Automated Verification Techniques for Probabilistic Systems

Vojtěch Forejt
Marta Kwiatkowska
Gethin Norman
Dave Parker

SFM-11:CONNECT Summer School, Bertinoro, June 2011

EU-FP7: CONNECT LSCITS/PSS VERIWARE

Overview

- Lecture 1 (9am-11am)
  - Introduction to Modelling and Quantitative Verification
  - Marta Kwiatkowska
- Invited lecture: Christel Baier
  - Component and Connector Modelling Formalisms
- Lecture 2 (2.30pm-4pm)
  - Quantitative Compositional Verification
  - Dave Parker
- Lab session (4.30pm-6pm)
  - Modelling and Compositional Verification of Probabilistic Component-Based Systems using PRISM
  - Dave Parker

- http://www.prismmodelchecker.org/courses/sfm11connect/

Part 1

Introduction

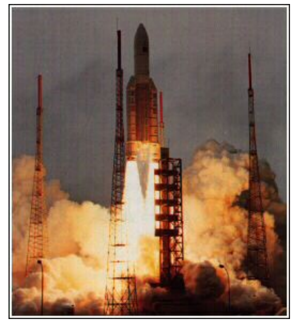
Quantitative verification

- Formal verification…
  - is the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
- Quantitative verification
  - applies formal verification techniques to the modelling and analysing of non-functional aspects of system behaviour (e.g. probability, time, cost, …)
- Probabilistic model checking…
  - is a an automated quantitative verification technique for systems that exhibit probabilistic behaviour

Why formal verification?

- Errors in computerised systems can be costly…

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

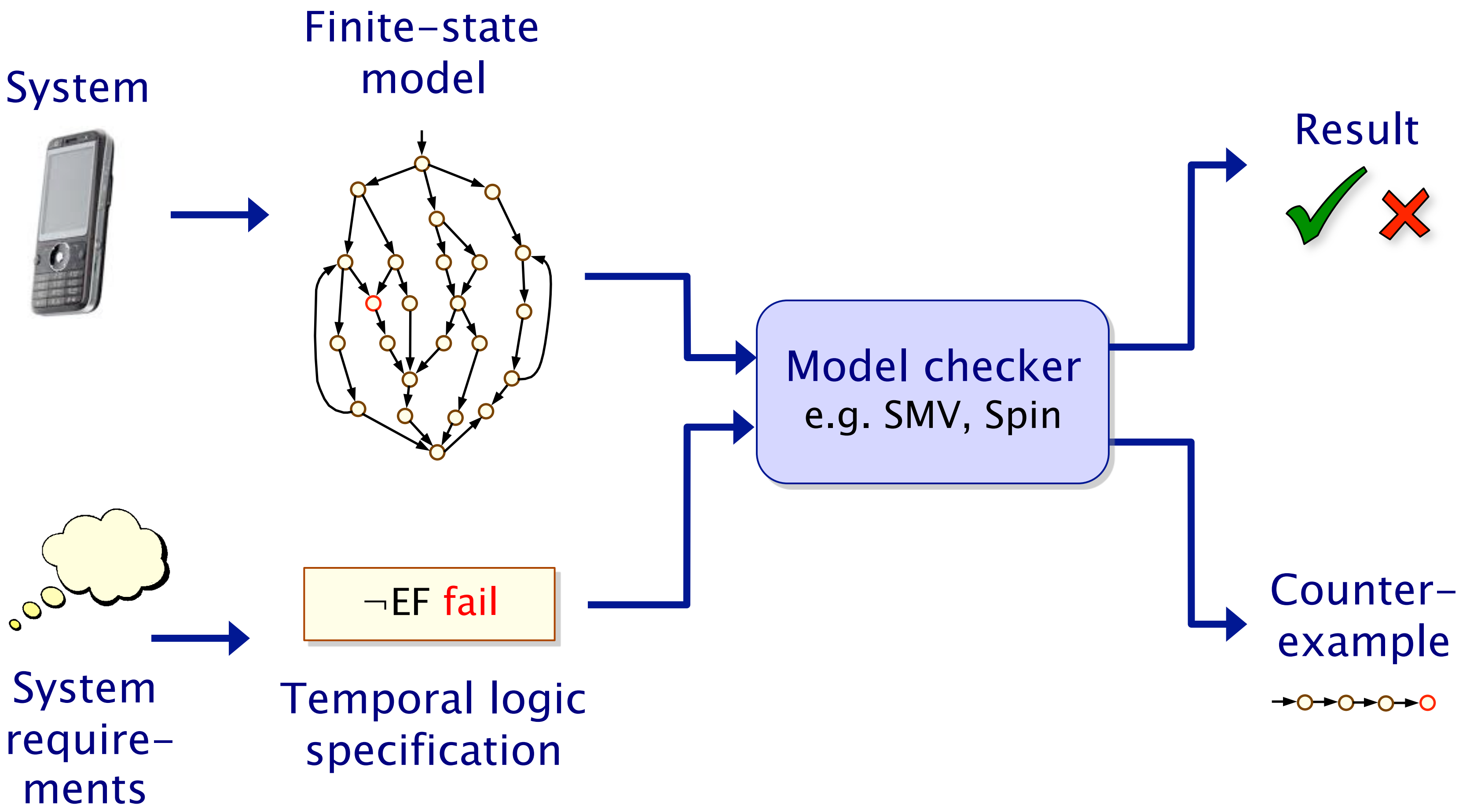


Ariane 5 (1996)
Self-destructs 37secs into maiden launch.
Cause: uncaught overflow exception.

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

- Why verify?
  - "Testing can only show the presence of errors, not their absence." [Edsger Dijstra]

Model checking



Why probability?

- Some systems are inherently probabilistic…

- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding

- Examples: real-world protocols featuring randomisation:
  - Randomised back-off schemes
    - CSMA protocol, 802.11 Wireless LAN
  - Random choice of waiting time
    - IEEE1394 Firewire (root contention), Bluetooth (device discovery)
  - Random choice over a set of possible addresses
    - IPv4 Zeroconf dynamic configuration (link-local addressing)
  - Randomised algorithms for anonymity, contract signing, …

Why probability?

- Some systems are inherently probabilistic…

- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding

- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service

- Examples:
  - computer networks, embedded systems
  - power management policies
  - nano-scale circuitry: reliability through defect-tolerance

Why probability?

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

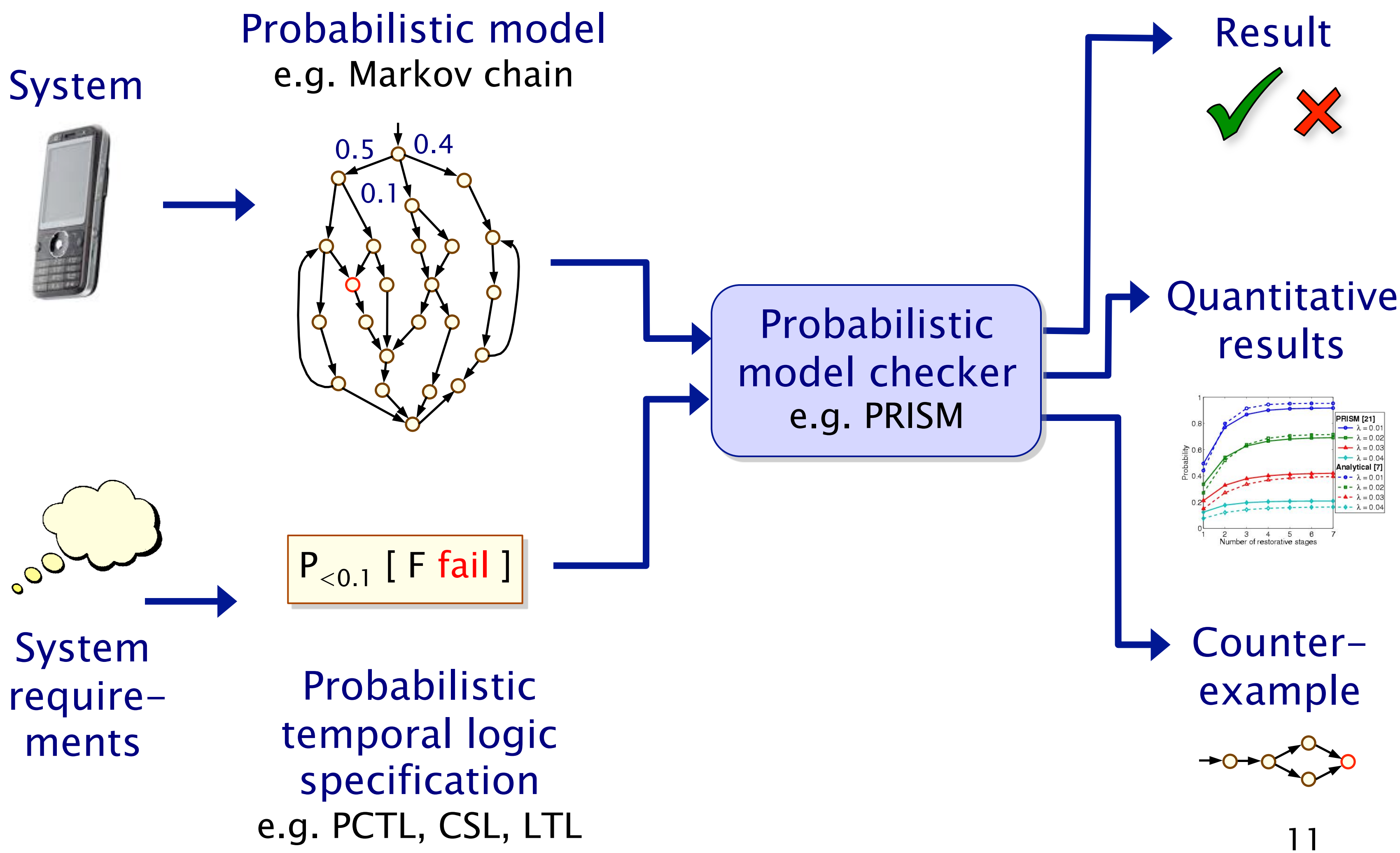
Verifying probabilistic systems

- We are not just interested in correctness

- We want to be able to quantify non-functional properties:
  - 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 the disaster service provider 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



CONNECTed probabilistic systems

- Many of the probabilistic systems that we want to verify are naturally decomposed into sub-systems
  - communication protocols, power management systems, …
- Need modelling formalisms to capture this behaviour
  - Markov decision processes (probabilistic automata)
  - combine probabilistic and nondeterministic behaviour
  - analysis non-trivial – need automated techniques and tools
- Component-based systems
  - offer opportunities to exploit their structure
  - compositional probabilistic verification: assume-guarantee
  - more generally, quantitative properties

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)

Overview

- Lectures 1 and 2:
  - 1 – Introduction
  - 2 – Discrete-time Markov chains
  - 3 – Markov decision processes
  - 4 – Compositional probabilistic verification

- Course materials available here:
  - http://www.prismmodelchecker.org/courses/sfm11connect/
  - lecture slides, reference list, tutorial chapter, lab session

Part 2

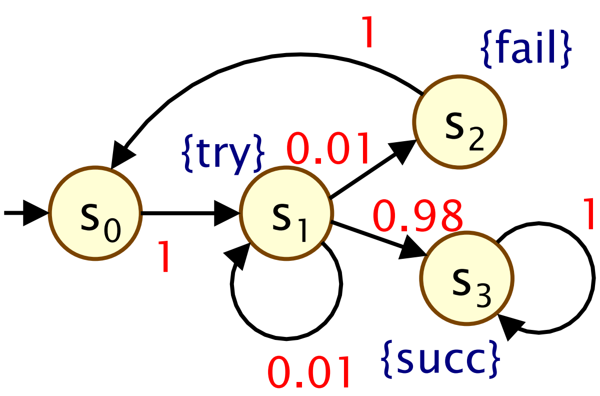
Discrete-time Markov chains

Overview (Part 2)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- Other properties: LTL, costs and rewards
- Case study: Bluetooth device discovery

Discrete-time Markov chains

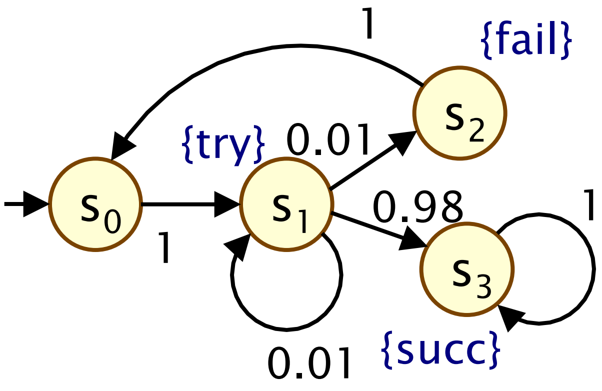
- Discrete-time Markov chains (DTMCs)
  - state-transition systems augmented with probabilities
- States
  - discrete set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability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

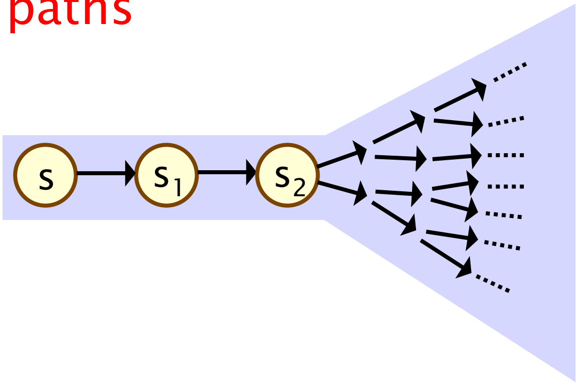
- Alternative definition: a DTMC is:
  - a family of random variables { X(k) | k=0,1,2,… }
  - X(k) are observations at discrete time-steps
  - i.e. X(k) is the state of the system at time-step k

- 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
  - $P(s_{k-1},s_k) = Pr(X(k)=s_k \mid X(k-1)=s_{k-1})$

Paths and probabilities

- A (finite or infinite) path through a DTMC
  - is a sequence of states $s_0s_1s_2s_3$… such that $P(s_i,s_{i+1}) > 0\ \forall i$
  - represents an execution (i.e. one possible behaviour) of the system which the DTMC is modelling
- To reason (quantitatively) about this system
  - need to define a probability space over paths
- Intuitively:
  - sample space: Path(s) = set of all infinite paths from a state s
  - events: sets of infinite paths from s
  - basic events: cylinder sets (or "cones")
  - cylinder set C(ω), for a finite path ω = set of infinite paths with the common finite prefix ω
  - for example: $C(ss_1s_2)$



Probability spaces

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

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

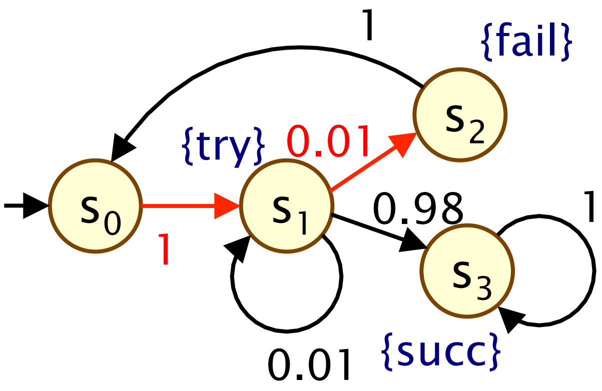
Probability space over paths

- Sample space Ω = Path(s)
  set of infinite paths with initial state s
- Event set $\Sigma_{Path(s)}$
  - the cylinder set $C(\omega) = \{ \omega' \in Path(s) \mid \omega$ 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 $\mathbf{P}_s(\omega)$ for finite path $\omega = ss_1 \ldots s_n$ as:
    - $\mathbf{P}_s(\omega) = 1$ if ω has length one (i.e. ω = s)
    - $\mathbf{P}_s(\omega) = \mathbf{P}(s,s_1) \cdot \ldots \cdot \mathbf{P}(s_{n-1},s_n)$ otherwise
    - define $Pr_s(C(\omega)) = \mathbf{P}_s(\omega)$ for all finite paths ω
  - Pr_s extends uniquely to a probability measure $Pr_s : \Sigma_{Path(s)} \rightarrow [0,1]$

- See [KSK76] for further details

Probability space – Example

- Paths where sending fails the first time
  - $\omega = s_0 s_1 s_2$
  - $C(\omega)$ = all paths starting $s_0 s_1 s_2 \ldots$
  - $\mathbf{P}_{s0}(\omega) = \mathbf{P}(s_0,s_1) \cdot \mathbf{P}(s_1,s_2)$
    $= 1 \cdot 0.01 = 0.01$
  - $Pr_{s0}(C(\omega)) = \mathbf{P}_{s0}(\omega) = 0.01$



- Paths which are eventually successful and with no failures
  - $C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots$
  - $Pr_{s0}(\, C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots\,)$

    $= \mathbf{P}_{s0}(s_0 s_1 s_3) + \mathbf{P}_{s0}(s_0 s_1 s_1 s_3) + \mathbf{P}_{s0}(s_0 s_1 s_1 s_1 s_3) + \ldots$

    $= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots$

    $= 0.9898989898\ldots$

    $= 98/99$

Overview (Part 2)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- Other properties: LTL, costs and rewards
- Case study: Bluetooth device discovery

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

  – ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $P_{\sim p}$ [ψ]   (state formulas)

  – ψ ::= X ϕ | $\phi\ U^{\leq k}\ \phi$ | $\phi\ U\ \phi$ (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, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

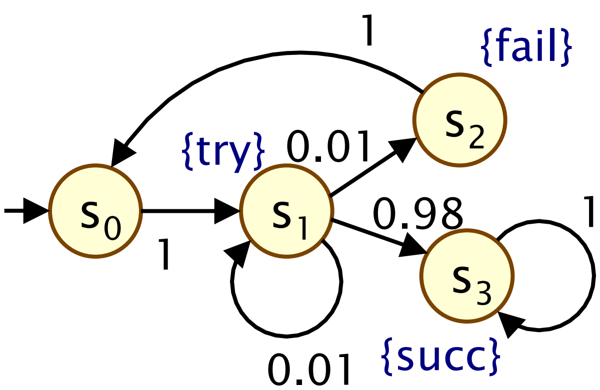
- A PCTL formula is always a state formula
  – path formulas only occur inside the P operator

PCTL semantics for DTMCs

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

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

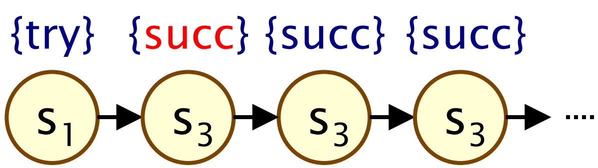
- Examples
  - $s_3 \models succ$
  - $s_1 \models try \wedge \neg fail$



