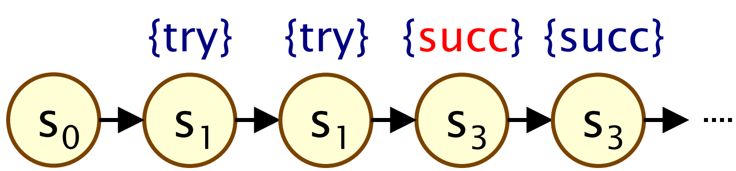
PCTL semantics for DTMCs

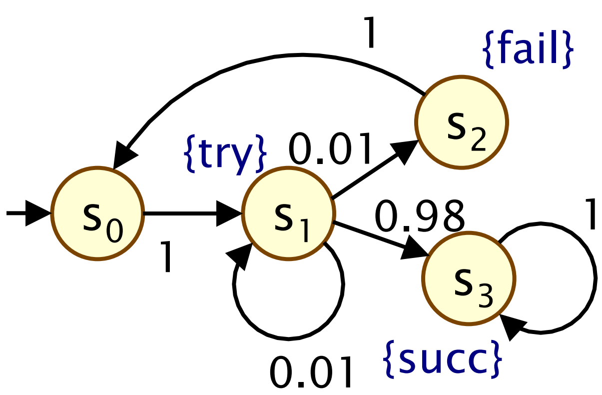
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \vDash X\ \phi \quad \Leftrightarrow \quad s_1 \vDash \phi$
  - $\omega \vDash \phi_1\ U^{\leq k}\ \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \vDash \phi_2$ and $\forall j<i,\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

- Some examples of satisfying paths:
  - X succ



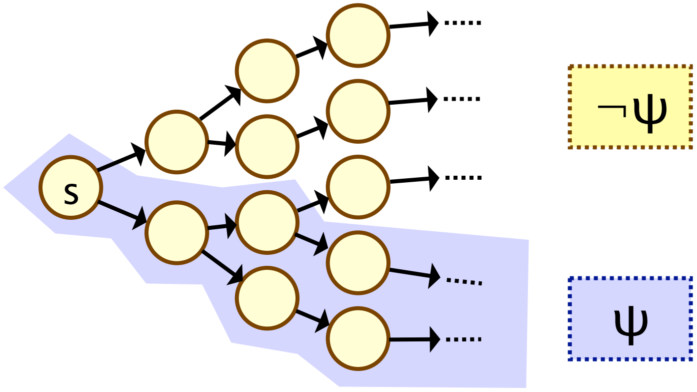
  - ¬fail U succ





PCTL semantics for DTMCs

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



Prob(s, ψ) ~ p ?

More PCTL…

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

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

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

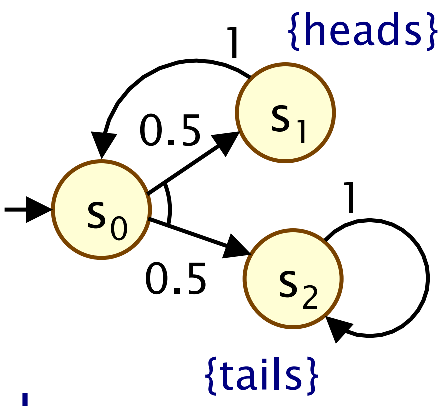
Qualitative vs. quantitative properties

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

- A PCTL property $P_{\sim p}$ [ψ] is…
  - qualitative when p is either 0 or 1
  - quantitative when p is in the range (0,1)

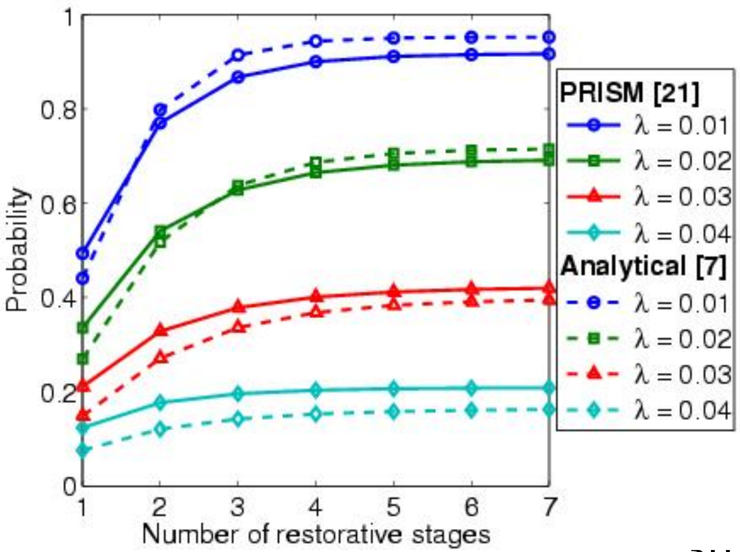
- $P_{>0}$ [F ϕ] is identical to EF ϕ
  - there exists a finite path to a ϕ-state

- $P_{\geq 1}$ [F ϕ] is (similar to but) weaker than AF ϕ
  - e.g. AF "tails" (CTL) ≠ $P_{\geq 1}$ [F "tails"] (PCTL)



Quantitative properties

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



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

Some real PCTL examples

reliability

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

performance

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

fairness

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

Overview (Part 1)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- LTL model checking
- Costs and rewards

PCTL model checking for DTMCs

- Algorithm for PCTL model checking [CY88,HJ94,CY95]
  - inputs: DTMC $D=(S,s_{init},P,L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = set of states satisfying ϕ

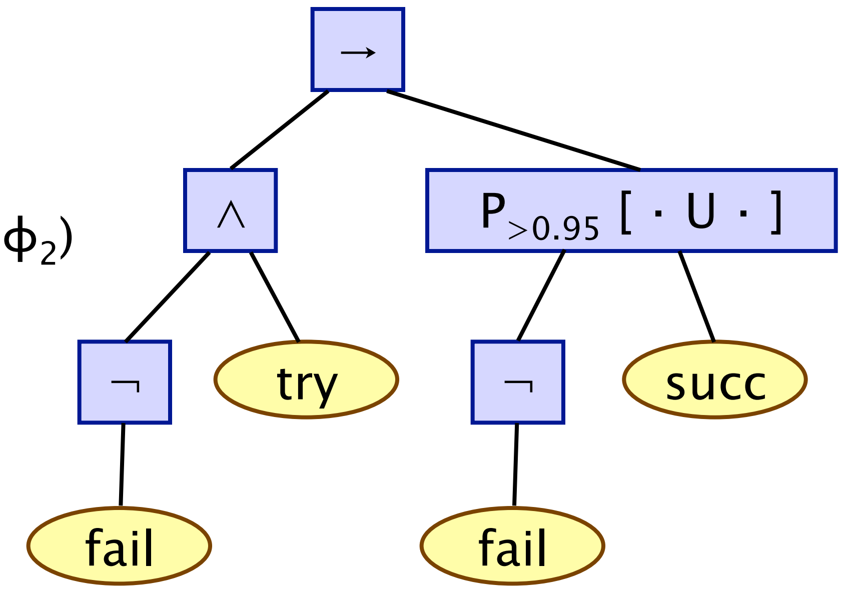
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - sometimes, want to check that $s \vDash \phi \ \forall \ s \in S$, i.e. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{init} \vDash \phi$, i.e. if $s_{init} \in Sat(\phi)$