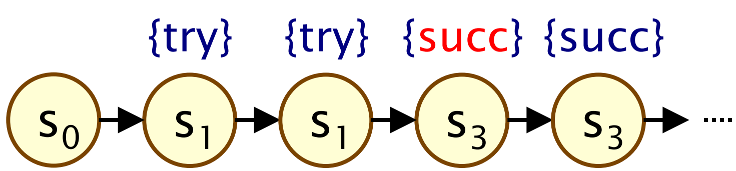
PCTL semantics for DTMCs

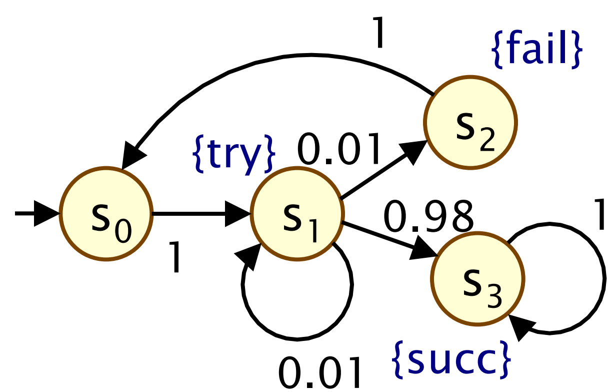
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \vDash X\ \phi \quad \Leftrightarrow \quad s_1 \vDash \phi$
  - $\omega \vDash \phi_1\ U^{\leq k}\ \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \vDash \phi_2$ and $\forall j<i,\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

- Some examples of satisfying paths:
  - X succ



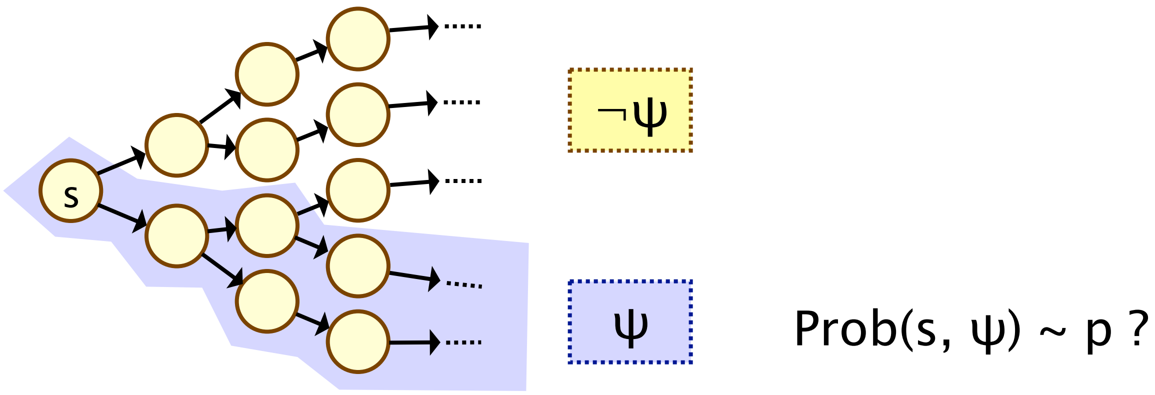
  - ¬fail U succ





PCTL semantics for DTMCs

- Semantics of the probabilistic operator P
  - informal definition: $s \vDash P_{\sim p}$ [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])



More PCTL…

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

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

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

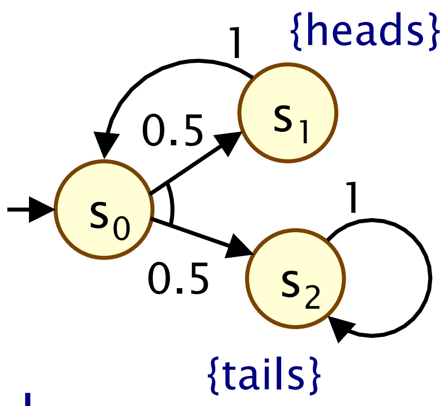
Qualitative vs. quantitative properties

- 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

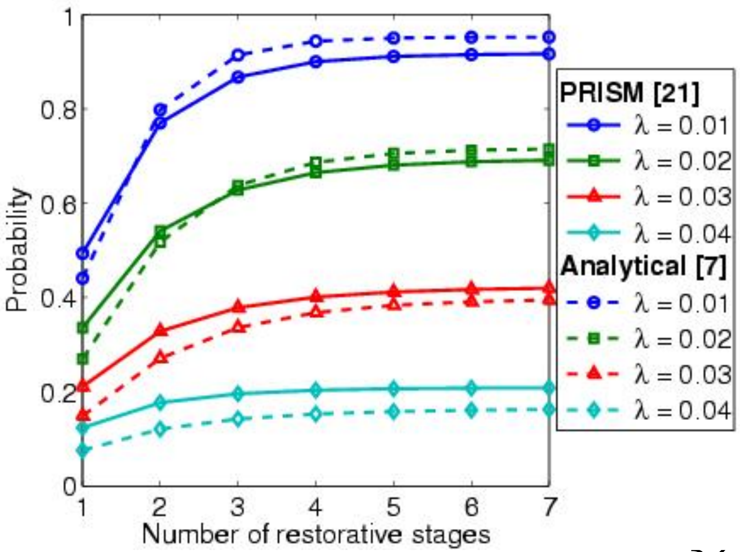
- $P_{\geq 1}$ [F ϕ] is (similar to but) weaker than AF ϕ
  - e.g. AF "tails" (CTL) ≠ $P_{\geq 1}$ [F "tails"] (PCTL)



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

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

reliability

- 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?"

performance

- 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?"

fairness

Overview (Part 2)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- Other properties: LTL, costs and rewards
- Case study: Bluetooth device discovery

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

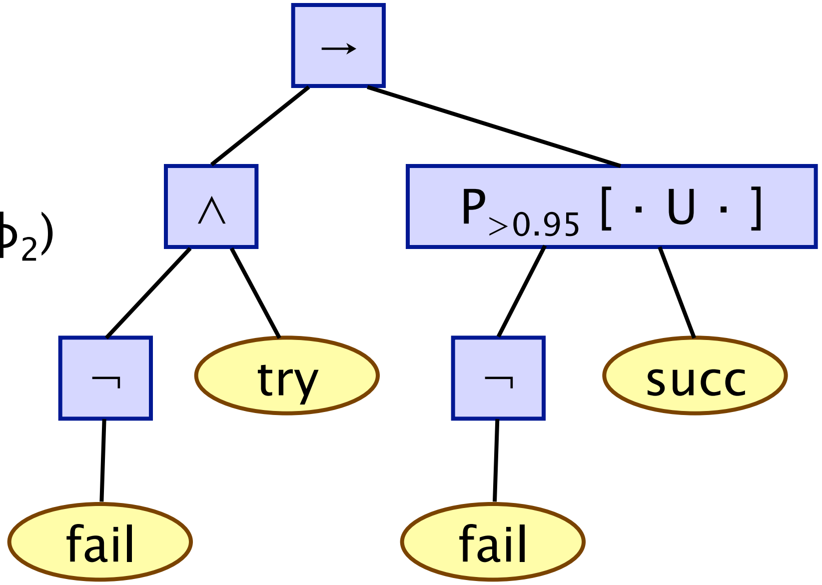
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - sometimes, want to check that $s \vDash \phi \ \forall \ s \in S$, i.e. $Sat(\phi) = S$
  - sometimes, just want to know if $s_{init} \vDash \phi$, i.e. if $s_{init} \in Sat(\phi)$

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

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



- 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: $\psi = \phi_1\ U\ \phi_2$

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($P_{\geq 1}$ [ϕ_1 U ϕ_2])
  - S^{no} = Sat($P_{\leq 0}$ [ϕ_1 U ϕ_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:

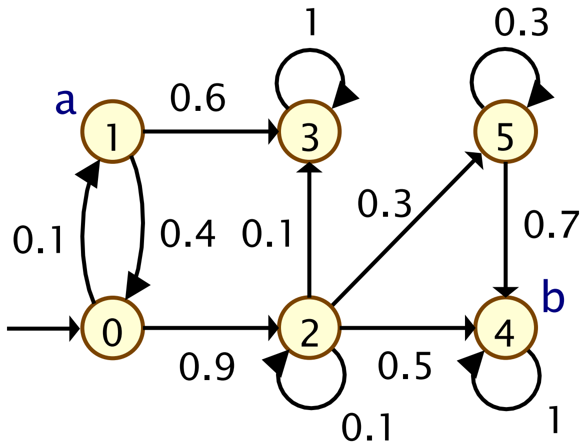
$$\text{Prob}(s, \phi_1 \text{ U } \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s,s') \cdot \text{Prob}(s', \phi_1 \text{ U } \phi_2) & \text{otherwise} \end{cases}$$

  – 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

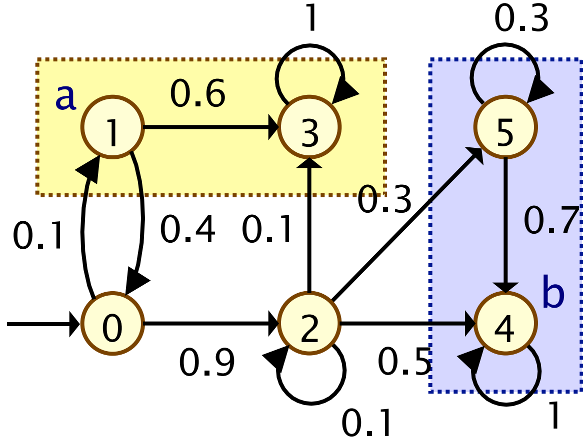
- Example: $P_{>0.8}$ [¬a U b]



PCTL until – Example

- Example: $P_{>0.8}$ [¬a U b]

S^{no} = Sat($P_{\leq 0}$ [¬a U b])

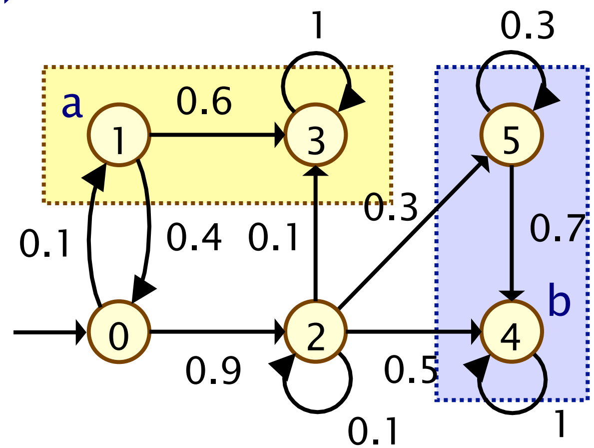


S^{yes} = Sat($P_{\geq 1}$ [¬a U b])

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])

$x_4 = x_5 = 1$

$x_1 = x_3 = 0$

$x_0 = 0.1x_1+0.9x_2 = 0.8$

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

Prob(¬a U b) = $\underline{x}$ = [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|

Overview (Part 2)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- Other properties: LTL, costs and rewards
- Case study: Bluetooth device discovery

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: $F\, \phi \equiv \text{true}\ U\, \phi$, $G\, \phi \equiv \neg(F\, \neg\phi)$
  - evaluated over paths of a model
- Examples
  - $(F\ tmp\_fail_1) \wedge (F\ tmp\_fail_2)$
  - "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"
  - $G\ (req \rightarrow X\ ack)$
  - "requests are always immediately acknowledged"

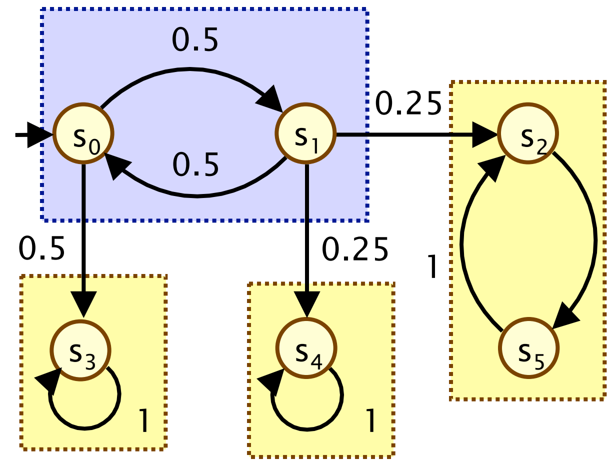
LTL for DTMCs

- Same idea as PCTL: probabilities of sets of path formulae
  - for a state s of a DTMC and an LTL formula ψ:
  - $\text{Prob}(s, \psi) = \text{Pr}_s \{ \omega \in \text{Path}(s) \mid \omega \vDash \psi \}$
  - 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. $P_{>0.5} [\text{ GF crit}_1] \wedge P_{>0.5} [\text{ GF crit}_2]$

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”



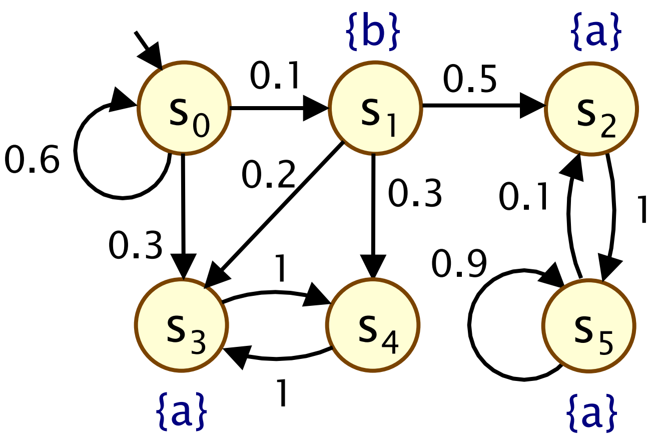
- Formally:
  - $Pr_s \{ \omega \in Path(s) \mid \exists\ i \geq 0,\ \exists$ BSCC T such that
    $\forall\ j \geq i\ \omega(i) \in T$ and
    $\forall\ s' \in T\ \omega(k) = s'$ for infinitely many k $\} = 1$

LTL model checking for DTMCs

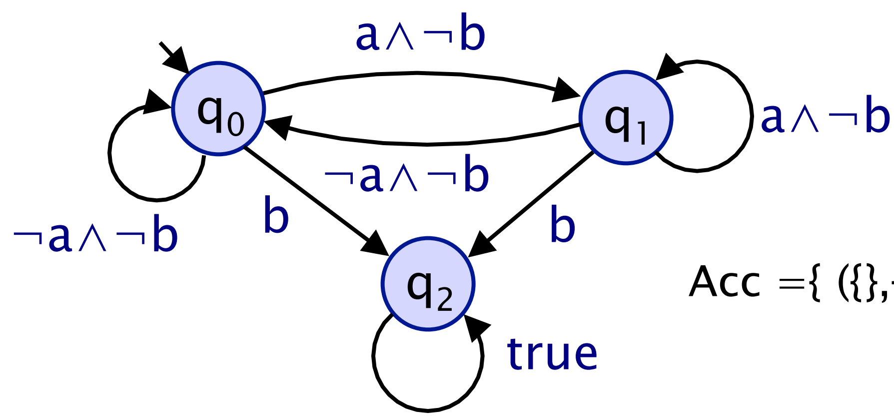
- Steps for model checking LTL property ψ on DTMC D
  - i.e. computing $Prob^D(s, \psi)$
- 1. Build a deterministic Rabin automaton (DRA) A for ψ
  - i.e. a DRA A over alphabet 2^{AP} accepting ψ-satisfying traces
- 2. Build the "product" DTMC $D \otimes A$
  - records state of A for path through D so far
- 3. Identify states T_{acc} in "accepting" BSCCs of $D \otimes A$
  - i.e. those that meet the acceptance condition of A
- 4. Compute probability of reaching T_{acc} in $D \otimes A$
  - which gives $Prob^D(s, \psi)$, as required

Example: LTL for DTMCs

DTMC D

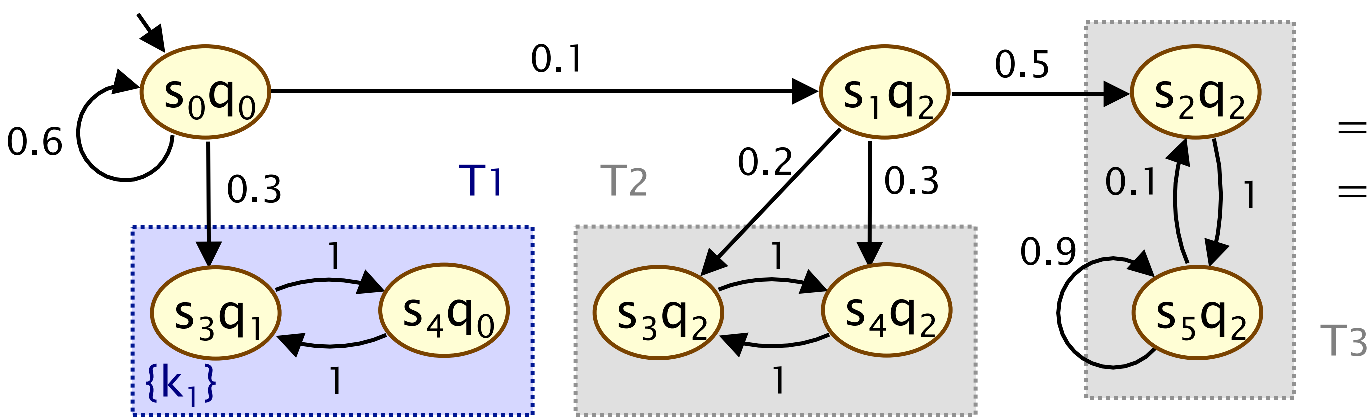


DRA A_ψ for $\psi = G\neg b \wedge GF\ a$



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

Product DTMC $D \otimes A_\psi$



$Prob^D(s, \psi)$

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

$= 3/4.$

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 $(\underline{\rho}, \boldsymbol{\iota})$
  - $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ is the state reward function (vector)
  - $\boldsymbol{\iota} : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition reward function (matrix)

- Example (for use with instantaneous properties)
  - "size of message queue": $\underline{\rho}$ maps each state to the number of jobs in the queue in that state, $\boldsymbol{\iota}$ 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 $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

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

Types of reward formulas

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

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

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

Reward formula semantics

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

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

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

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

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]

Overview (Part 2)

- Discrete-time Markov chains (DTMCs)
- PCTL: A temporal logic for DTMCs
- PCTL model checking
- Other properties: LTL, costs and rewards
- Case study: Bluetooth device discovery

The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL), runs on all major OSs
- Support for:
  - discrete-/continuous-time Markov chains (D/CTMCs)
  - Markov decision processes (MDPs)
  - probabilistic timed automata (PTAs)
  - PCTL, CSL, LTL, PCTL*, costs/rewards, …
- Multiple efficient model checking engines
  - mostly symbolic (BDDs) (up to 10^{10} states, 10^7–10^8 on avg.)
- Successfully applied to a wide range of case studies
  - communication protocols, security protocols, dynamic power management, cell signalling pathways, …
- See: http://www.prismmodelchecker.org/

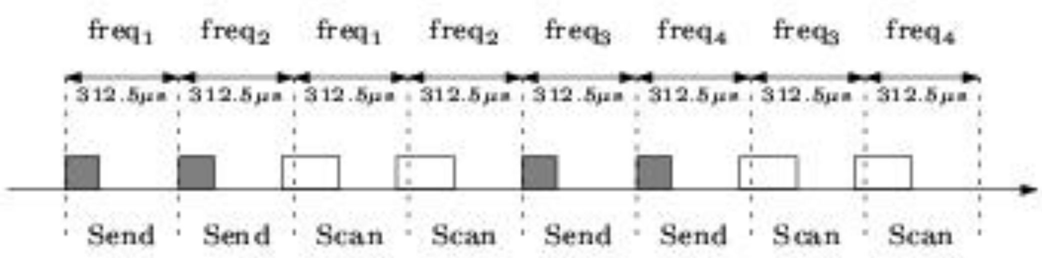
Bluetooth device discovery

- Bluetooth: short-range low-power wireless protocol
  - widely available in phones, PDAs, laptops, ...
  - open standard, specification freely available
- Uses frequency hopping scheme
  - to avoid interference (uses unregulated 2.4GHz band)
  - pseudo-random selection over 32 of 79 frequencies
- Formation of personal area networks (PANs)
  - piconets (1 master, up to 7 slaves)
  - self-configuring: devices discover themselves
- Device discovery
  - mandatory first step before any communication possible
  - relatively high power consumption so performance is crucial
  - master looks for devices, slaves listens for master

Master (sender) behaviour

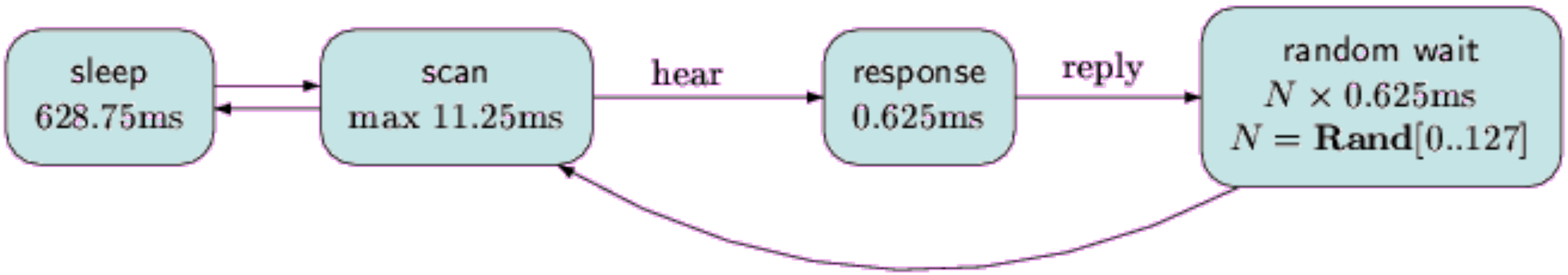
- 28 bit free-running clock CLK, ticks every 312.5µs
- Frequency hopping sequence determined by clock:
  - freq = $[CLK_{16-12}+k+(CLK_{4-2,0}-CLK_{16-12}) \bmod 16] \bmod 32$
  - 2 trains of 16 frequencies (determined by offset k), 128 times each, swap between every 2.56s
- Broadcasts “inquiry packets”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

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



Slave (receiver) behaviour

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



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

Bluetooth – PRISM model

- Modelled/analysed using PRISM model checker [DKNP06]
  - model scenario with one sender and one receiver
  - synchronous (clock speed defined by Bluetooth spec)
  - model at lowest-level (one clock-tick = one transition)
  - randomised behaviour so model as a DTMC
  - use real values for delays, etc. from Bluetooth spec

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

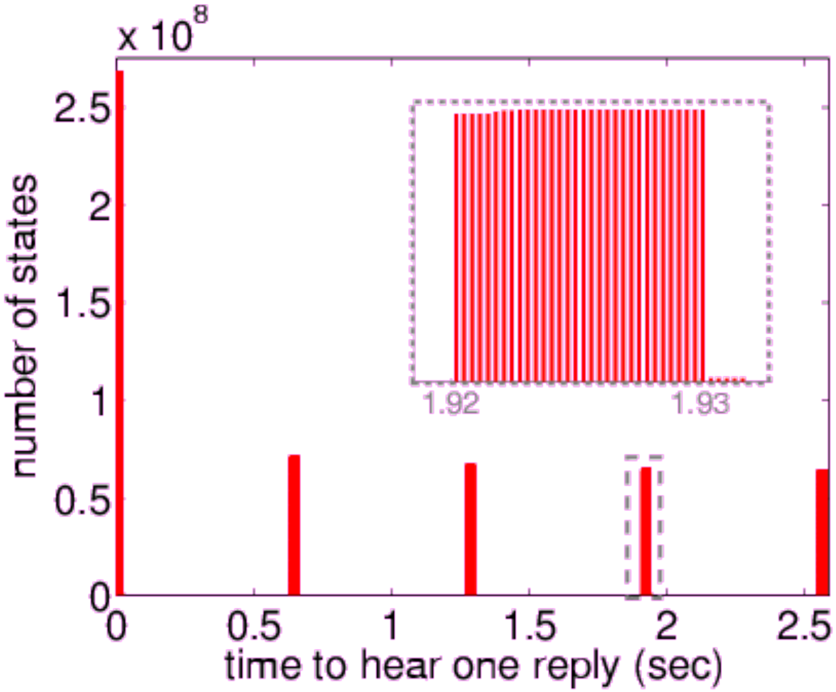
Bluetooth – Results

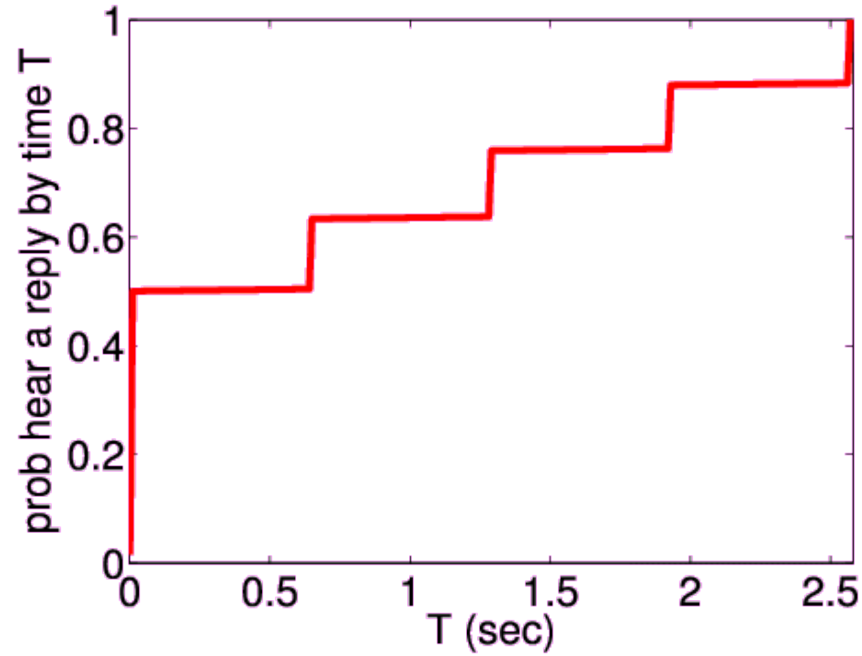
- Huge DTMC – initially, model checking infeasible
  - partition into 32 scenarios, i.e. 32 separate DTMCs
  - on average, approx. 3.4×10^9 states (536,870,912 initial)
  - can be built/analysed with PRISM's MTBDD engine

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

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

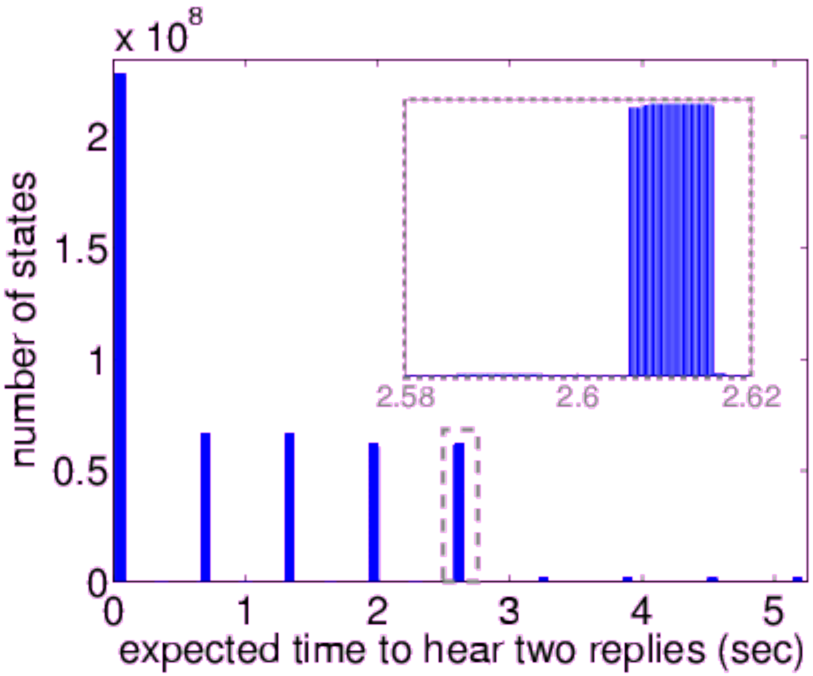
Bluetooth – Time to hear 1 reply

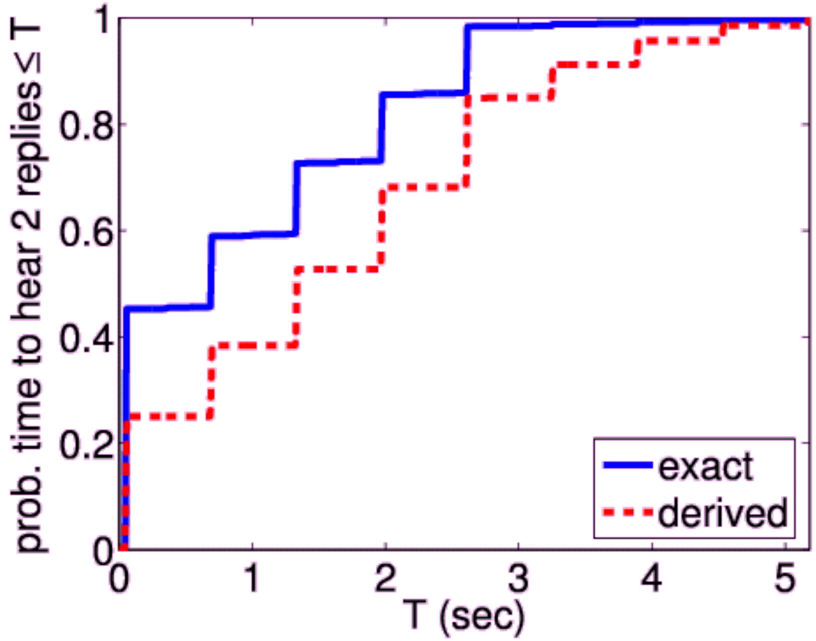




- Worst-case expected time = 2.5716 sec
  - in 921,600 possible initial states
  - best-case = 635 μs

Bluetooth – Time to hear 2 replies





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

Bluetooth – Results

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

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

Summary (Parts 1 & 2)

- 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

- Next: Markov decision processes (MDPs)

Part 3

Markov decision processes

Overview

- Lectures 1 and 2:
  - 1 – Introduction
  - 2 – Discrete-time Markov chains
  - 3 – Markov decision processes
  - 4 – Compositional probabilistic verification
- Course materials available here:
  - http://www.prismmodelchecker.org/courses/sfm11connect/
  - lecture slides, reference list, tutorial chapter, lab session

Probabilistic models

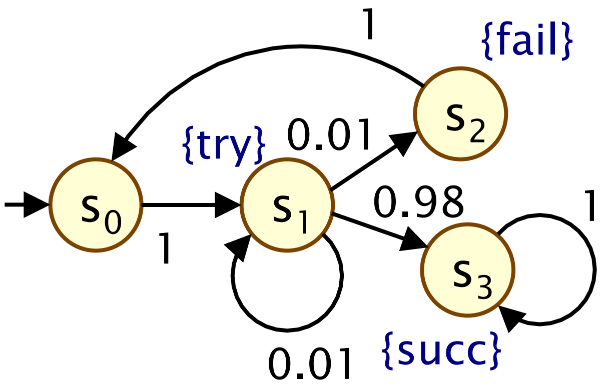
	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) (probabilistic automata)
Continuous time	Continuous-time Markov chains (CTMCs)	Probabilistic timed automata (PTAs)
		CTMDPs / IMCs

Overview (Part 3)

- Markov decision processes (MDPs)
- Adversaries & probability spaces
- Properties of MDPs: The temporal logic PCTL
- PCTL model checking for MDPs
- Case study: Firewire root contention

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
  - $P : S \times S \to [0,1]$ is the transition probability matrix
  - $L : S \to 2^{AP}$ 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 $\to P_{\geq 0.95}$ [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}

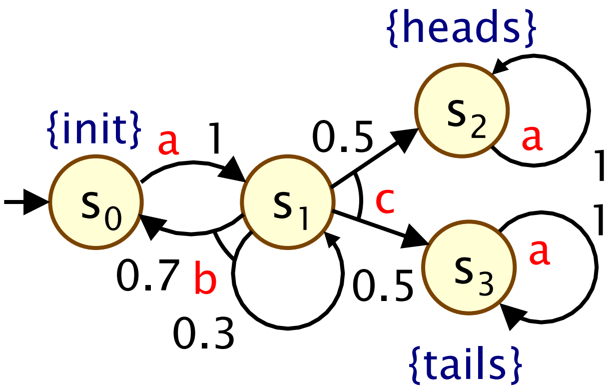
- Unknown environments
  - e.g. probabilistic security protocols – unknown adversary

Markov decision processes

- Markov decision processes (MDPs)
  - extension of DTMCs which allow nondeterministic choice

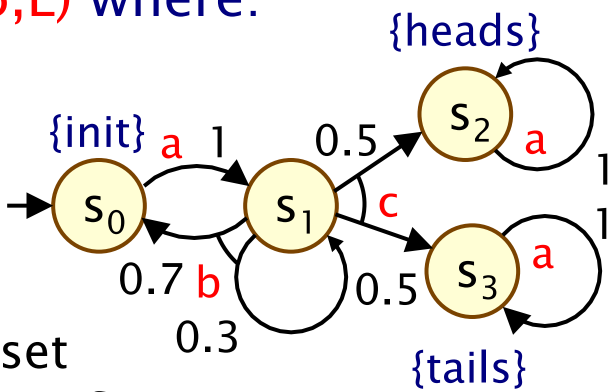
- Like DTMCs:
  - discrete set of states representing possible configurations of the system being modelled
  - transitions between states occur in discrete time-steps

- Probabilities and nondeterminism
  - in each state, a nondeterministic choice between several discrete probability distributions over successor states



Markov decision processes

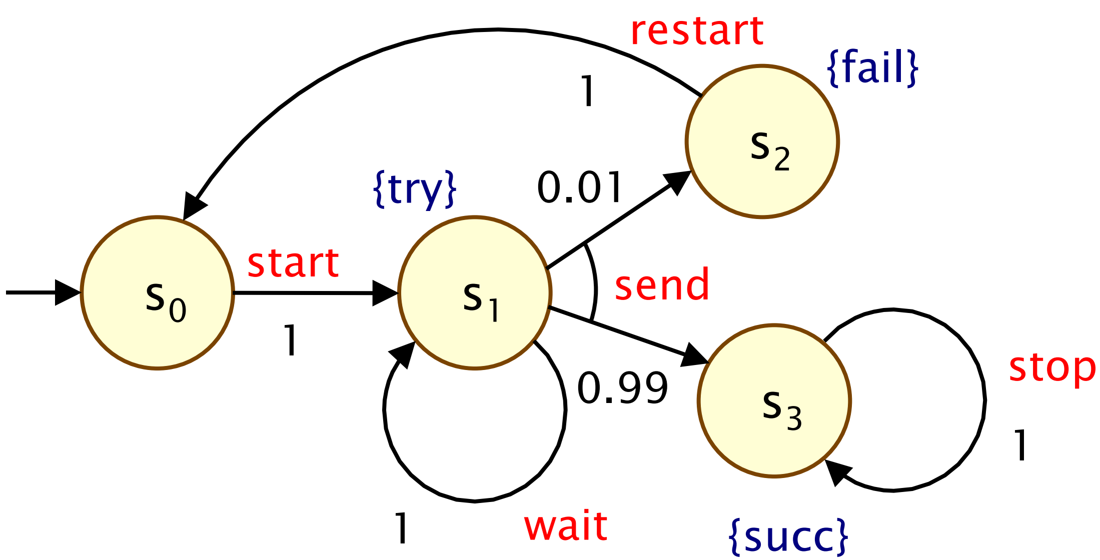
- Formally, an MDP M is a tuple $(S, s_{init}, \alpha, \delta, L)$ where:
  - S is a set of states ("state space")
  - $s_{init} \in S$ is the initial state
  - α is an alphabet of action labels
  - $\delta \subseteq S \times \alpha \times Dist(S)$ is the transition probability relation, where Dist(S)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



- Notes:
  - we also abuse notation and use δ as a function
  - i.e. $\delta : S \rightarrow 2^{\alpha \times Dist(S)}$ 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]