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

PCTL model checking for DTMCs

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

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

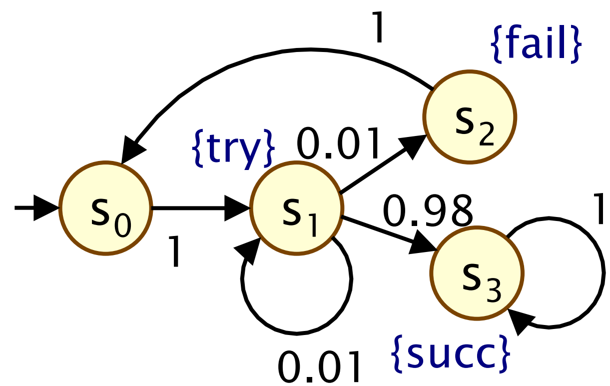


- For the $P_{\sim p}$ [ψ] operator
  - need to compute the probabilities Prob(s, ψ) for all states $s \in S$
  - focus here on "until" case: $\psi = \phi_1 \text{ U } \phi_2$

PCTL next – Example

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

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



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

PCTL until for DTMCs

- Computation of probabilities Prob(s, ϕ_1 U ϕ_2) for all s $\in$ S
- First, identify all states where the probability is 1 or 0
  - $S^{yes} = Sat(P_{\geq 1}[\ \phi_1\ U\ \phi_2\ ])$
  - $S^{no} = Sat(P_{\leq 0}[\ \phi_1\ U\ \phi_2\ ])$
- Then solve linear equation system for remaining states

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

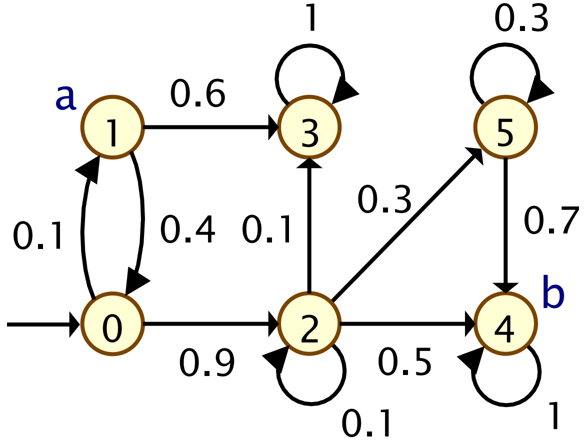
$$\text{Prob}(s, \phi_1 \text{ U } \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot \text{Prob}(s', \phi_1 \text{ U } \phi_2) & \text{otherwise} \end{cases}$$

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

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

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]



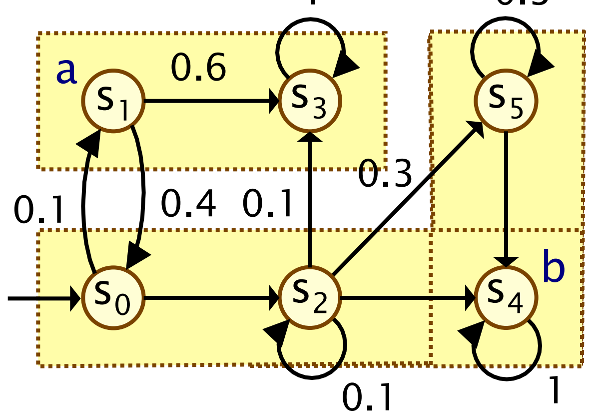
Precomputation – Prob0

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

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

Example:

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



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

Prob0 algorithm

$\textsc{Prob0}(Sat(\phi_1), Sat(\phi_2))$

$R := Sat(\phi_2)$

$done :=$ **false**

while $(done =$ **false**$)$

    $R' := R \cup \{s \in Sat(\phi_1) \mid \exists s' \in R \,.\, \mathbf{P}(s, s') > 0\}$

    if $(R' = R)$ **then** $done :=$ **true**

    $R := R'$

endwhile

return $S \backslash R$

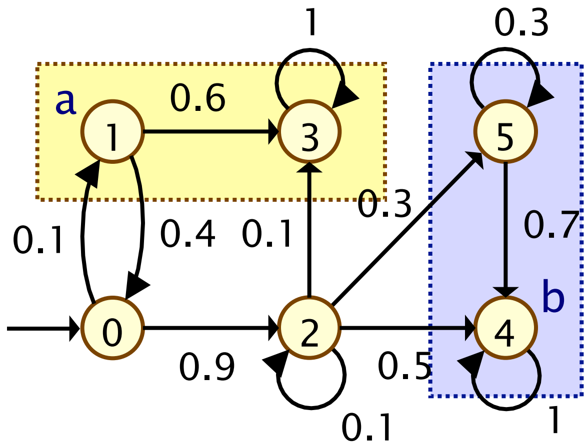
- Note: can be formulated as a least fixed point computation
  - also well suited to computation with binary decision diagrams

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

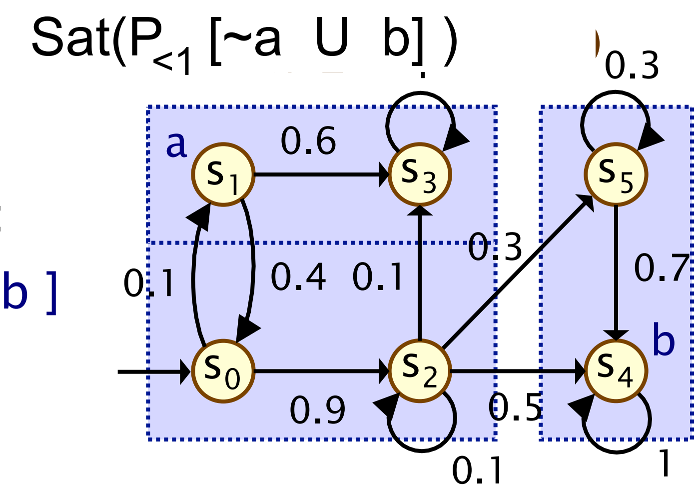
Sat($P_{\geq 1}$ [¬a U b])