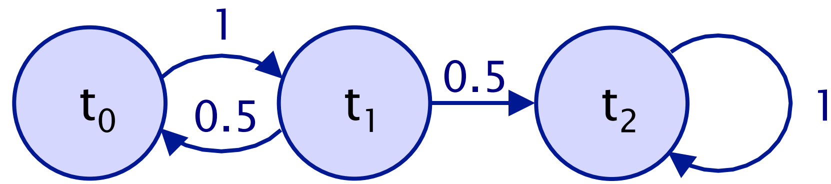
Simple MDP example

- 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

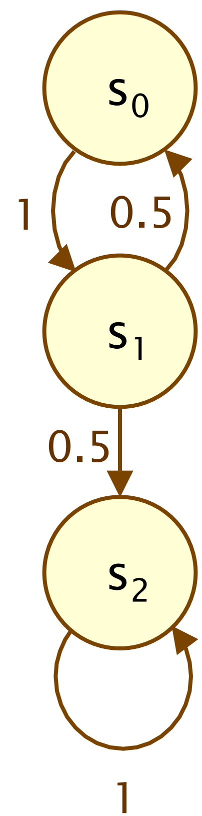


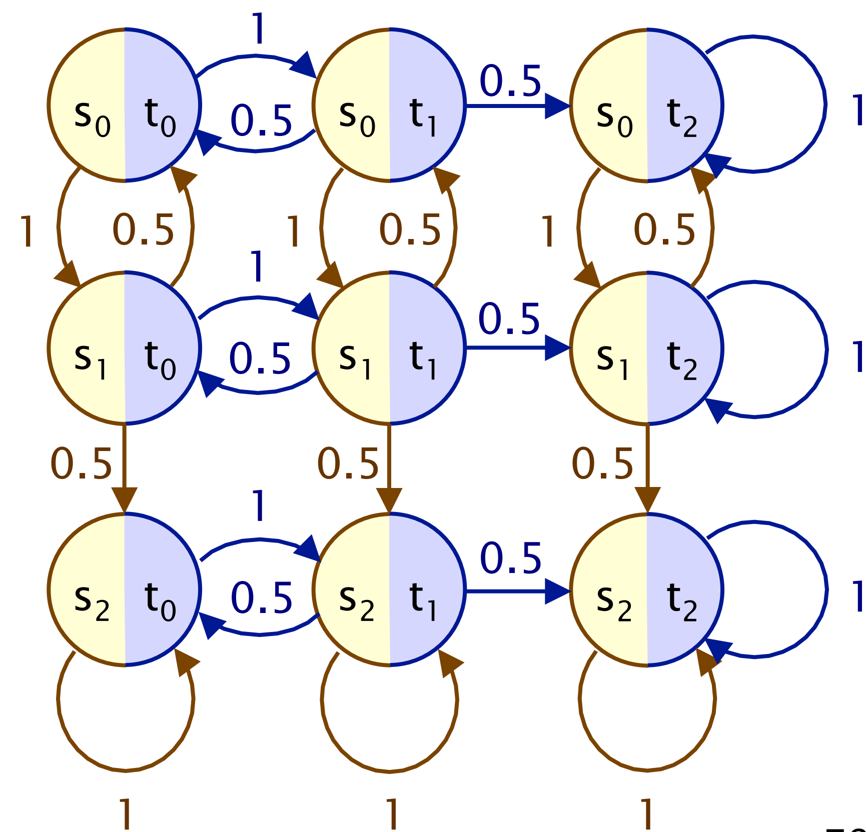
Example – Parallel composition

Asynchronous parallel composition of two 3-state DTMCs



Action labels omitted here





Paths and probabilities

- A (finite or infinite) path through an MDP M
  - is a sequence of states and action/distribution pairs
  - e.g. $s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$
  - such that $(a_i,\mu_i) \in \delta(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geq 0$
  - represents an execution (i.e. one possible behaviour) of the system which the MDP is modelling
  - note that a path resolves both types of choices: nondeterministic and probabilistic
  - $Path_{M,s}$ (or just $Path_s$) is the set of all infinite paths starting from state s in MDP M; the set of finite paths is $PathFin_s$

- To consider the probability of some behaviour of the MDP
  - first need to resolve the nondeterministic choices
  - …which results in a DTMC
  - …for which we can define a probability measure over paths

Overview (Part 3)

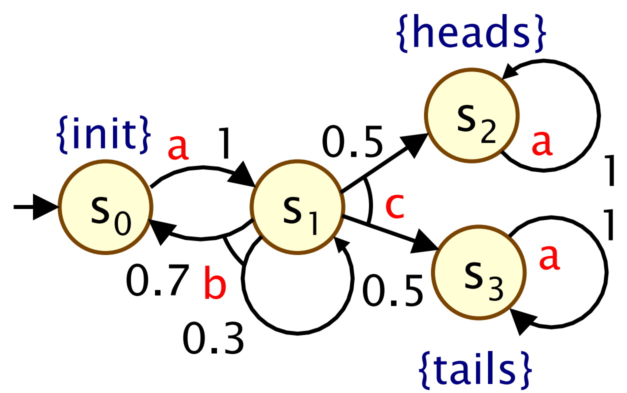
- Markov decision processes (MDPs)
- Adversaries & probability spaces
- Properties of MDPs: The temporal logic PCTL
- PCTL model checking for MDPs
- Case study: Firewire root contention

Adversaries

- An adversary resolves nondeterministic choice in an MDP
  - also known as “schedulers”, “strategies” or “policies”
- Formally:
  - an adversary σ of an MDP is a function mapping every finite path $\omega = s_0(a_0,\mu_0)s_1 \ldots s_n$ to an element of $\delta(s_n)$
- Adversary σ restricts the MDP to certain paths
  - $Path_s^\sigma \subseteq Path_s^\sigma$ and $PathFin_s^\sigma \subseteq PathFin_s^\sigma$
- Adversary σ induces a probability measure Pr_s^σ over paths
  - constructed through an infinite state DTMC $(PathFin_s^\sigma, s, \mathbf{P}_s^\sigma)$
  - states of the DTMC are the finite paths of σ starting in state s
  - initial state is s (the path starting in s of length 0)
  - $\mathbf{P}_s^\sigma(\omega,\omega')=\mu(s)$ if $\omega'= \omega(a,\mu)s$ and $\sigma(\omega)=(a,\mu)$
  - $\mathbf{P}_s^\sigma(\omega,\omega')=0$ otherwise

Adversaries – Examples

- Consider the simple MDP below
  - note that s_1 is the only state for which $|\delta(s)| > 1$
  - i.e. s_1 is the only state for which an adversary makes a choice
  - let μ_b and μ_c denote the probability distributions associated with actions b and c in state s_1

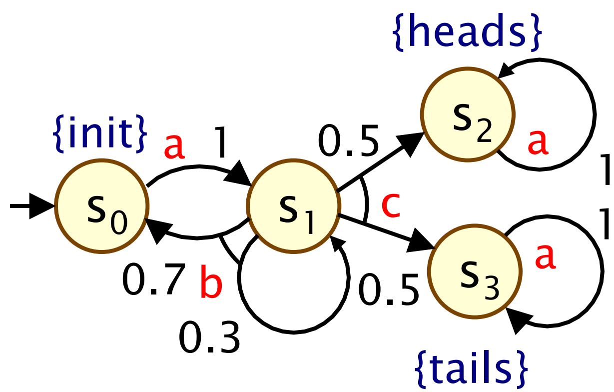


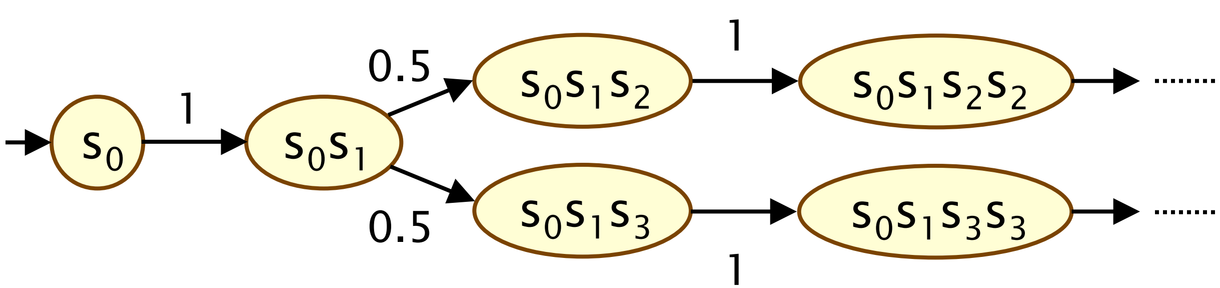
- Adversary σ_1
  - picks action c the first time
  - $\sigma_1(s_0 s_1)=(c,\mu_c)$

- Adversary σ_2
  - picks action b the first time, then c
  - $\sigma_2(s_0 s_1)=(b,\mu_b)$, $\sigma_2(s_0 s_1 s_1)=(c,\mu_c)$, $\sigma_2(s_0 s_1 s_0 s_1)=(c,\mu_c)$

Adversaries – Examples

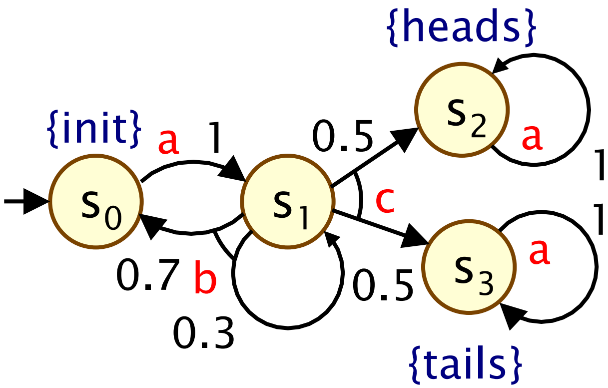
- Fragment of DTMC for adversary σ_1
  - σ_1 picks action c the first time

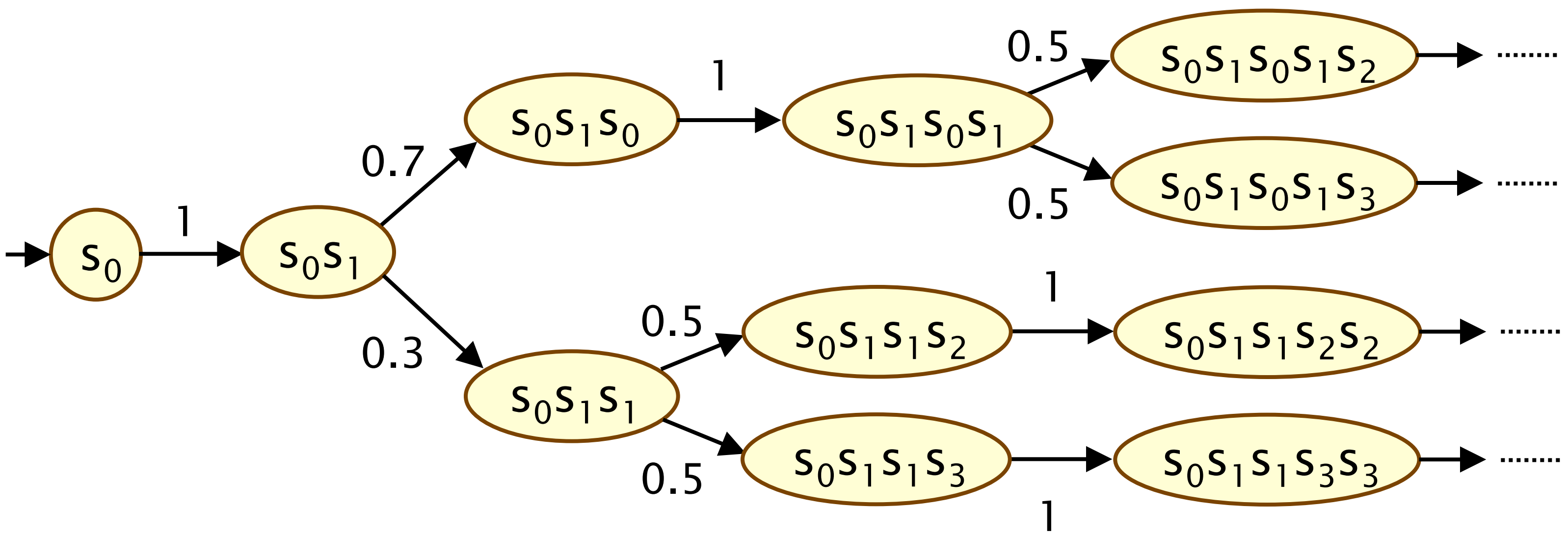




Adversaries – Examples

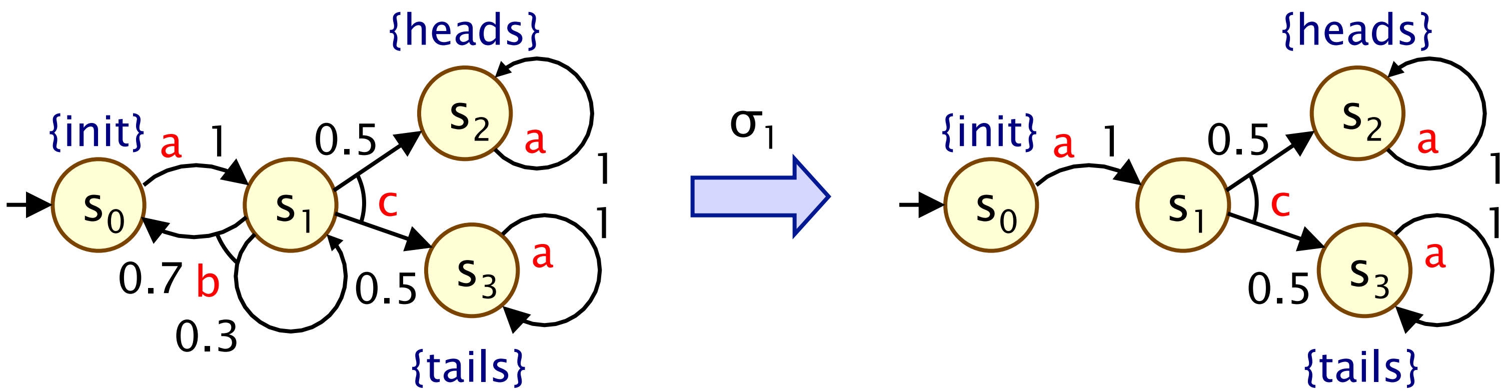
- Fragment of DTMC for adversary σ_2
  - σ_2 picks action b, then c





Memoryless adversaries

- Memoryless adversaries always pick same choice in a state
  - also known as: positional, simple, Markov
  - formally, for adversary σ:
  - $\sigma(s_0(a_0,\mu_0)s_1 \ldots s_n)$ depends only on s_n
  - resulting DTMC can be mapped to a $|S|$-state DTMC
- From previous example:
  - adversary σ_1 (picks c in s_1) is memoryless, σ_2 is not



Overview (Part 3)

- Markov decision processes (MDPs)
- Adversaries & probability spaces
- Properties of MDPs: The temporal logic PCTL
- PCTL model checking for MDPs
- Case study: Firewire root contention

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 $\rightarrow$ $P_{\geq 0.95}$ [true $U^{\leq 10}$ deliver]

PCTL semantics for MDPs

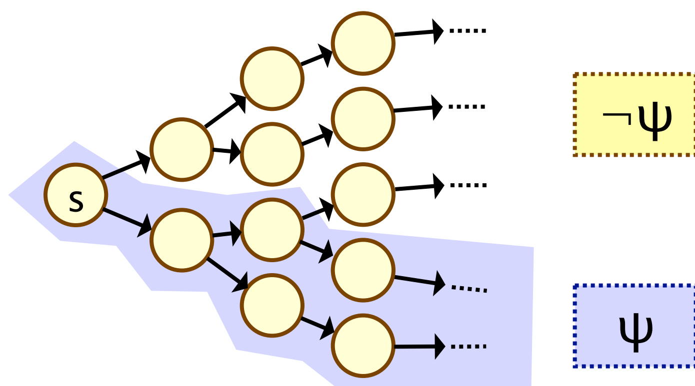
- PCTL formulas interpreted over states of an MDP
  - $s \vDash \phi$ denotes ϕ is "true in state s" or "satisfied in state s"

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

- Semantics of path formulas:
  - for a path $\omega = s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$ in the MDP:
  - $\omega \vDash X\,\phi \quad \Leftrightarrow \quad s_1 \vDash \phi$
  - $\omega \vDash \phi_1\ U^{\leq k}\ \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ 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 $\omega \vDash \phi_1\ U^{\leq k}\ \phi_2$

PCTL semantics for MDPs

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



$Pr_s^{\sigma}(\psi) \sim p$

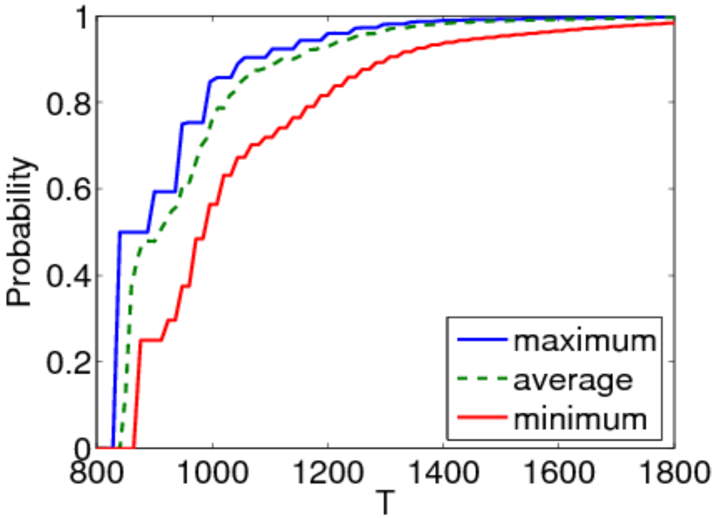
- Some equivalences:
  - $F\ \phi \equiv \lozenge\ \phi \equiv \text{true}\ U\ \phi$ (eventually, "future")
  - $G\ \phi \equiv \square\ \phi \equiv \neg(F\ \neg\phi)$ (always, "globally")

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 $s \vDash P_{\sim p} [\ \psi\ ] \ \Leftrightarrow\ Pr_s^{min}(\psi) \sim p$
  - if $\sim \in \{<,\leq\}$, then $s \vDash P_{\sim p} [\ \psi\ ] \ \Leftrightarrow\ Pr_s^{max}(\psi) \sim p$
- Model checking $P_{\sim p}[\ \psi\ ]$ reduces to the computation over all adversaries of either:
  - the minimum probability of ψ holding
  - the maximum probability of ψ holding
- Crucial result for model checking PCTL on MDPs
  - memoryless adversaries suffice, i.e. there are always memoryless adversaries 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"



Other classes of adversary

- A more general semantics for PCTL over MDPs
  - parameterise by a class of adversaries Adv
- Only change is:
  - $s \vDash_{Adv} P_{\sim p} [\psi] \;\Leftrightarrow\; Pr_s^{\sigma}(\psi) \sim p$ for all adversaries $\sigma \in Adv$
- Original semantics obtained by taking Adv to be the set of all adversaries for the MDP
- Alternatively, take Adv to be the set of all fair adversaries
  - path fairness: if a state is occurs on a path infinitely often, then each non-deterministic choice occurs infinite often
  - see e.g. [BK98]

Some real PCTL examples

- Byzantine agreement protocol
  - $P_{min=?}$ [F (agreement $\wedge$ rounds$\leq$2)]
  - "what is the minimum probability that agreement is reached within two rounds?"
- CSMA/CD communication protocol
  - $P_{max=?}$ [F collisions=k]
  - "what is the maximum probability of k collisions?"
- Self-stabilisation protocols
  - $P_{min=?}$ [$F^{\leq t}$ stable]
  - "what is the minimum probability of reaching a stable state within k steps?"

Overview (Part 3)

- Markov decision processes (MDPs)
- Adversaries & probability spaces
- Properties of MDPs: The temporal logic PCTL
- PCTL model checking for MDPs
- Case study: Firewire root contention

PCTL model checking for MDPs

- Algorithm for PCTL model checking [BdA95]
  - inputs: MDP $M=(S,s_{init},\alpha,\delta,L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = 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
  - next ($X \ \phi$) and bounded until ($\phi_1 \ U^{\leq k} \ \phi_2$) are straightforward extensions of the DTMC case

PCTL until for MDPs

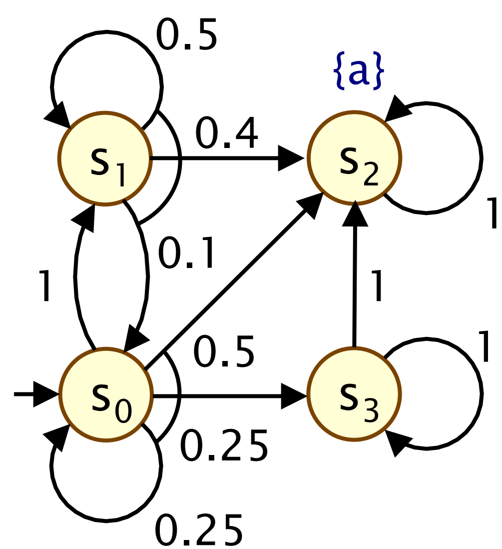
- Computation of probabilities $Pr_s^{min}(\phi_1 \text{ U } \phi_2)$ for all $s \in S$
- First identify all states where the probability is 1 or 0
  - "precomputation" algorithms, yielding sets S^{yes}, S^{no}
- Then compute (min) probabilities for remaining states ($S^?$)
  - either: solve linear programming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

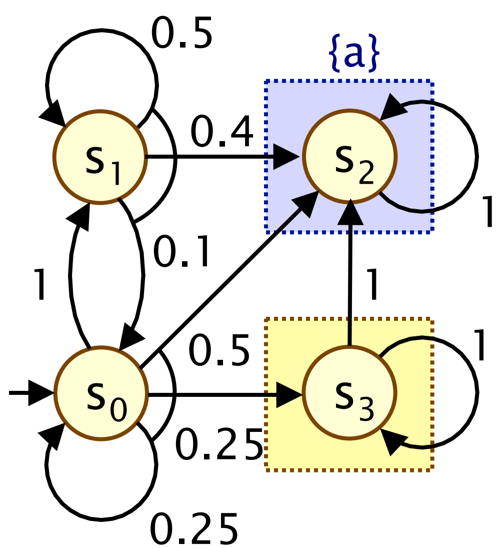


PCTL until – Precomputation

- 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 adversaries the probability of satisfying ϕ_1 U ϕ_2 is 1
  - algorithm Prob0E computes S^{no}
    - there exists an adversary 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:

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

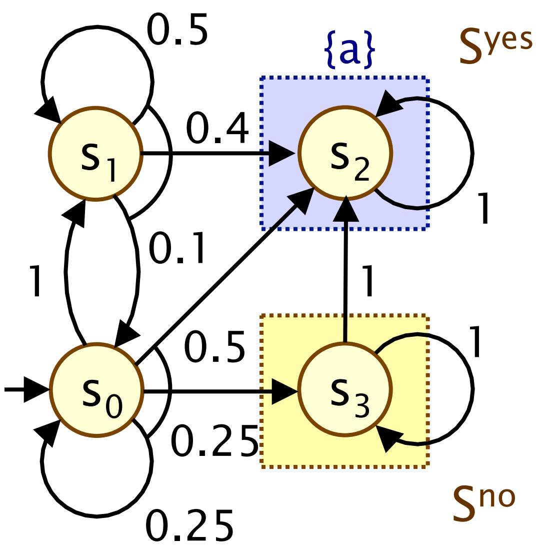
$$x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s')$$

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

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

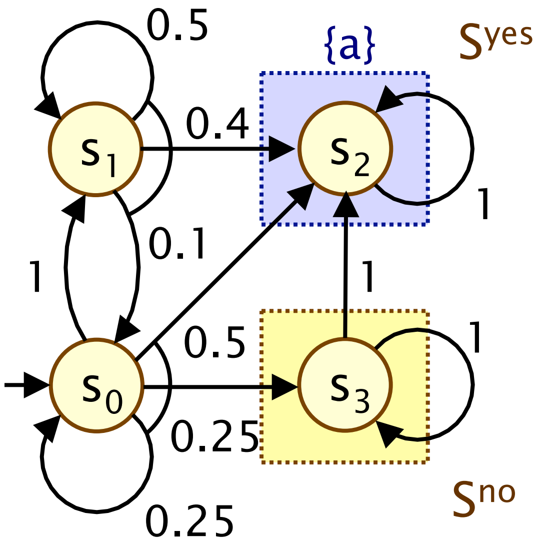
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

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)



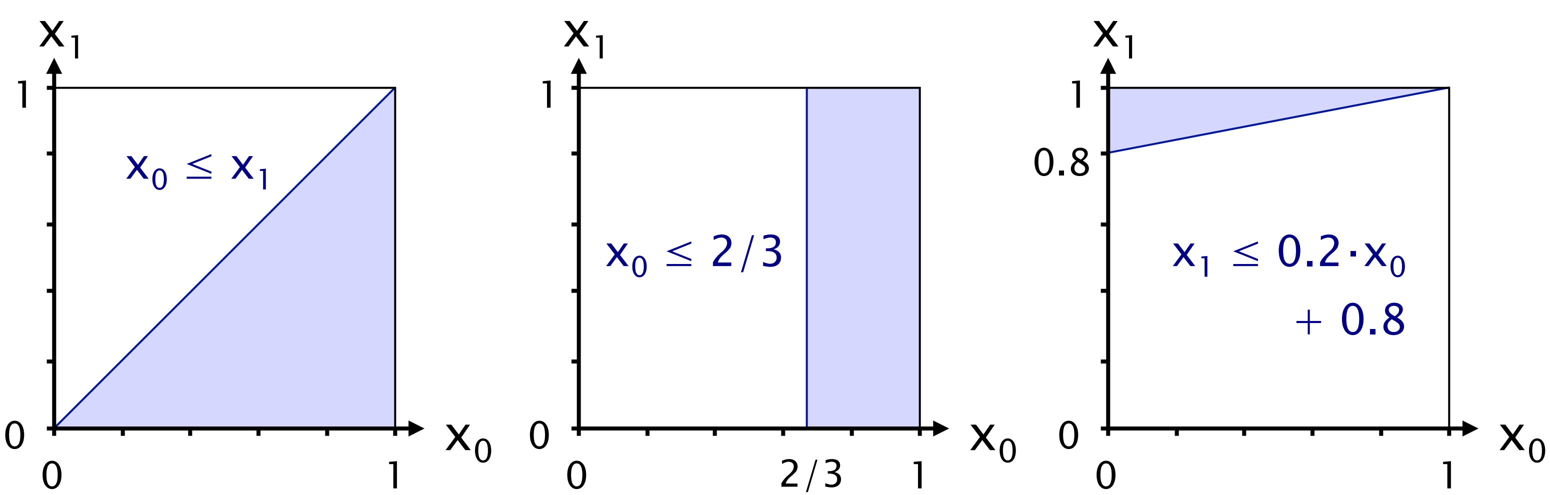
Let $x_i = Pr_{s_i}^{min}(F\ a)$

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

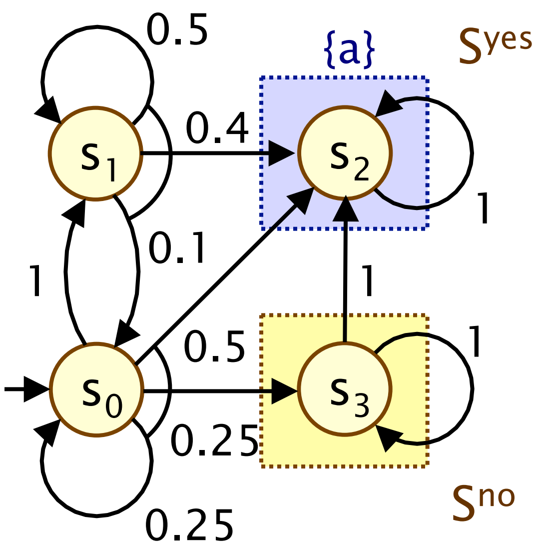
For $S^? = \{x_0, x_1\}$:

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)



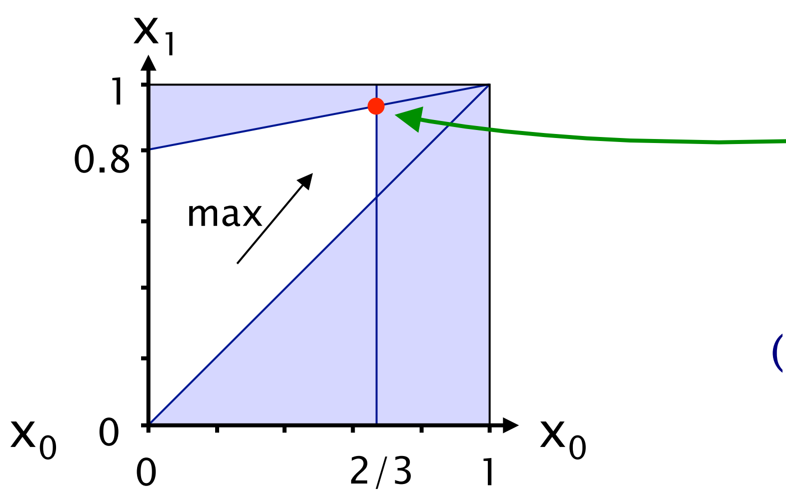
Let $x_i = Pr_{s_i}^{min}(F\ a)$

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

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

Maximise x_0+x_1 subject to constraints:

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



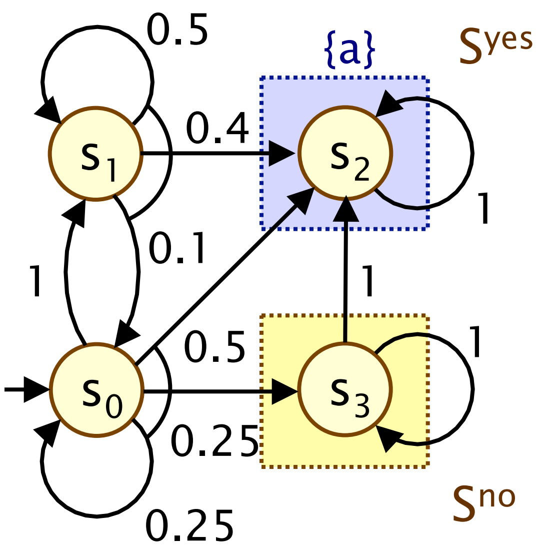
Solution:

(x_0, x_1)

$=$

$(2/3, 14/15)$

Example – PCTL until (LP)



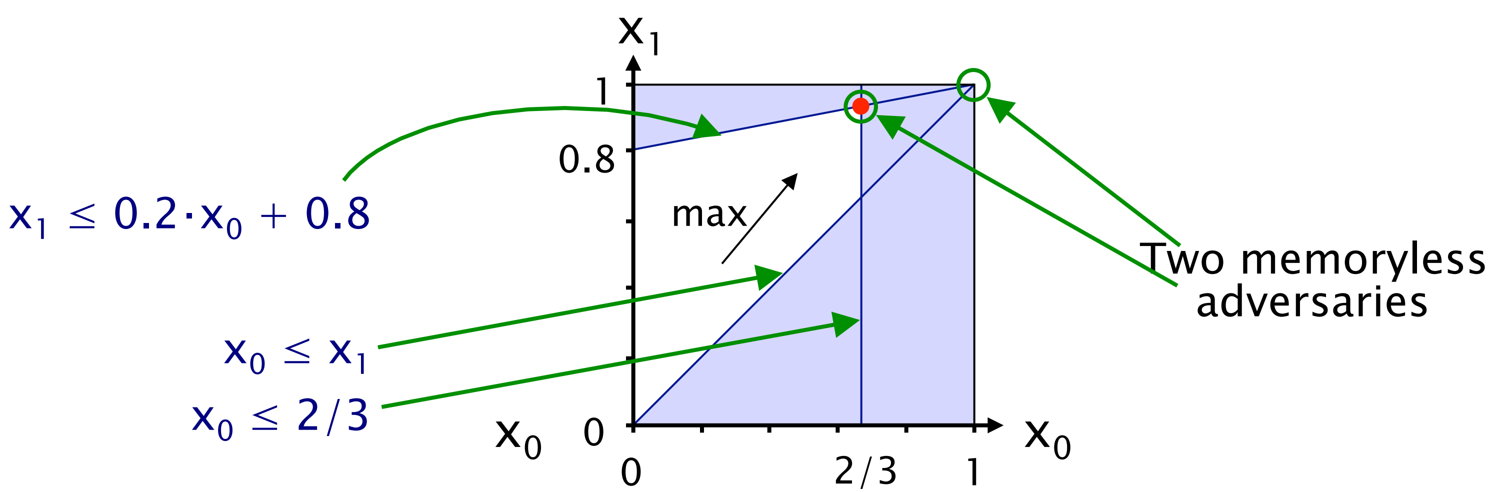
Let $x_i = Pr_{s_i}^{min}(F\ a)$

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

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

Maximise x_0+x_1 subject to constraints:

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



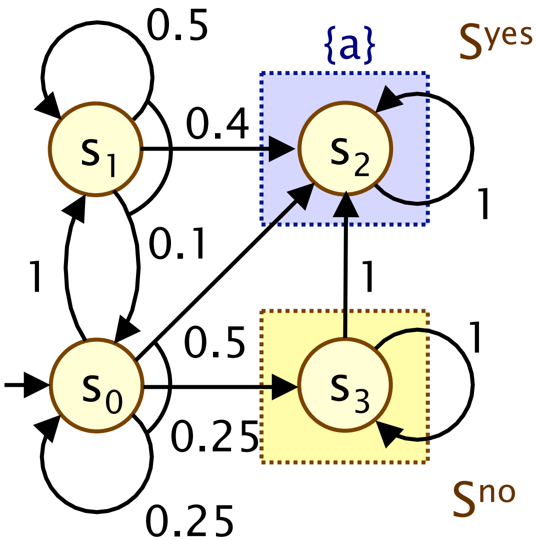
Method 2 – Value iteration

- 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(0,0.25·0+0.5),

0.1·0+0.5·0+0.4, 1, 0]

= [0, 0.4, 1, 0]

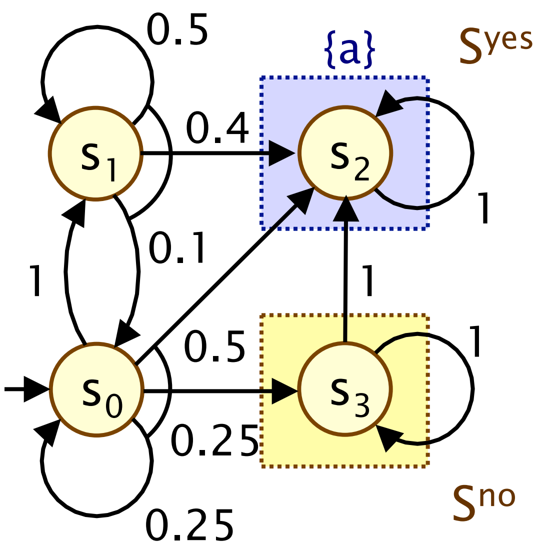
n=2: [min(0.4,0.25·0+0.5),

0.1·0+0.5·0.4+0.4, 1, 0]

= [0.4, 0.6, 1, 0]

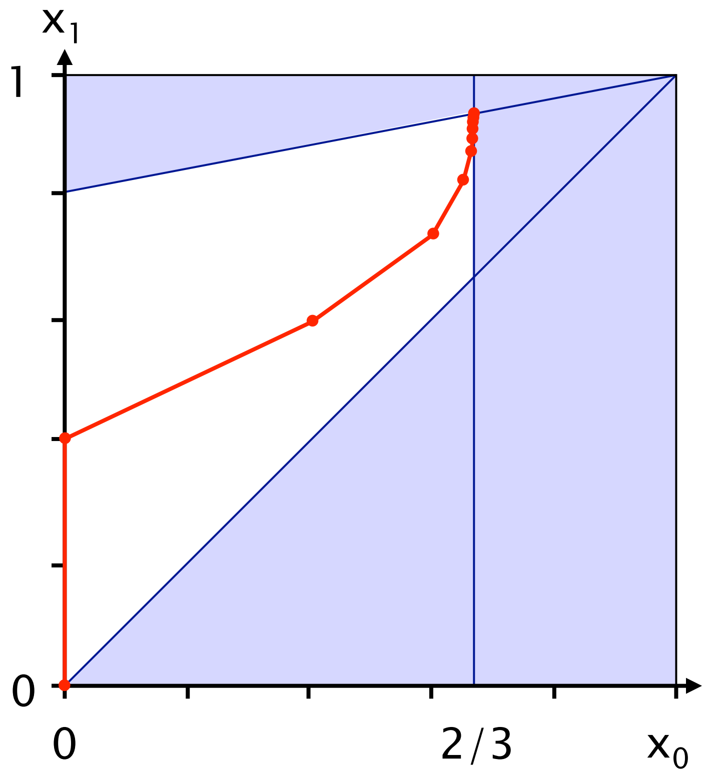
n=3: …

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 adversaries ("policies")

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

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

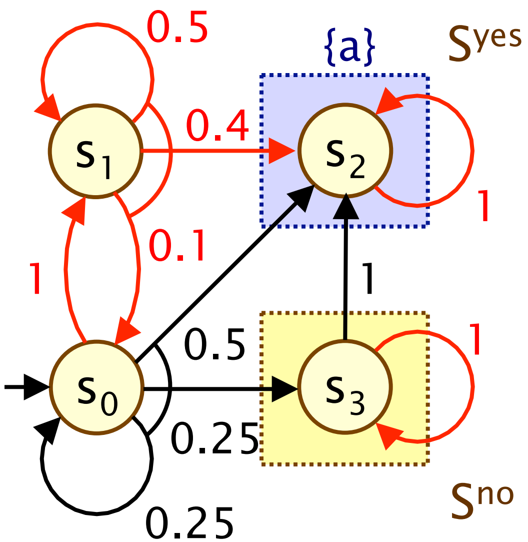
Method 3 – Policy iteration

- 1. Start with an arbitrary (memoryless) adversary 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 adversary in each state

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

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

Example – Policy iteration



Arbitrary adversary 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:

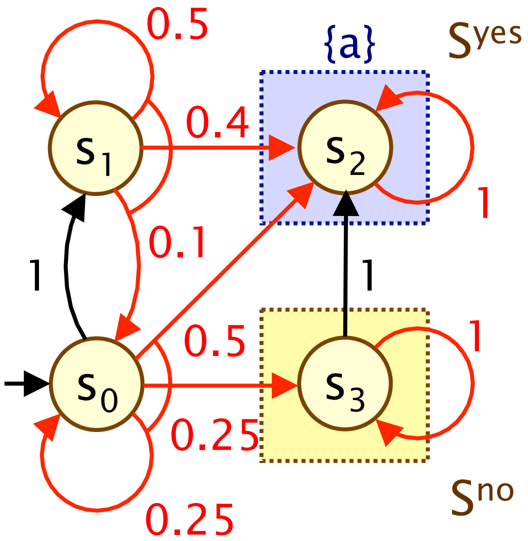
$\underline{Pr}^{\sigma}(F\ a) = [\ 1, 1, 1, 0\ ]$

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 adversary σ':

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

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

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

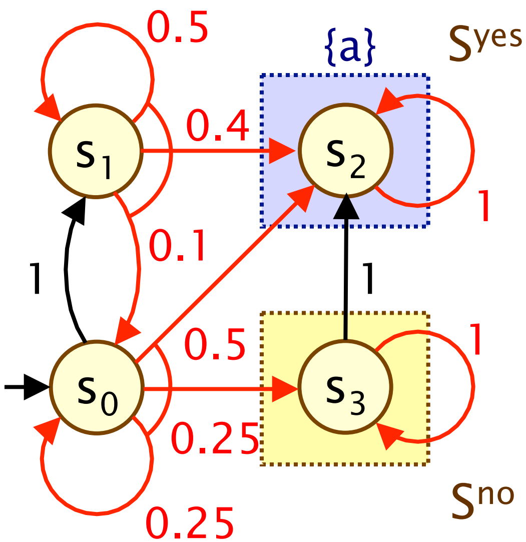
- $x_0 = 0.25 \cdot x_0 + 0.5$
- $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

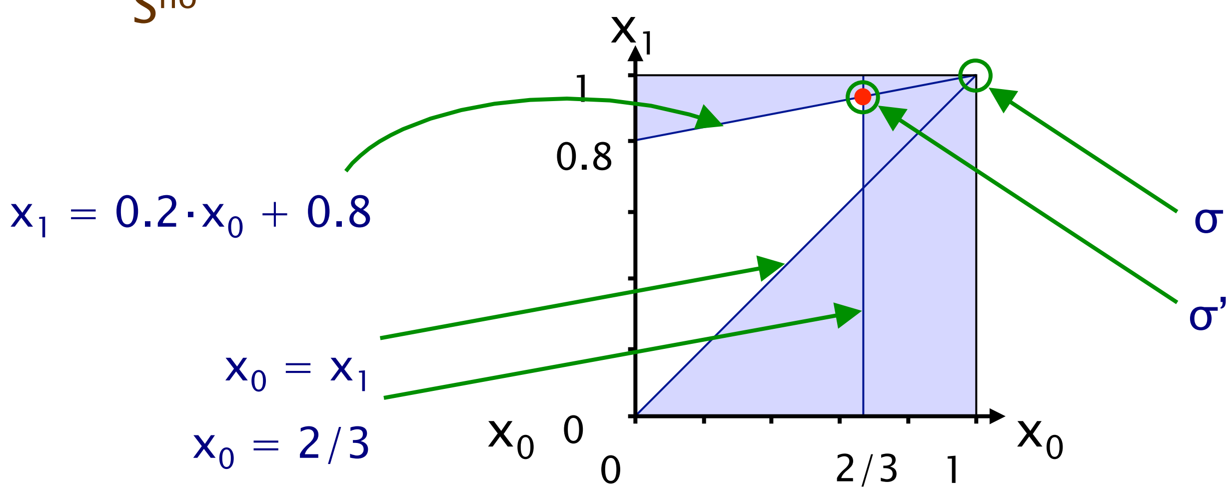
Solution:

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

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, $O(|S|^2)$
  - Φ_1 $U^{\leq k}$ Φ_2 : k matrix-vector multiplications, $O(k|S|^2)$
  - Φ_1 U Φ_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

Overview (Part 3)

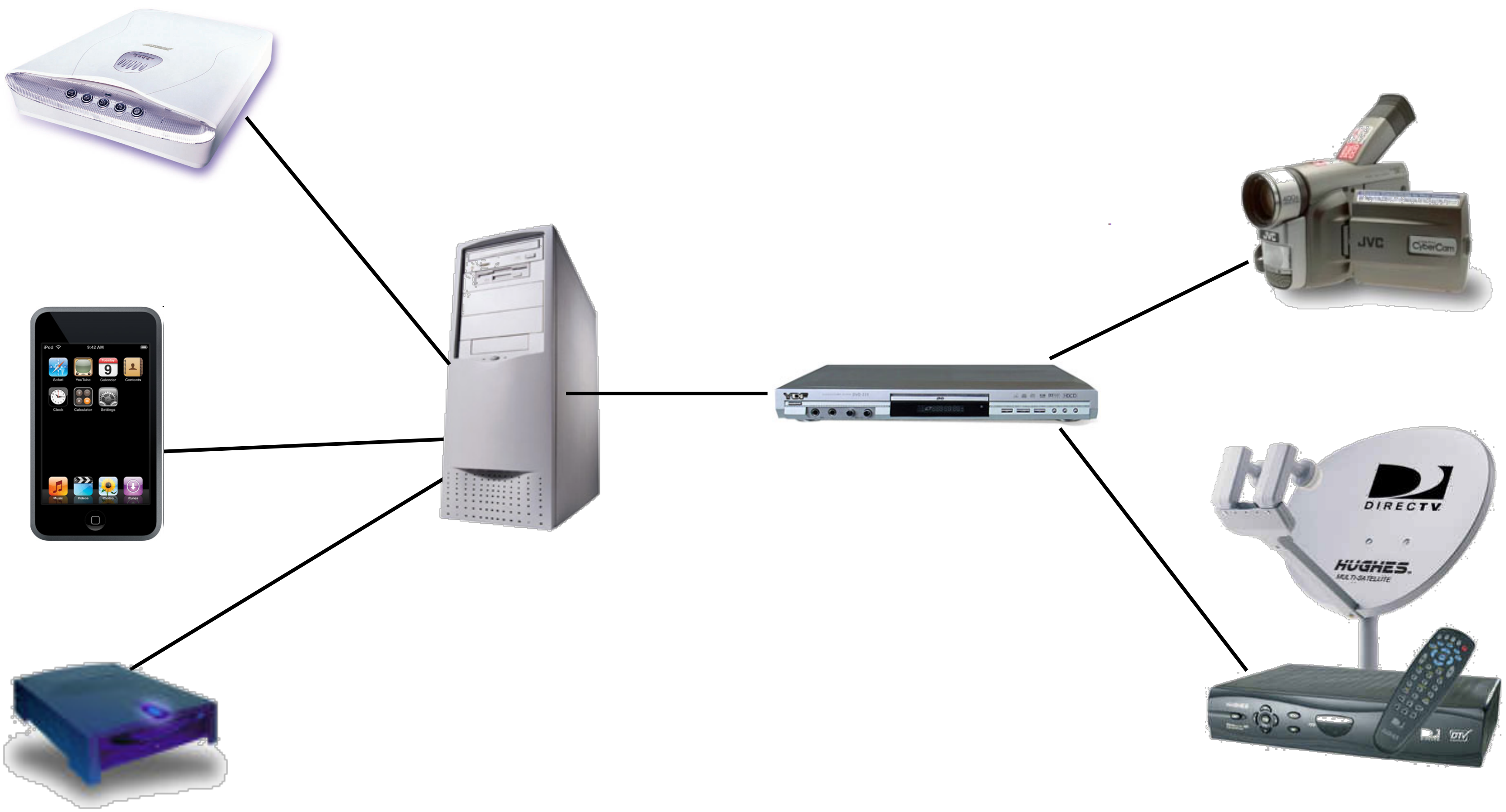
- Markov decision processes (MDPs)
- Adversaries & probability spaces
- Properties of MDPs: The temporal logic PCTL
- PCTL model checking for MDPs
- Case study: Firewire root contention

Case study: FireWire protocol

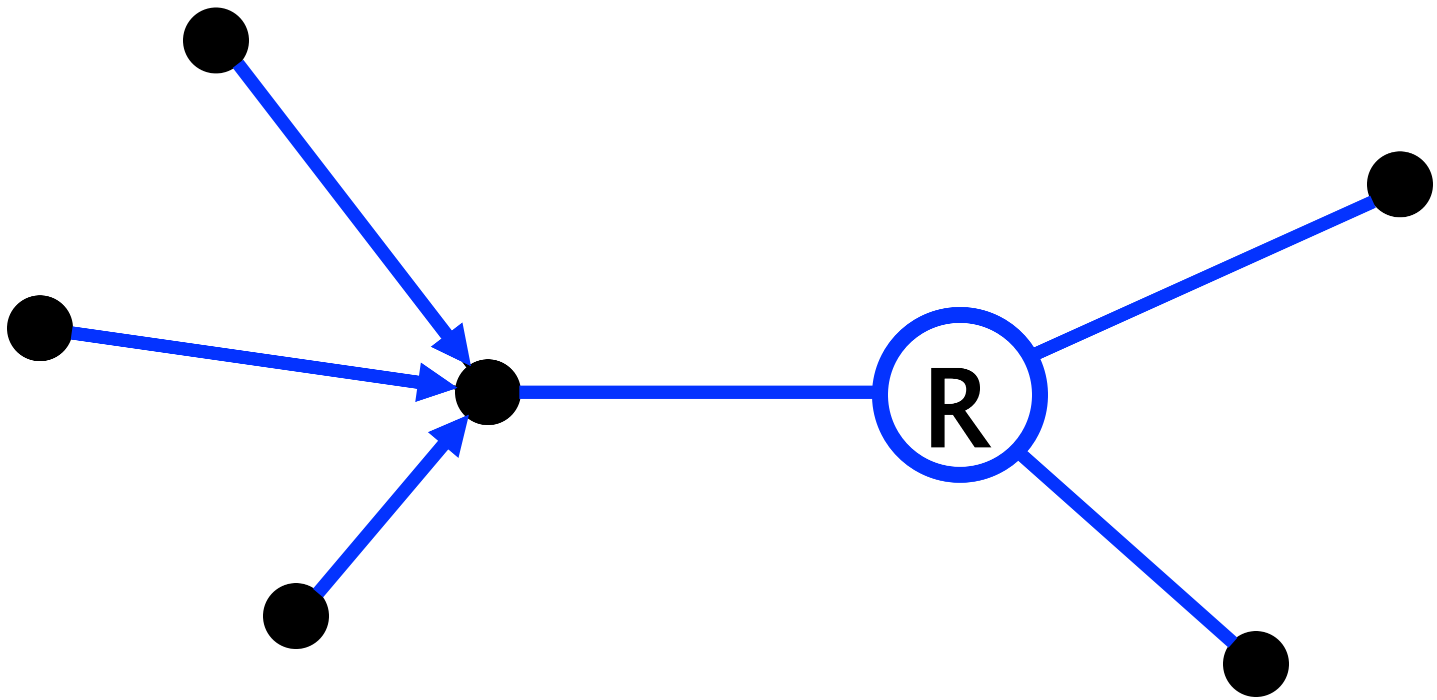
- FireWire (IEEE 1394)
  - high-performance serial bus for networking multimedia devices; originally by Apple
  - "hot-pluggable" - add/remove devices at any time
  - no requirement for a single PC (need acyclic topology)

- Root contention protocol
  - leader election algorithm, when nodes join/leave
  - symmetric, distributed protocol
  - uses electronic coin tossing and timing delays
  - nodes send messages: "be my parent"
  - root contention: when nodes contend leadership
  - random choice: "fast"/"slow" delay before retry