Precomputation – Prob1

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

Example:

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



$S^{yes} =$

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

Prob1 algorithm

$\textsc{Prob1}(Sat(\phi_1), Sat(\phi_2), S^{no})$

$R := S^{no}$

$done :=$ **false**

while $(done =$ **false**$)$

$\quad R' := R \cup \{s \in (Sat(\phi_1) \backslash Sat(\phi_2)) \mid \exists s' \in R \,.\, \mathbf{P}(s,s') > 0\}$

$\quad$ **if** $(R' = R)$ **then** $done :=$ **true**

$\quad R := R'$

endwhile

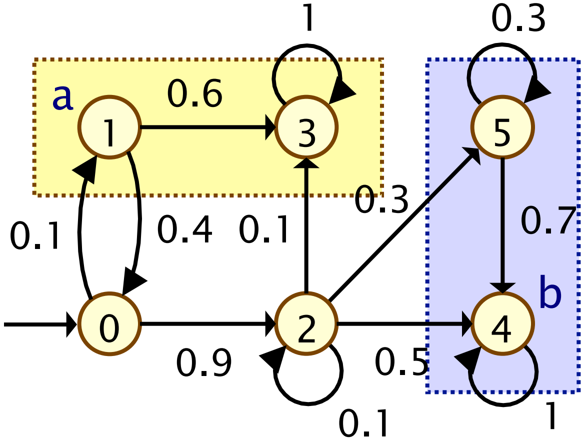
return $S \backslash R$

PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



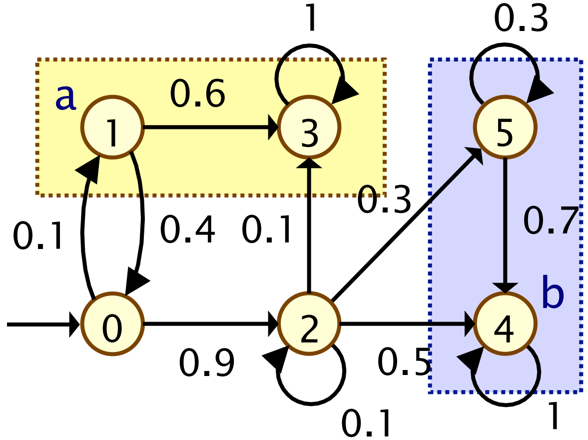
S^{yes} =

Sat($P_{\geq 1}$ [¬a U b])

PCTL until – Example

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

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



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

$x_4 = x_5 = 1$

$x_1 = x_3 = 0$

$x_0 = 0.1x_1 + 0.9x_2 = 0.8$

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

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

note: this Prob-with-underscore is called "probability-vector"

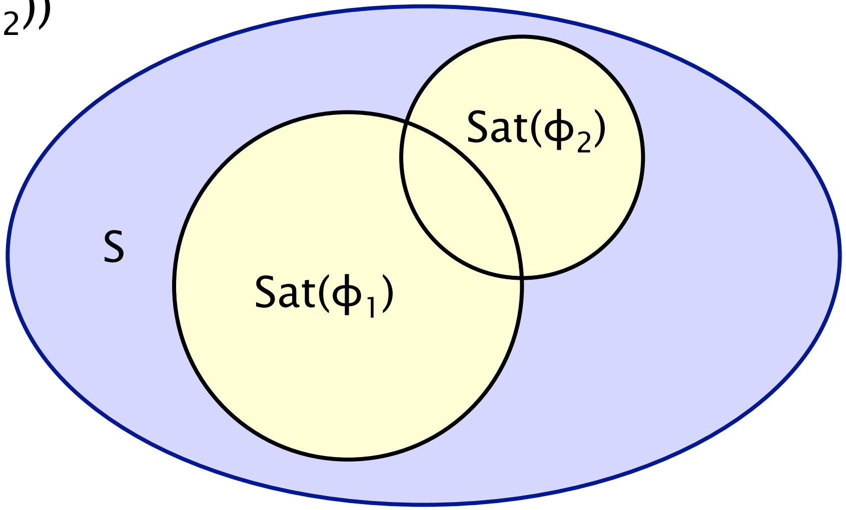
Sat($P_{>0.8}$ [$\neg$a U b]) = { s_2, s_4, s_5 }

PCTL bounded until for DTMCs

- Computation of probabilities for PCTL $U^{\leq k}$ operator
  - $Sat(P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]) = \{\ s \in S \mid Prob(s,\ \phi_1\ U^{\leq k}\ \phi_2) \sim p\ \}$
  - need to compute $Prob(s,\ \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$
- First identify (some) states where probability is trivially 1/0
  - $S^{yes} = Sat(\phi_2)$
  - $S^{no} = S \setminus (Sat(\phi_1) \cup Sat(\phi_2))$

then calculate

$S^{?} = S \setminus (S^{yes} \cup S^{no})$



PCTL bounded until for DTMCs

- Simultaneous computation of vector $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2)$
  - i.e. probabilities $\text{Prob}(s, \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$
- Iteratively define in terms of matrices and vectors
  - define matrix $\mathbf{P'}$ as follows: $\mathbf{P'}(s,s') = \mathbf{P}(s,s')$ if $s \in S^?$, $\mathbf{P'}(s,s') = 1$ if $s \in S^{yes}$ and $s=s'$, $\mathbf{P'}(s,s') = 0$ otherwise
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq 0}\ \phi_2) = \underline{\phi}_2$
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2) = \mathbf{P'} \cdot \underline{\text{Prob}}(\phi_1\ U^{\leq k-1}\ \phi_2)$
  - requires k matrix-vector multiplications
- Note that we could express this in terms of matrix powers
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2) = (\mathbf{P'})^k \cdot \underline{\phi}_2$ and compute $(\mathbf{P'})^k$ in $\log_2 k$ steps
  - but this is actually inefficient: $(\mathbf{P'})^k$ is much less sparse than $\mathbf{P'}$