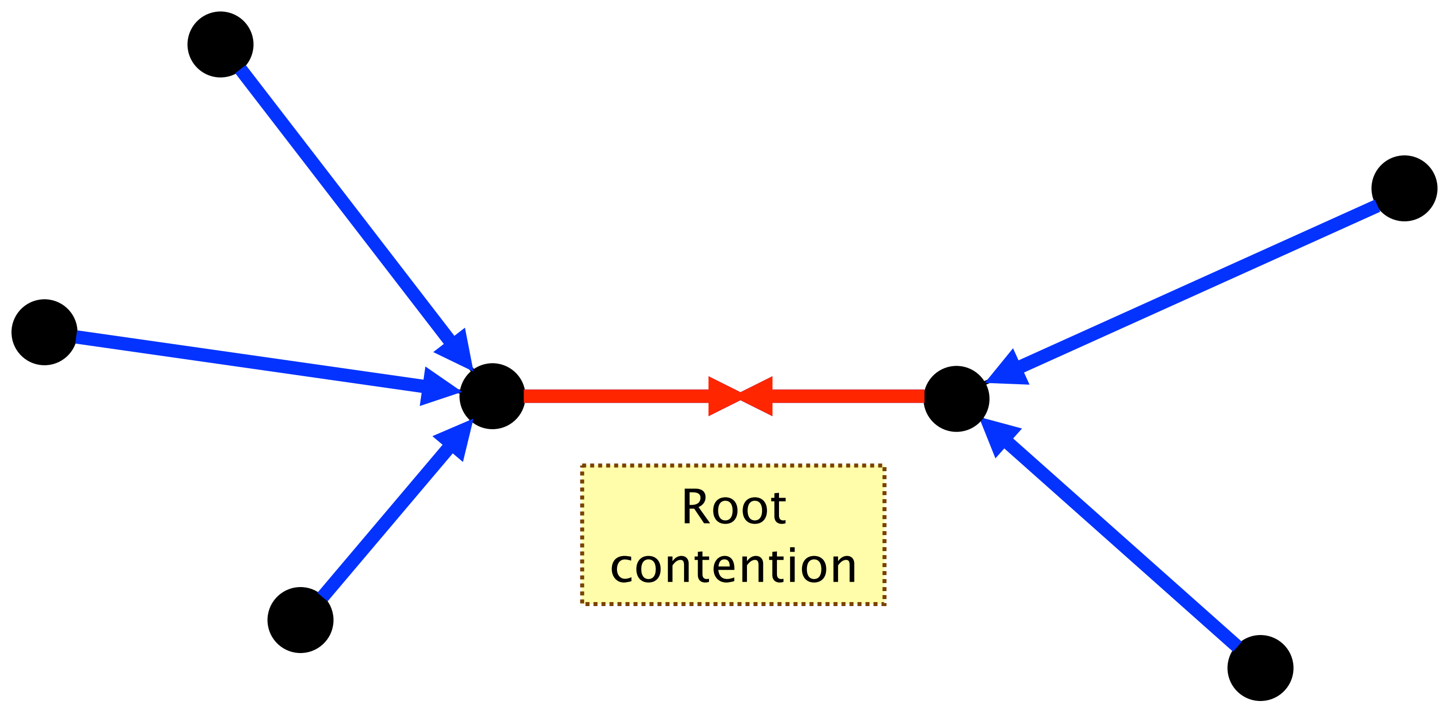
FireWire example



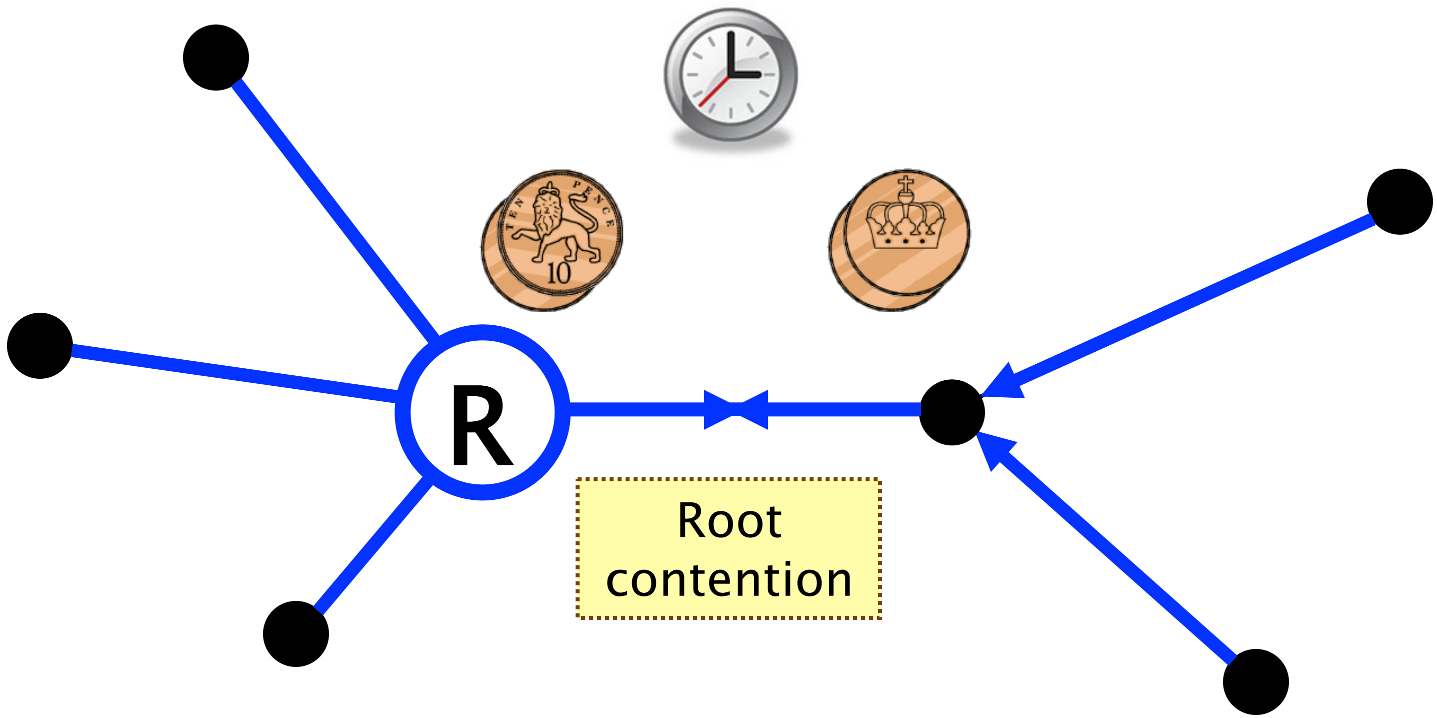
FireWire leader election



FireWire root contention

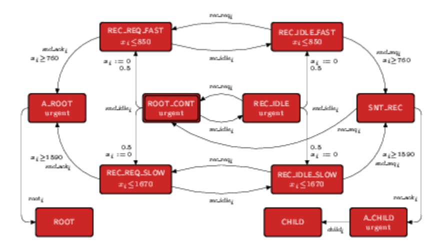


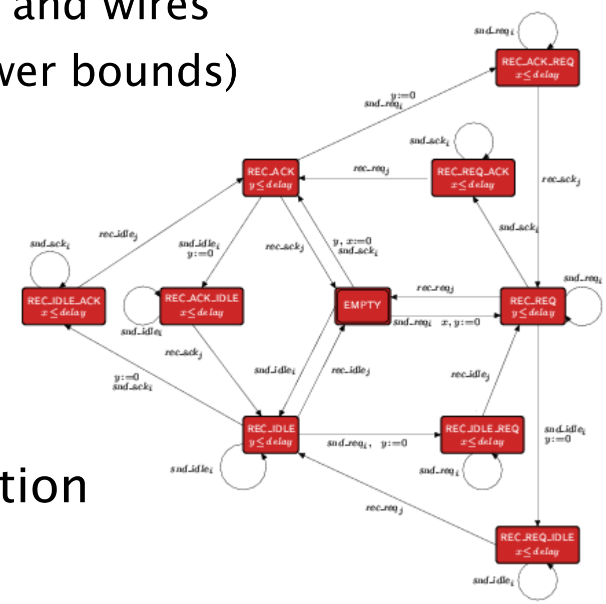
FireWire root contention



FireWire analysis

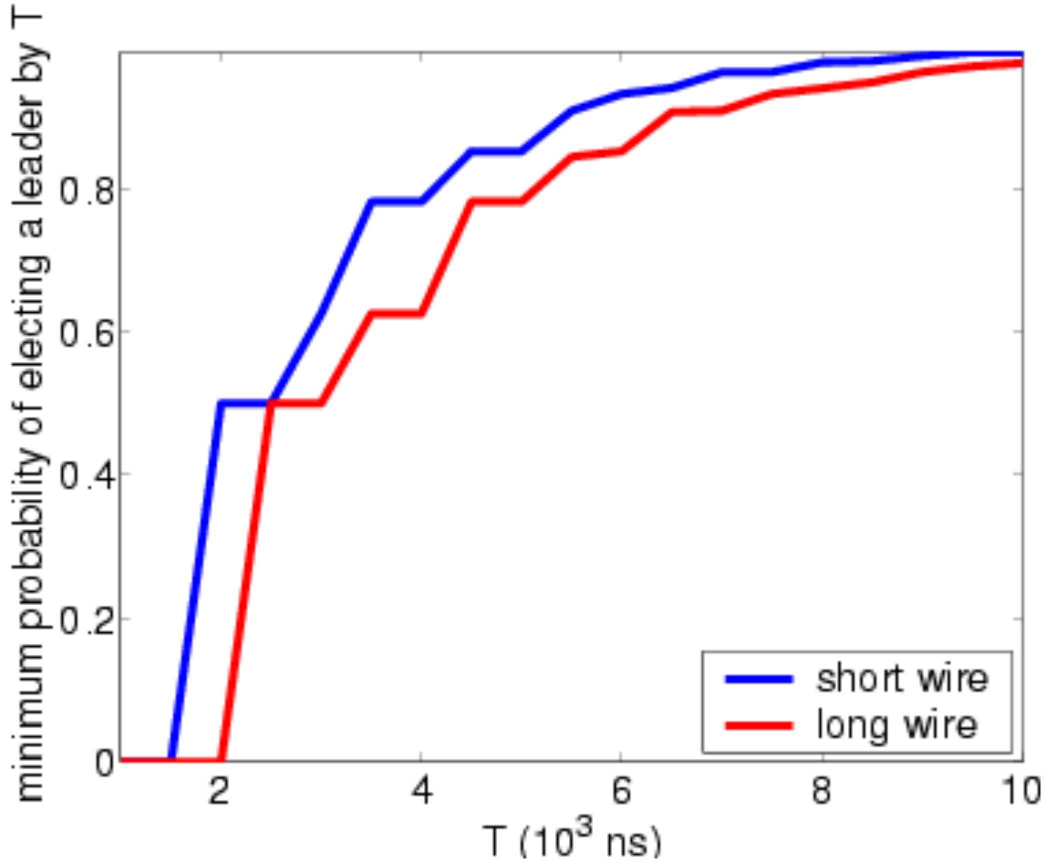
- Probabilistic model checking
  - model constructed and analysed using PRISM
  - timing delays taken from standard
  - model includes:
    - concurrency: messages between nodes and wires
    - underspecification of delays (upper/lower bounds)
  - max. model size: 170 million states





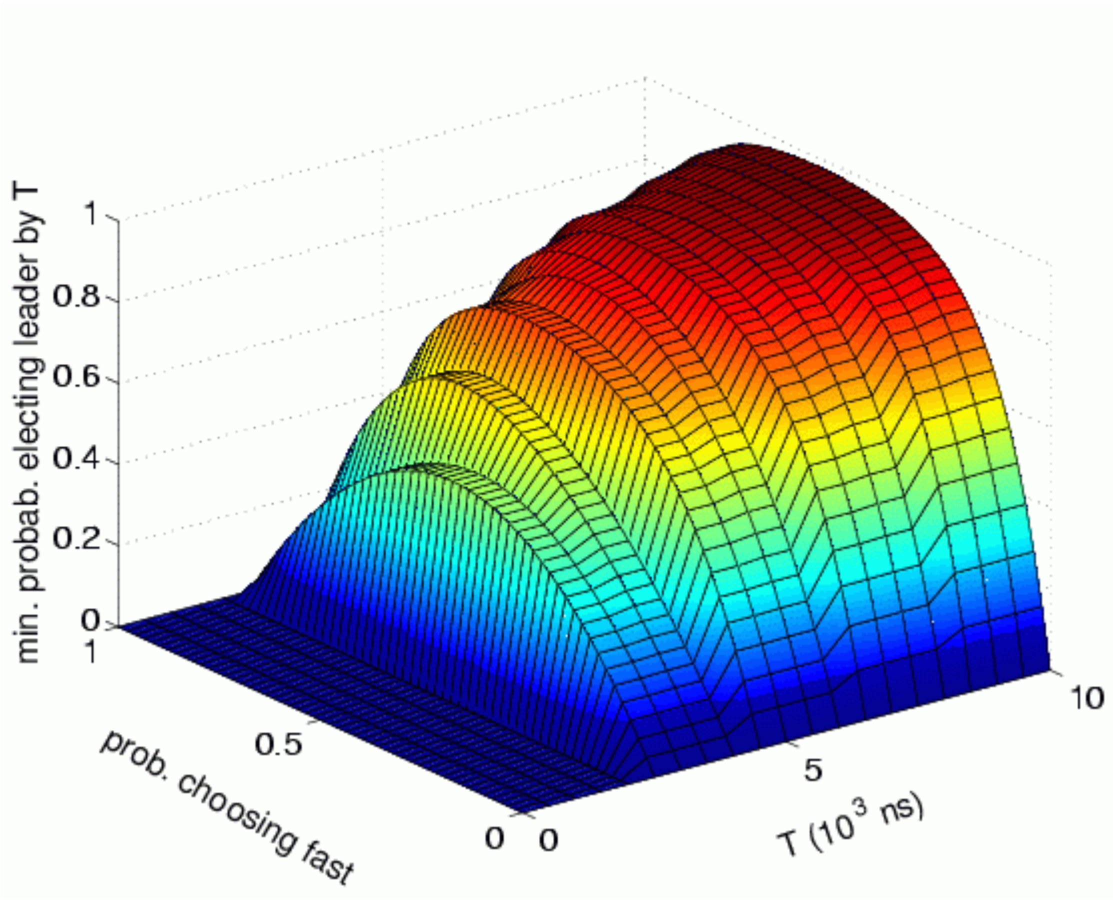
- Analysis:
  - verified that root contention always resolved with probability 1
  - investigated time taken for leader election
  - and the effect of using biased coin
    - based on a conjecture by Stoelinga

FireWire: Analysis results



"minimum probability of electing leader by time T"

FireWire: Analysis results

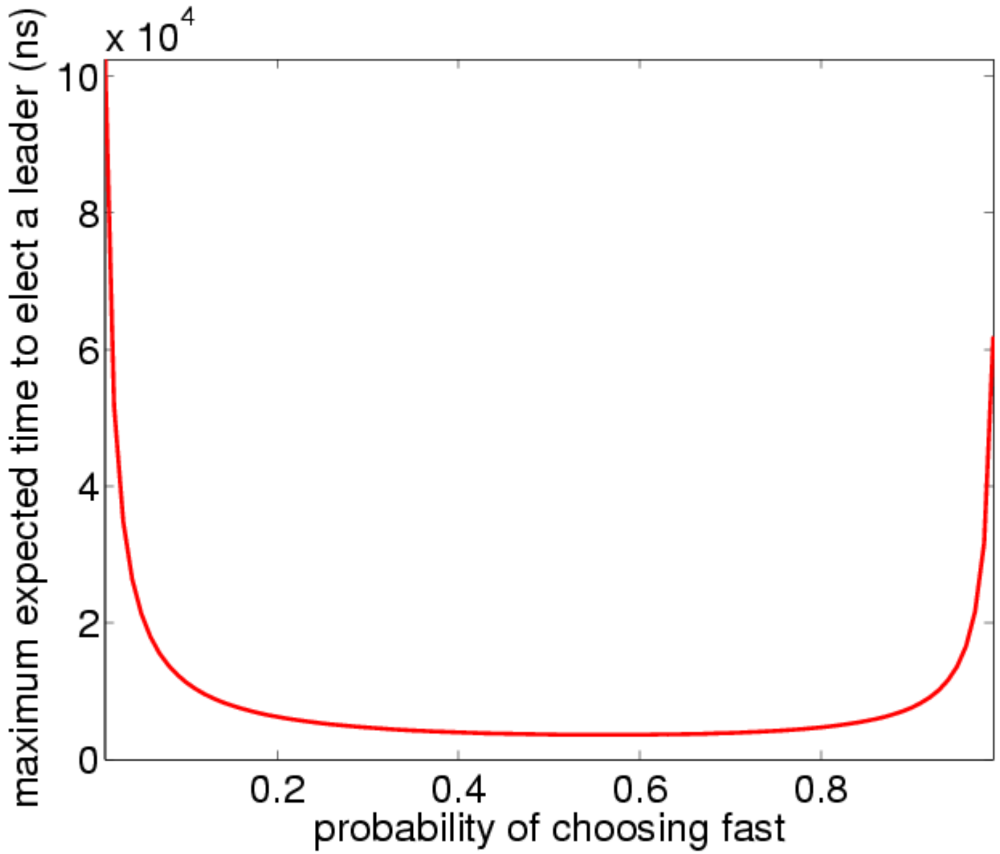


"minimum probability of electing leader by time T"

(short wire length)

Using a biased coin

FireWire: Analysis results

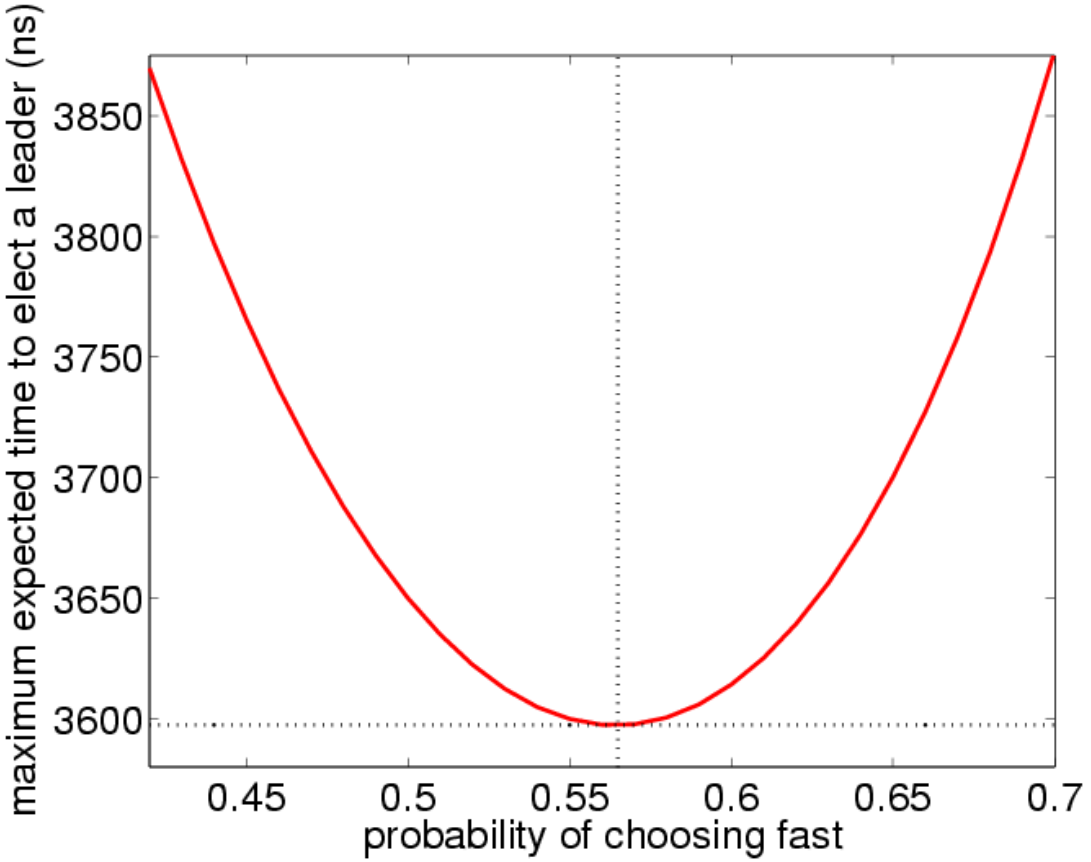


"maximum expected time to elect a leader"

(short wire length)

Using a biased coin

FireWire: Analysis results



"maximum expected time to elect a leader"

(short wire length)

Using a biased coin is beneficial!

Summary (Part 3)

- Markov decision processes (MDPs)
  - extend DTMCs with nondeterminism
  - to model concurrency, underspecification, …
- Adversaries resolve nondeterminism in an MDP
  - induce a probability space over paths
  - consider minimum/maximum probabilities over all adversaries
- Property specifications
  - PCTL: exactly same syntax as for DTMCs
  - but quantify over all adversaries
- Model checking algorithms
  - covered three basic techniques for MDPs: linear programming, value iteration, or policy iteration

- Next: Compositional probabilistic verification

Part 4

Compositional probabilistic verification

Overview

- Lectures 1 and 2:
  - 1 – Introduction
  - 2 – Discrete-time Markov chains
  - 3 – Markov decision processes
  - 4 – Compositional probabilistic verification
- PRISM lab session (4.30pm)
  - PC lab downstairs – or install PRISM on your own laptop
- Course materials available here:
  - http://www.prismmodelchecker.org/courses/sfm11connect/
  - lecture slides, reference list, tutorial chapter, lab session

Overview (Part 4)

- Compositional verification
  - assume-guarantee reasoning
- Markov decision processes
  - probabilistic safety properties
  - multi-objective model checking
- Probabilistic assume guarantee
  - semantics, model checking
  - assume-guarantee proof rules
  - quantitative approaches
  - implementation & experimental results
  - assumption generation with learning

Compositional verification

- Goal: scalability through modular verification
  - e.g. decide if $M_1 || M_2 \models G$
  - by analysing M_1 and M_2 separately

- Assume-guarantee (AG) reasoning
  - use assumption A about the context of a component M_2
  - $\langle A \rangle\ M_2\ \langle G \rangle$ – "whenever M_2 is part of a system satisfying A, then the system must also guarantee G"
  - example of asymmetric (non-circular) A/G rule:

$$\frac{M_1 \models A \qquad \langle A \rangle\ M_2\ \langle G \rangle}{M_1 \ || \ M_2 \models G}$$

[Pasareanu/Giannakopoulou/et al.]

AG rules for probabilistic systems

- How to formulate AG rules for MDPs?

$$\frac{M_1 \models A \qquad \langle A \rangle\, M_2\, \langle G \rangle}{M_1 \,||\, M_2 \models G}$$

- Key questions:
  - 1. What form do assumptions A take?
    - needs to be compositional
    - needs to be efficient to check
    - needs to allow compact assumptions
  - 2. How do we generate suitable assumptions?
    - preferably in a fully automated fashion
  - 3. Can we get "quantitative" results?
    - i.e. numerical values, rather than "yes"/"no"

AG rules for probabilistic systems

- How to formulate AG rules for MDPs?

$$\frac{M_1 \models A \qquad \langle A \rangle\, M_2\, \langle G \rangle}{M_1 \,||\, M_2 \models G}$$

- Key questions:
  - 1. What form do assumptions A take?
    - needs to be compositional
    - needs to be efficient to check
    - needs to allow compact assumptions

  ▷ various compositional relations exist
    - e.g. strong/weak (probabilistic) (bi)simulation
    - but these are either too fine (difficult to get small assumptions) or expensive to check

  ▷ here, we use: probabilistic safety properties [TACAS'10]
    - less expressive, but compact and efficient
    - (see also generalisation to liveness/rewards [TACAS'11])

AG rules for probabilistic systems

- How to formulate AG rules for MDPs?

$$\frac{M_1 \models A \quad \langle A \rangle\, M_2\, \langle G \rangle}{M_1 \,||\, M_2 \models G}$$

- Key questions:
  - 2. How do we generate suitable assumptions?
    - preferably in a fully automated fashion

  ▷ algorithmic learning (based on L* algorithm)
  adapt techniques for (non-probabilistic) assumptions

  - 3. Can we get “quantitative” results?
    - i.e. numerical values, rather than “yes”/”no”

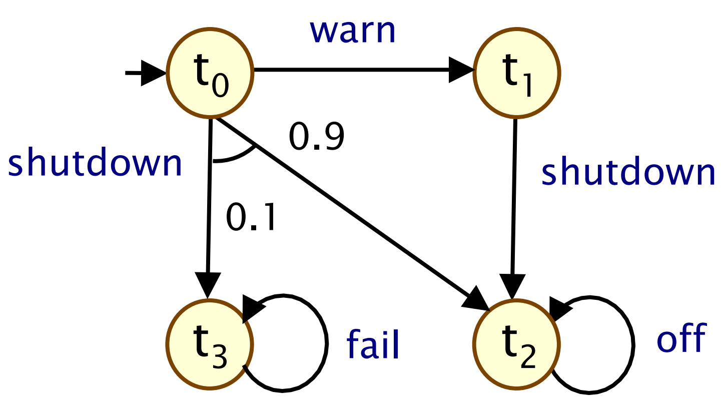
  ▷ yes: generate lower/upper bounds on probabilities

Overview (Part 4)

- Compositional verification
  - assume-guarantee reasoning
- Markov decision processes
  - probabilistic safety properties
  - multi-objective model checking
- Probabilistic assume guarantee
  - semantics, model checking
  - assume-guarantee proof rules
  - quantitative approaches
  - implementation & experimental results
  - assumption generation with learning

Recap: Markov decision processes

- Markov decision processes (MDPs)
  - model probabilistic and nondeterministic behaviour
- An MDP is a tuple $M = (S, s_{init}, \alpha_M, \delta_M, L)$:
  - S is the state space
  - $s_{init} \in S$ is the initial state
  - α_M is the action alphabet
  - $\delta_M \subseteq S \times (\alpha_M \cup \tau) \times Dist(S)$ is the transition probability relation
  - $L : S \rightarrow 2^{AP}$ labels states with atomic propositions