PCTL bounded until – Example

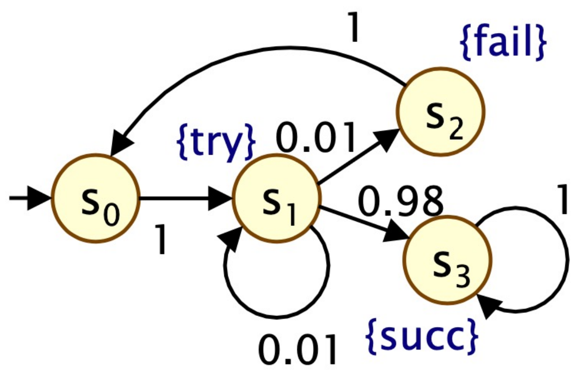
- Model check: $P_{>0.98}$ [$F^{\leq 2}$ succ] ≡ $P_{>0.98}$ [true $U^{\leq 2}$ succ]
  - Sat (true) = S = $\{s_0,s_1,s_2,s_3\}$, Sat(succ) = $\{s_3\}$
  - $S^{yes} = \{s_3\}$, $S^{no} = \varnothing$, $S^? = \{s_0,s_1,s_2\}$, **P' = P**
  - $\underline{\text{Prob}}(\text{true } U^{\leq 0} \text{ succ}) = \underline{\text{succ}} = [0, 0, 0, 1]$

$$\underline{\text{Prob}}(\text{true } U^{\leq 1} \text{ succ}) = P' \cdot \underline{\text{Prob}}(\text{true } U^{\leq 0} \text{ succ}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.98 \\ 0 \\ 1 \end{bmatrix}$$

$$\underline{\text{Prob}}(\text{true } U^{\leq 2} \text{ succ}) = P' \cdot \underline{\text{Prob}}(\text{true } U^{\leq 1} \text{ succ}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0.98 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.98 \\ 0.9898 \\ 0 \\ 1 \end{bmatrix}$$

  - Sat($P_{>0.98}$ [$F^{\leq 2}$ succ]) = $\{s_1, s_3\}$

The construction of P’ for bounded Until

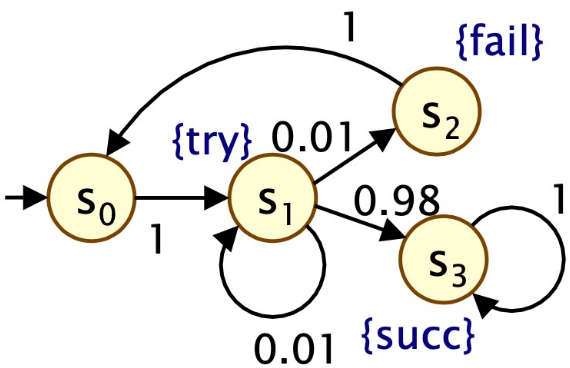


0	1	0	0
0	0.01	0.01	0.98
1	0	0	0
0	0	0	1

The probability matric P of the DTMC on the left.

- Consider as an example to check whether the DTMC satisfies $P_{>0.99}$[try ∨ ¬fail **U**$^{\leq 2}$ succ].
  1. Calculate first the probability vector Prob[try ∨ ¬fail **U**$^{\leq 2}$ succ].
  2. From there you can calculate the set Sat($P_{>0.99}$[try ∨ ¬fail **U**$^{\leq 2}$ succ]).
  3. **If the initial state s_0 is in** the blue Sat-set then the property $P_{>0.99}$[try ∨ ¬fail **U**$^{\leq 2}$ succ holds on the DTMC.

The construction of P' for bounded Until

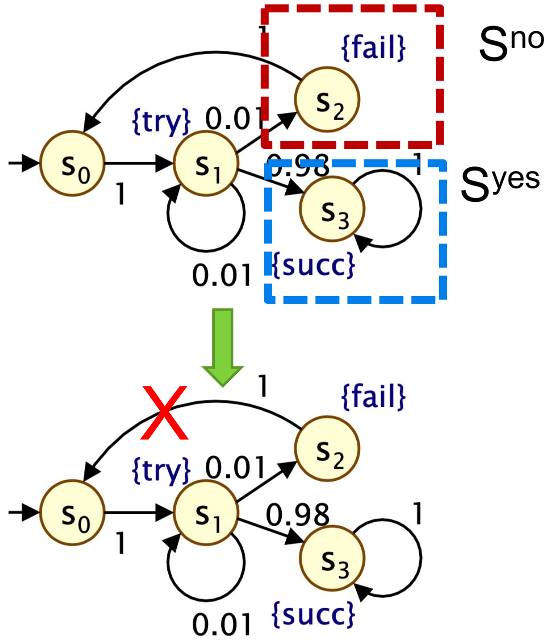


0	1	0	0
0	0.01	0.01	0.98
1	0	0	0
0	0	0	1

The probability matric P of the DTMC on the left.

- To calculate the probability vector Prob[try ∨ ¬fail **U**$^{\leq 2}$ succ], we would like to use the matrix P above, however it will also "contain" transitions that cause you to break the green-property. So the idea is to use a "modified" matrix P'.
- We pre-calculate first the S^{yes} = Sat(succ) = {s3}. On all states in S^{yes}, you have the green property immediately (in 0 step).
- We pre-calculate S^{no}, we take S^{no} = Sat(¬ (try ∨ ¬fail) ∧ ¬ succ) = { s2 }. Executions starting from S^{no} won't satisfy your green-property above,

The construction of P’ for bounded Until



P :

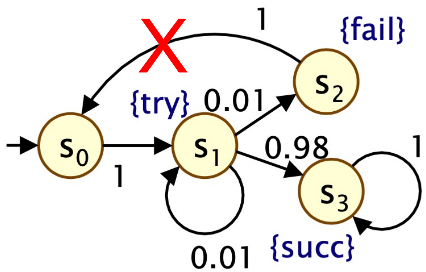
0	1	0	0
0	0.01	0.01	0.98
1	0	0	0
0	0	0	1

P’ :

0	1	0	0
0	0.01	0.01	0.98
0	0	0	0
0	0	0	1

1. We remove outgoing arrows from the states in S^{no} and S^{yes}.
2. We keep all arrows that go out from states which are **not** in S^{no} nor S^{yes}.
3. We add a self-loop $s \rightarrow s$ with probability 1 for any state s in S^{yes}.

Using P’ for bounded Until

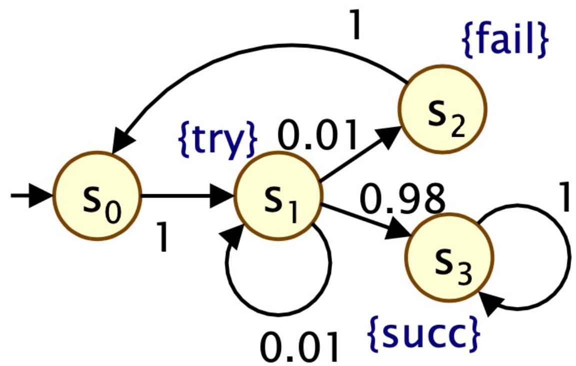


P’ :

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- We now use P’ to iteratively calcualte Prob[try ∨ ¬fail $\mathbf{U}^{\leq 2}$ succ]
- From S^{yes} you know that Prob[try ∨ ¬fail $\mathbf{U}^{\leq 0}$ succ] = $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$
- Prob[try ∨ ¬fail $\mathbf{U}^{\leq 1}$ succ] = P’ × $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ = $\begin{pmatrix} 0 \\ 0.98 \\ 0 \\ 1 \end{pmatrix}$
- Prob[try ∨ ¬fail $\mathbf{U}^{\leq 2}$ succ] = P’ × $\begin{pmatrix} 0 \\ 0.98 \\ 0 \\ 1 \end{pmatrix}$ = $\begin{pmatrix} 0.98 \\ 0.9898 \\ 0 \\ 1 \end{pmatrix}$

So, does the property hold?



- We have calculated <u>Prob</u>[try ∨ ¬fail **U**$^{\leq 2}$ succ] = [0.98, 0.9898, 0, 1]
- So, the set Sat($P_{>0.99}$[try ∨ ¬fail **U**$^{\leq 2}$ succ]) = $\{s_3\}$
- So we conclude that the DTMC does **not** satisfy the claimed property $P_{>0.99}$[try ∨ ¬fail **U**$^{\leq 2}$ succ].

PCTL model checking – Summary

- Computation of set $Sat(\Phi)$ 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, $O(|S|^2)$
  – $\Phi_1\ U^{\leq k}\ \Phi_2$: 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 $|\Phi|$ and polynomial in $|S|$

Overview (Part 1)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- LTL model checking
- Costs and rewards

Limitations of PCTL

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

- More expressive logics can be used, for example:
  - LTL [Pnu77] – (non-probabilistic) linear-time temporal logic
  - PCTL* [ASB+95,BdA95] – which subsumes both PCTL and LTL
  - both allow path operators to be combined
  - (in PCTL, $P_{\sim p}$ […] always contains a single temporal operator)

- Another direction: extend DTMCs with costs and rewards…

LTL – Linear temporal logic

- LTL syntax (path formulae only)
  - $\psi ::= \text{true} \mid a \mid \psi \wedge \psi \mid \neg\psi \mid X\,\psi \mid \psi\,U\,\psi$
  - where $a \in AP$ is an atomic proposition
  - usual equivalences hold: $F\,\phi \equiv \text{true}\;U\;\phi$, $G\,\phi \equiv \neg(F\,\neg\phi)$

- LTL semantics (for a path ω)
  - $\omega \vDash \text{true}$ always
  - $\omega \vDash a \Leftrightarrow a \in L(\omega(0))$
  - $\omega \vDash \psi_1 \wedge \psi_2 \Leftrightarrow \omega \vDash \psi_1$ and $\omega \vDash \psi_2$
  - $\omega \vDash \neg\psi \Leftrightarrow \omega \nvDash \psi$
  - $\omega \vDash X\,\psi \Leftrightarrow \omega[1\ldots] \vDash \psi$
  - $\omega \vDash \psi_1\,U\,\psi_2 \Leftrightarrow \exists k \geq 0$ s.t. $\omega[k\ldots] \vDash \psi_2 \wedge \forall i<k\; \omega[i\ldots] \vDash \psi_1$

where $\omega(i)$ is i^{th} state of ω, and $\omega[i\ldots]$ is suffix starting at $\omega(i)$

LTL examples

- $(\mathrm{F}\ tmp\_fail_1) \wedge (\mathrm{F}\ tmp\_fail_2)$
  - "both servers suffer temporary failures at some point"
- GF ready
  - "the server always eventually returns to a ready-state"
- FG error
  - "an irrecoverable error occurs"
- G (req → X ack)
  - "requests are always immediately acknowledged"

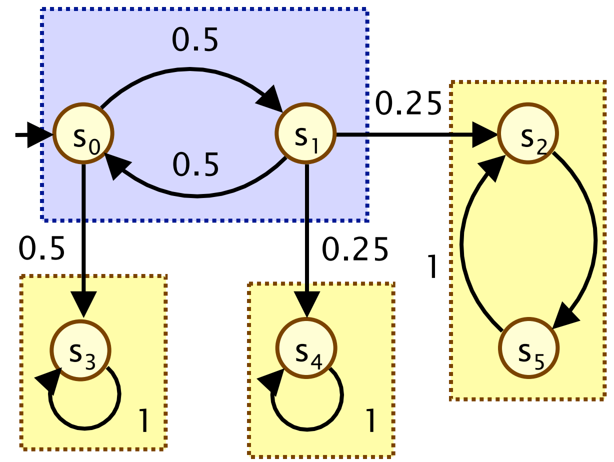
LTL for DTMCs

- Same idea as PCTL: probabilities of sets of path formulae
  - for a state s of a DTMC and an LTL formula ψ:
  - $\mathrm{Prob}(s, \psi) = \mathrm{Pr}_s \{ \omega \in \mathrm{Path}(s) \mid \omega \vDash \psi \}$
  - all such path sets are measurable [Var85]
- A (probabilistic) LTL specification often comprises an LTL (path) formula and a probability bound
  - e.g. $P_{\geq 1}$ [GF ready] – "with probability 1, the server always eventually returns to a ready-state"
  - e.g. $P_{<0.01}$ [FG error] – "with probability at most 0.01, an irrecoverable error occurs"
- PCTL* subsumes both LTL and PCTL
  - e.g. $P_{>0.5}$ [GF crit_1] $\wedge$ $P_{>0.5}$ [GF crit_2]

Fundamental property of DTMCs

- Strongly connected component (SCC)
  - maximally strongly connected set of states
- Bottom strongly connected component (BSCC)
  - SCC T from which no state outside T is reachable from T

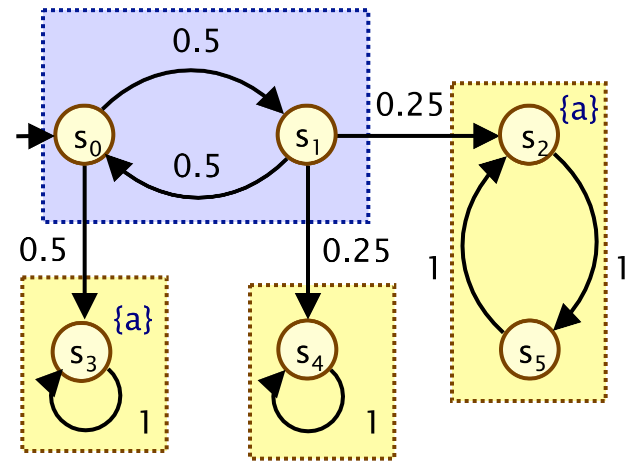
- Fundamental property of DTMCs:
  - "with probability 1,
    a BSCC will be reached
    and all of its states
    visited infinitely often"



- Formally:
  - Pr_s { $\omega \in Path(s)$ | $\exists$ i≥0, $\exists$ BSCC T such that
    $\forall$ j≥i $\omega(i) \in$ T and
    $\forall$ s'∈T $\omega(k)$ = s' for infinitely many k } = 1

LTL model checking for DTMCs

- LTL model checking for DTMCs relies on:
  - computing probability of reaching a set of "accepting" BSCCs
  - e.g. for two simple LTL formulae: GF a ("always eventually a"), FG a ("eventually always a') we have:



- Prob(s, GF a) = Prob(s, F T_{GFa})
  - where T_{GFa} = union of all BSCCs containing some state satisfying a

- Prob(s, FG a) = Prob(s, F T_{FGa})
  - where T_{FGa} = union of all BSCCs containing only a-states

- To extend this idea to arbitrary LTL formula, we use ω-automata…

Example:

Prob(s_0, GF a)

= Prob(s_0, F T_{GFa})

= Prob(s_0, F $\{s_3,s_2,s_5\}$)

= 2/3 + 1/6 = 5/6

Deterministic Rabin automata

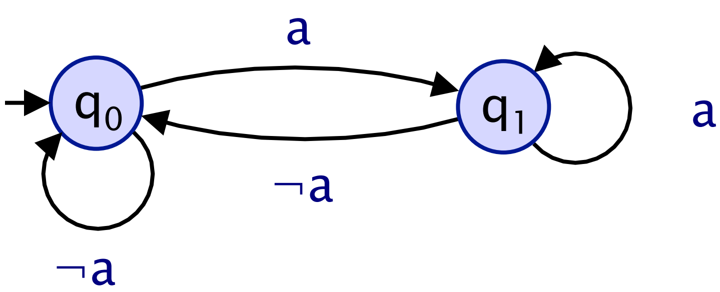
- ω-automata represent sets of infinite words
  - e.g. Buchi automata, Rabin automata, …
  - for probabilistic model checking, need deterministic automata
  - so we use deterministic Rabin automata (DRAs)

- A deterministic Rabin automaton is a tuple $(Q, \Sigma, \delta, q_0, Acc)$:
  - Q is a finite set of states, $q_0 \in Q$ is an initial state
  - Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function
  - $Acc = \{ (L_i, K_i) \}_{i=1..k} \subseteq 2^Q \times 2^Q$ is an acceptance condition

- A run of a word on a DRA is accepting iff:
  - for some pair (L_i, K_i), the states in L_i are visited finitely often and (some of) the states in K_i are visited infinitely often
  - or in LTL: $\bigvee_{1 \leq i \leq k} (FG \neg L_i \wedge GF K_i)$

LTL & DRAs

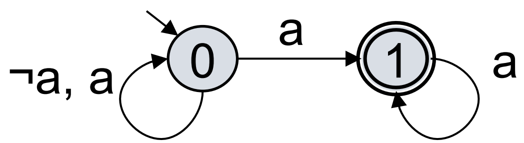
- Example: DRA for FG a
  - acceptance condition is Acc = { $(\{q_0\},\{q_1\})$ }



- Can convert any LTL formula ψ on atomic propositions AP
  - into an equivalent DRA A_ψ over alphabet 2^{AP}
  - i.e. $\omega \vDash \psi \Leftrightarrow \mathrm{trace}(\omega) \in L(A_\psi)$ for any path ω
  - can potentially incur a double exponential blow-up (but, in practice, this does not occur and ψ is small anyway)
- LTL model checking for DTMCs – the basic idea
  - construct product of DTMC D and DRA A_ψ
  - compute $\mathrm{Prob}^D(s, \psi)$ on product DTMC $D \otimes A$

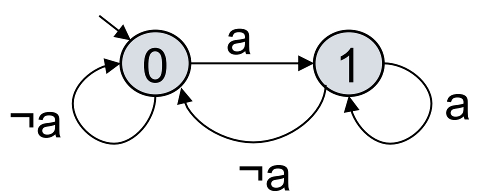
Buchi vs Rabin

- Consider the LTL property ◊□a. This can be described by this Buchi automaton, wirth {1} as the accepting state:



Notice that this Buchi is non-deterministic. As such, we can use it for model checking on a probabilistic model such as DTMC.

- We can however represent the property with a deterministic Rabin automaton, with the pair ({0}, {1}) as its accepting condition.



Product DTMC for a DRA

- The product DTMC $D \otimes A$ for:
  - for DTMC $D = (S, s_{init}, \mathbf{P}, L)$ and
  - and (total) DRA $A = (Q, \Sigma, \delta, q_0, \{ (L_i, K_i) \}_{i=1..k})$
  - is the DTMC $(S \times Q, (s_{init}, q_{init}), \mathbf{P}', L')$ where:

$$q_{init} = \delta(q_0, L(s_{init}))$$

$$\mathbf{P}'((s_1,q_1),(s_2,q_2)) = \begin{cases} \mathbf{P}(s_1,s_2) & \text{if } q_2 = \delta(q_1, L(s_2)) \\ 0 & \text{otherwise} \end{cases}$$

$$l_i \in L'(s,q) \text{ if } q \in L_i \text{ and } k_i \in L'(s,q) \text{ if } q \in K_i$$

- Note:
  - $D \otimes A$ can be seen as unfolding of D where q for each state (s,q) records state of automata A for path fragment so far
  - since A is deterministic, $D \otimes A$ is a DTMC
  - each path in D has a corresponding (unique) path in $D \otimes A$
  - the probabilities of paths in D are preserved in $D \otimes A$

Product DTMC for a DRA

- For DTMC D and DRA A

$$\mathrm{Prob}^{D}(s, A) = \mathrm{Prob}^{D\otimes A}((s,q_s), \vee_{1\leq i\leq k} (FG\ \neg l_i \wedge GF\ k_i))$$

  - where $q_s = \delta(q_0, L(s))$
- Hence:

$$\mathrm{Prob}^{D}(s, A) = \mathrm{Prob}^{D\otimes A}((s,q_s), F\ T_{Acc})$$

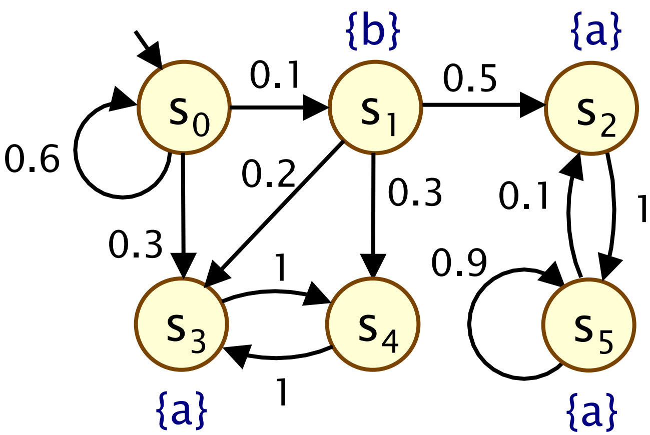
  - where T_{Acc} is the union of all accepting BSCCs in $D\otimes A$
  - an accepting BSCC T of $D\otimes A$ is such that, for some $1\leq i\leq k$, no states in T satisfy l_i and some state in T satisfies k_i

- Reduces to computing BSCCs and reachability probabilities
  - so overall complexity for LTL is doubly exponential in $|\psi|$, polynomial in $|M|$; but can be reduced to singly exponential

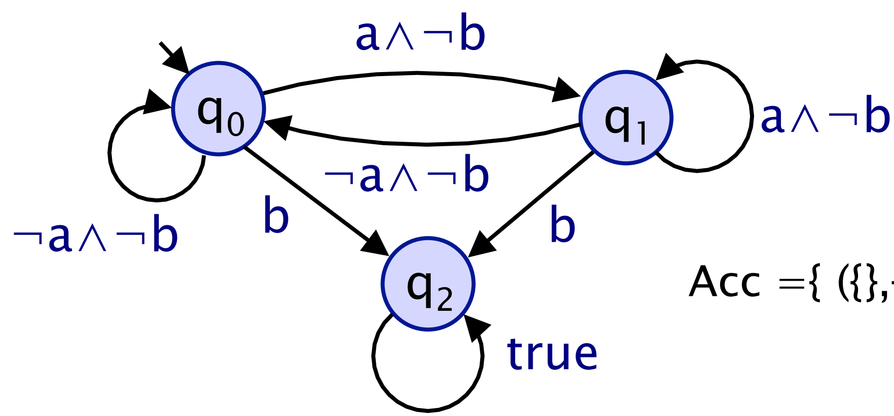
Example: LTL for DTMCs

- Compute Prob(s_0, G¬b ∧ GF a) for DTMC D:

DTMC D



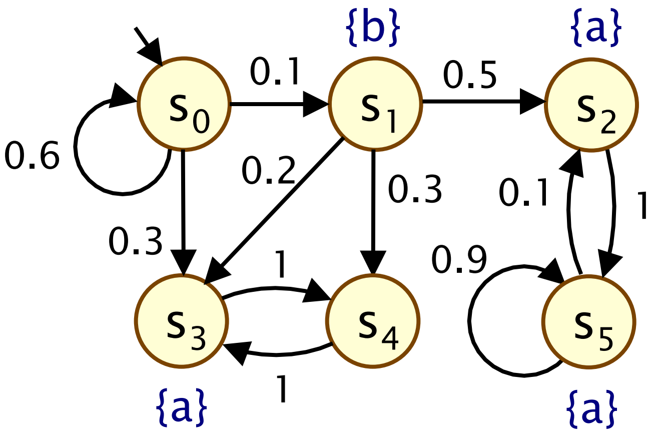
DRA A_ψ for ψ = G¬b ∧ GF a



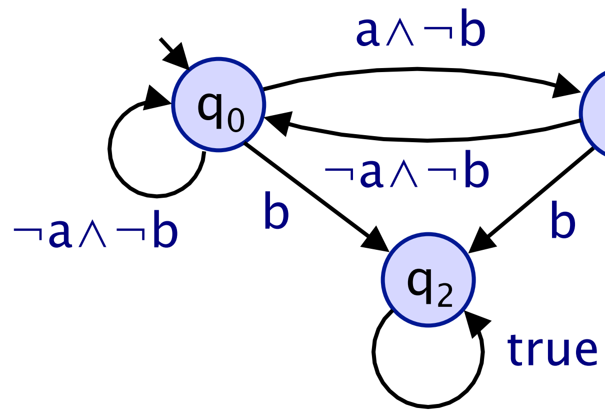
Acc ={ ({},{q_1}) }

Example: LTL for DTMCs

DTMC D

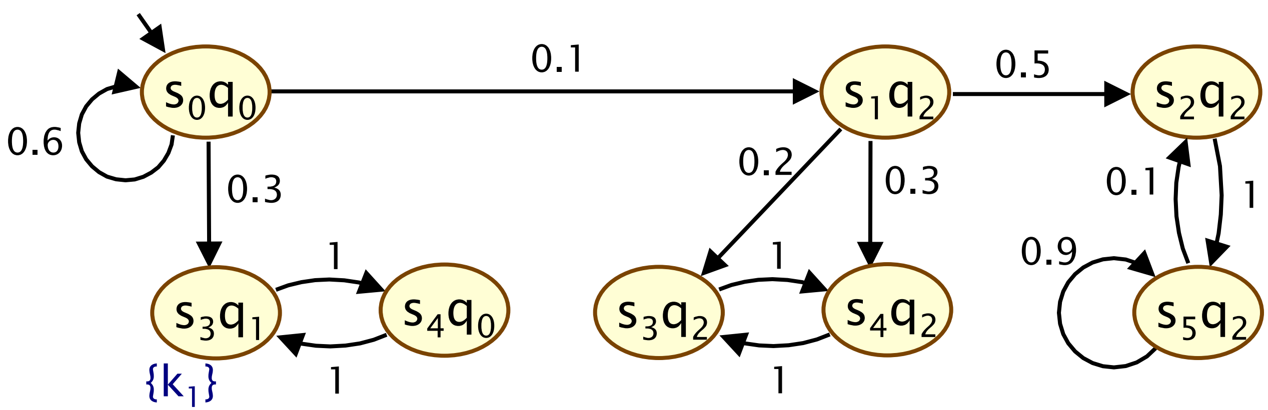


DRA A_ψ for $\psi = G\neg b \wedge GF\, a$



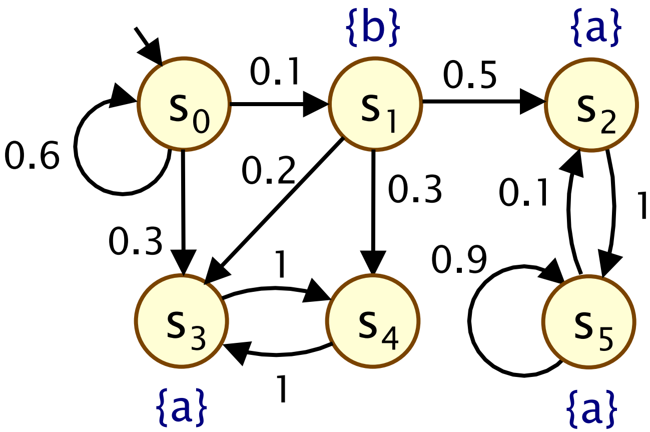
Acc ={ ({},{q_1}) }

Product DTMC $D \otimes A_\psi$

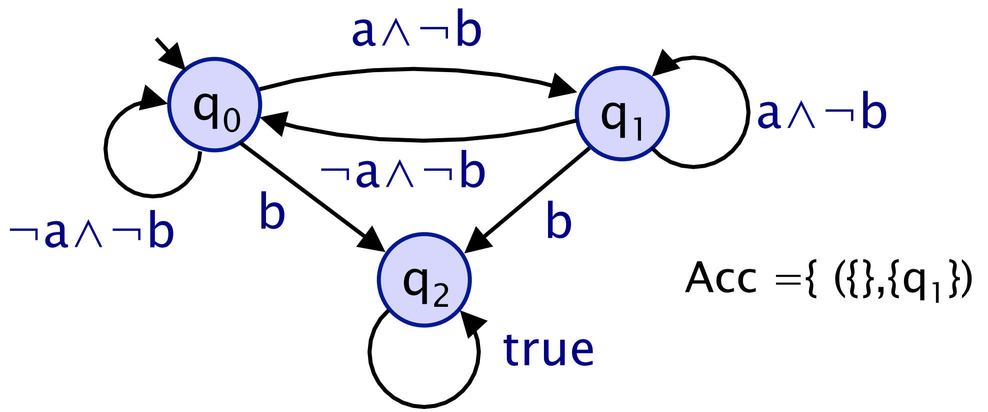


Example: LTL for DTMCs

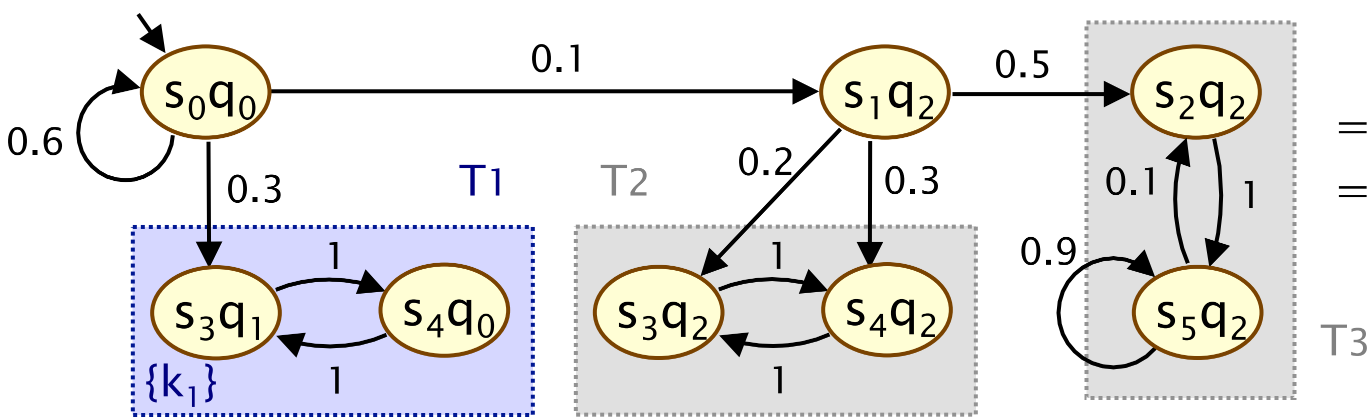
DTMC D



DRA A_ψ for $\psi = G\neg b \wedge GF\ a$



Product DTMC $D \otimes A_\psi$



$Prob^D(s, \psi)$

$= Prob^{D\otimes A\psi}(F\ T_1)$

$= 3/4.$

Overview (Part 1)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- LTL model checking
- Costs and rewards

Costs and rewards

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

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

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

Reward-based properties

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

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

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

- Cumulative properties
  - the expected cumulated reward over some period

DTMC reward structures

- For a DTMC $(S,s_{init},\mathbf{P},L)$, a reward structure is a pair $(\underline{\rho},\boldsymbol{\iota})$
  - $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is the state reward function (vector)
  - $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geq 0}$ 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, $\boldsymbol{\iota}$ is not used

- Examples (for use with cumulative properties)
  - "time-steps": $\underline{\rho}$ returns 1 for all states and $\boldsymbol{\iota}$ is zero (equivalently, $\underline{\rho}$ is zero and $\boldsymbol{\iota}$ returns 1 for all transitions)
  - "number of messages lost": $\underline{\rho}$ is zero and $\boldsymbol{\iota}$ 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 $\boldsymbol{\iota}$ 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

  - $\phi \;::=\; \ldots \;\mid\; P_{\sim p}\,[\,\psi\,] \;\mid\; R_{\sim r}\,[\,I^{=k}\,] \;\mid\; R_{\sim r}\,[\,C^{\leq k}\,] \;\mid\; R_{\sim r}\,[\,F\,\phi\,]$

"instantaneous" "cumulative" "reachability"

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

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

Types of reward formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - "the expected value of the state reward at time-step k is ~r"
  - e.g. "the expected queue size after exactly 90 seconds"
- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - "the expected reward cumulated up to time-step k is ~r"
  - e.g. "the expected power consumption over one hour"
- Reachability: $R_{\sim r}$ [F ϕ]
  - "the expected reward cumulated before reaching a state satisfying ϕ is ~r"
  - e.g. "the expected time for the algorithm to terminate"

Reward formula semantics

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

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

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

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

Reward formula semantics

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

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

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

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

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

Model checking reward properties

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$]
  - variant of the method for computing bounded until probabilities
  - solution of recursive equations

- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to computing until probabilities
  - precomputation phase (identify infinite reward states)
  - then reduces to solving a system of linear equation

- For more details, see e.g. [KNP07a]

Summary

- Probabilistic model checking
  - automated quantitative verification of stochastic systems
  - to model randomisation, failures, …
- 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

- Tomorrow: Markov decision processes (MDPs)