- Notes:
  - α_M, δ_M have subscripts to avoid confusion with other automata
  - transitions can also be labelled with a "silent" τ action
  - we write $s \xrightarrow{a} \mu$ as shorthand for $(s,a,\mu) \in \delta_M$
  - MDPs, here, are identical to probabilistic automata [Segala]

Recap: Model checking for MDPs

- An adversary σ resolves the nondeterminism in an MDP M
  - make a (possibly randomised) choice, based on history
  - induces probability measure Pr_M^σ over (infinite) paths $Path_M^\sigma$
  - can compute probability of some measurable property ϕ
    - e.g. $F\ err \equiv \Diamond err$ – "an error eventually occurs"
    - or automata over action labels (see later)

- Property specifications: quantify over all adversaries
  - e.g. PCTL: $M \vDash P_{\geq p}[\phi] \;\Leftrightarrow\; Pr_M^\sigma(\phi) \geq p$ for all adv.s $\sigma \in Adv_M$
  - corresponds to best-/worst-case behaviour analysis
  - requires computation of $Pr_M^{min}(\phi)$ or $Pr_M^{max}(\phi)$
  - or in a more quantitative fashion:
  - just ask e.g. $P_{min=?}(\phi)$ or $P_{max=?}(\phi)$
  - also extends to (min/max) expected costs & rewards

Parallel composition for MDPs

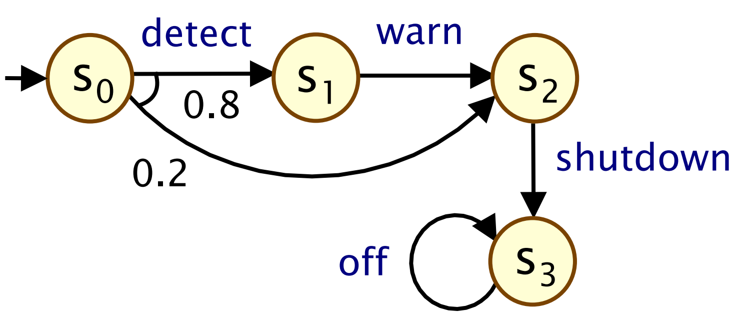
- The parallel composition of M_1 and M_2 is denoted $M_1 \,||\, M_2$
  - CSP style: synchronise over all common (non-τ) actions
  - when synchronising, transition probabilities are multiplied

- Formally, if $M_i = (S_i, s_{init,i}, \alpha_{M_i}, \delta_{M_i}, L_i)$ for $i=1,2$, then:
- $M_1||M_2 = (S_1 \times S_2, (s_{init,1},s_{init,2}), \alpha_{M_1} \cup \alpha_{M_2}, \delta_{M_1||M_2}, L_{12})$ where:
  - $L_{12}(s_1,s_2) = L_1(s_1) \cup L_2(s_2)$
  - $\delta_{M_1||M_2}$ is defined such that $(s_1,s_2) \xrightarrow{a} \mu_1 \times \mu_2$ iff one of:
    - $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ and $a \in \alpha_{M_1} \cap \alpha_{M_2}$ (synchronous)
    - $s_1 \xrightarrow{a} \mu_1$, $\mu_2 = \eta_{s_2}$ and $a \in (\alpha_{M_1} \setminus \alpha_{M_2}) \cup \{\tau\}$ (asynchronous)
    - $s_2 \xrightarrow{a} \mu_2$, $\mu_1 = \eta_{s_1}$ and $a \in (\alpha_{M_2} \setminus \alpha_{M_1}) \cup \{\tau\}$ (asynchronous)
  - where $\mu_1 \times \mu_2$ denotes the product of distributions μ_1, μ_2
  - and $\eta_s \in Dist(S)$ is the Dirac (point) distribution on $s \in S$

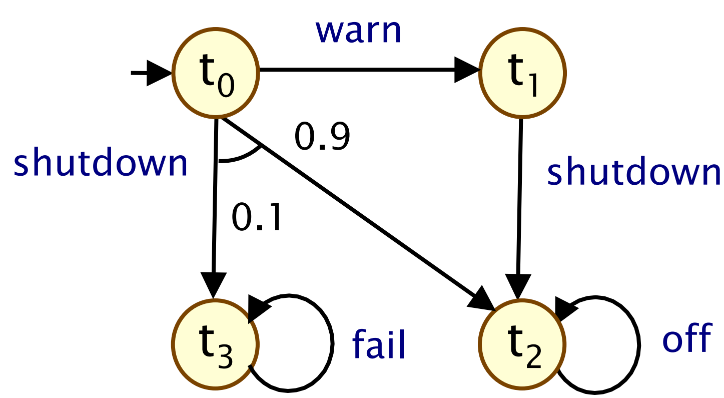
Running example

- Two components, each a Markov decision process:
  - M_1: controller which shuts down devices (after warning first)
  - M_2: device to be shut down (may fail if no warning sent)

MDP M_1 ("controller")

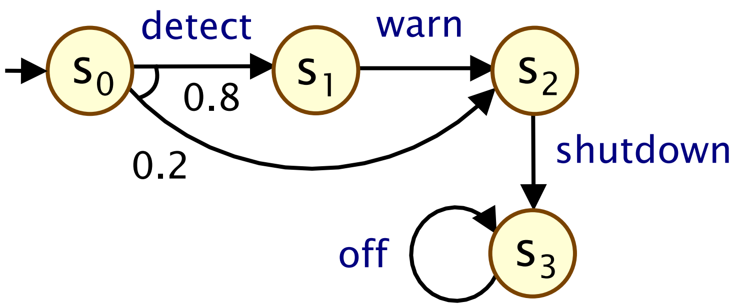


MDP M_2 ("device")

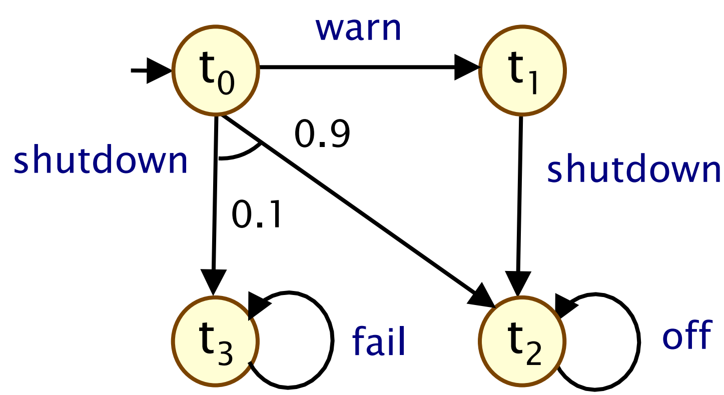


Running example

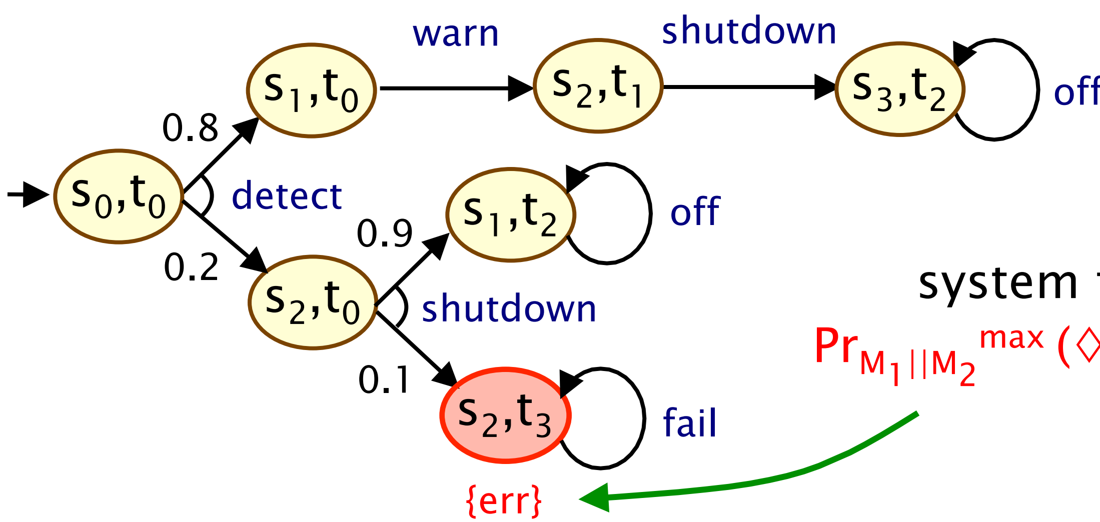
MDP M_1 ("controller")



MDP M_2 ("device")



Parallel composition: $M_1 \,||\, M_2$

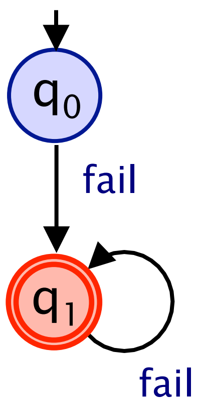


system failure:

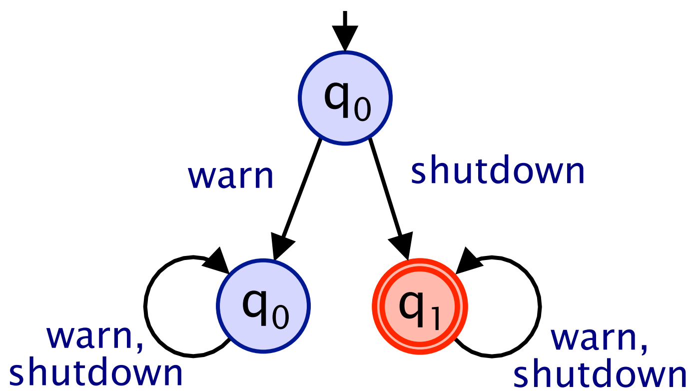
$Pr_{M_1||M_2}^{max}(\Diamond err) = 0.02$

Safety properties

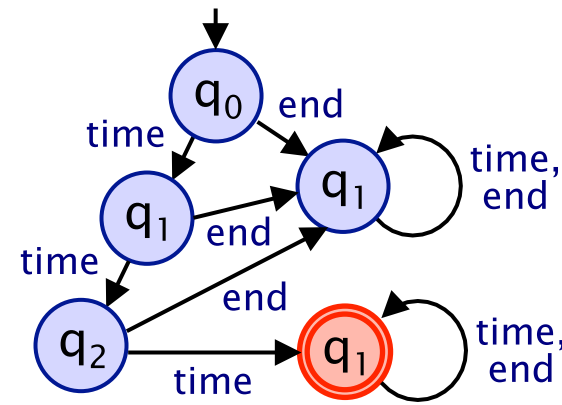
- Safety property: language of infinite words (over actions)
  - characterised by a set of “bad prefixes” (or “finite violations”)
  - i.e. finite words of which any extension violates the property
- Regular safety property
  - bad prefixes are represented by a regular language
  - property A stored as deterministic finite automaton (DFA) A_{err}



“a fail action never occurs”



“warn occurs before shutdown”



“at most 2 time steps pass before termination”

Probabilistic safety properties

- A probabilistic safety property $P_{\geq p}$ [A] comprises
  - a regular safety property A + a rational probability bound p
  - "the probability of satisfying A must be at least p"
  - $M \vDash P_{\geq p}[A] \;\Leftrightarrow\; Pr_M^{\sigma}(A) \geq p$ for all $\sigma \in Adv_M \;\Leftrightarrow\; Pr_M^{min}(A) \geq p$

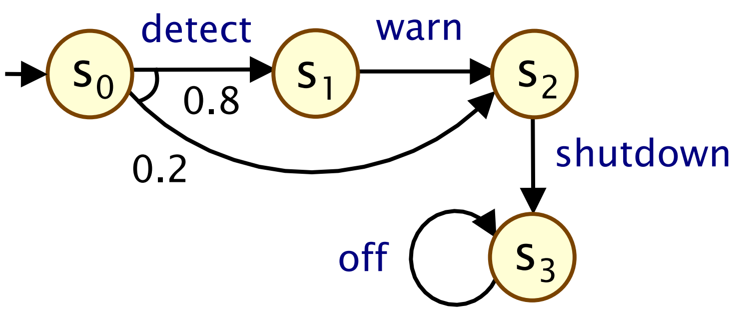
- Examples:
  - "*warn* occurs before *shutdown* with probability at least 0.8"
  - "the probability of a failure occurring is at most 0.02"
  - "probability of terminating within k time-steps is at least 0.75"

- Model checking: $Pr_M^{min}(A) = 1 - Pr_{M \otimes A_{err}}^{max}(\Diamond err_A)$
  - where err_A denotes "accept" states for DFA A
  - i.e. construct (synchronous) MDP-DFA product $M \otimes A_{err}$
  - then compute reachability probabilities on product MDP

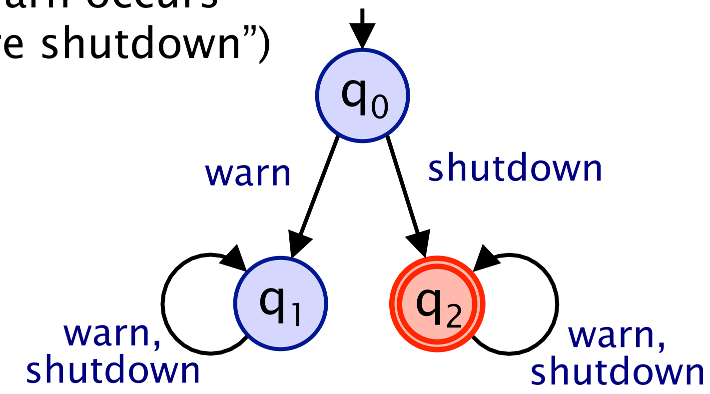
Running example

- Does probabilistic safety property $P_{\geq 0.8}$ [A] hold in M_1?

MDP M_1 ("controller")



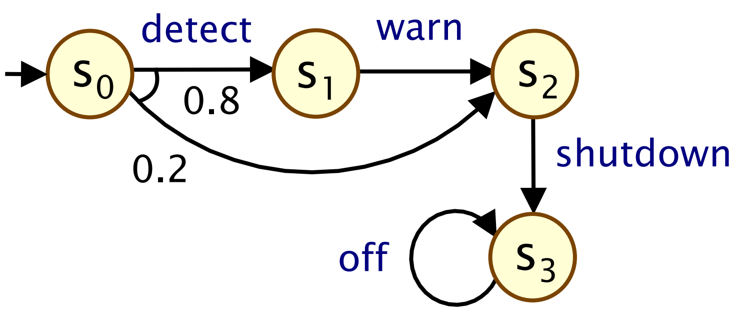
A ("warn occurs before shutdown")



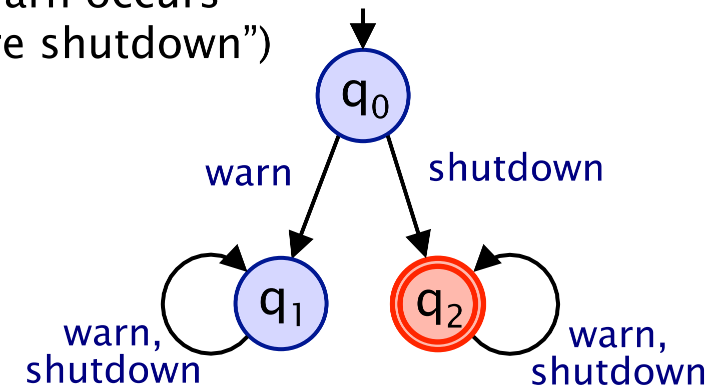
Running example

- Does probabilistic safety property $P_{\geq 0.8}$ [A] hold in M_1?

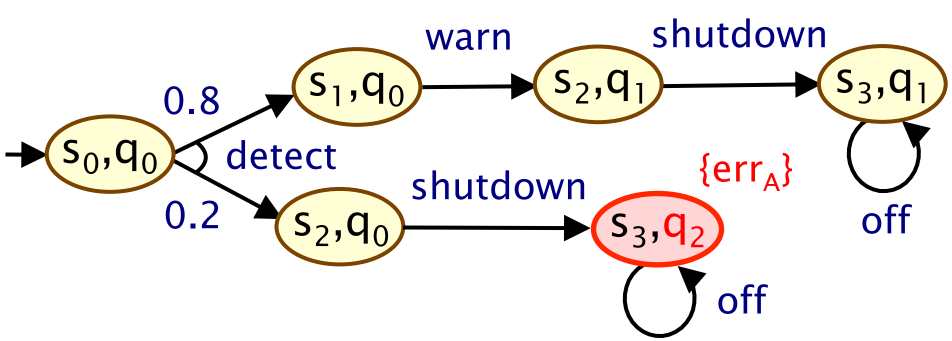
MDP M_1 ("controller")



A ("warn occurs before shutdown")



Product MDP $M_1 \otimes A_{err}$



$Pr_{M_1}^{min}(A)$

$= 1 - Pr_{M_1 \otimes A_{err}}^{max}(\lozenge err_A)$

$= 1 - 0.2$

$= 0.8$

$\rightarrow M_1 \models P_{\geq 0.8}$ [A]

Multi-objective MDP model checking

- Consider multiple (linear-time) objectives for an MDP M
  - LTL formulae $\Phi_1,\ldots,\Phi_k$ and probability bounds $\sim_1 p_1,\ldots,\sim_k p_k$
  - question: does there exist an adversary $\sigma \in Adv_M$ such that:

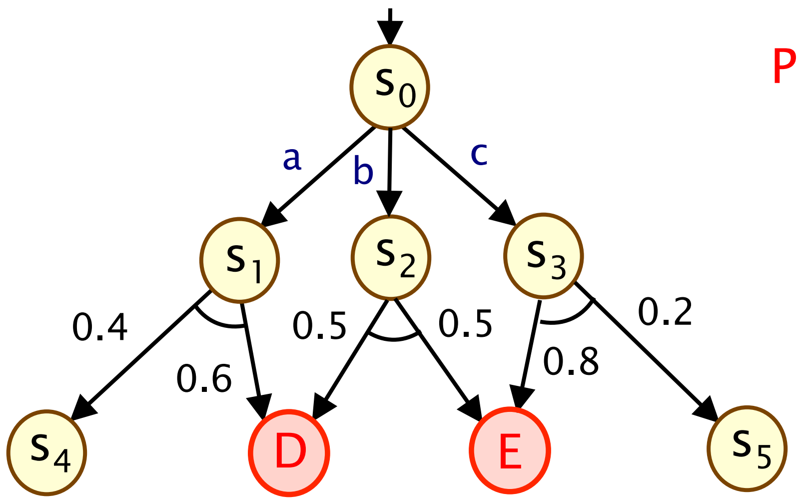
    $Pr_M^{\sigma}(\phi_1) \sim_1 p_1 \wedge \ldots \wedge Pr_M^{\sigma}(\phi_k) \sim_k p_k$

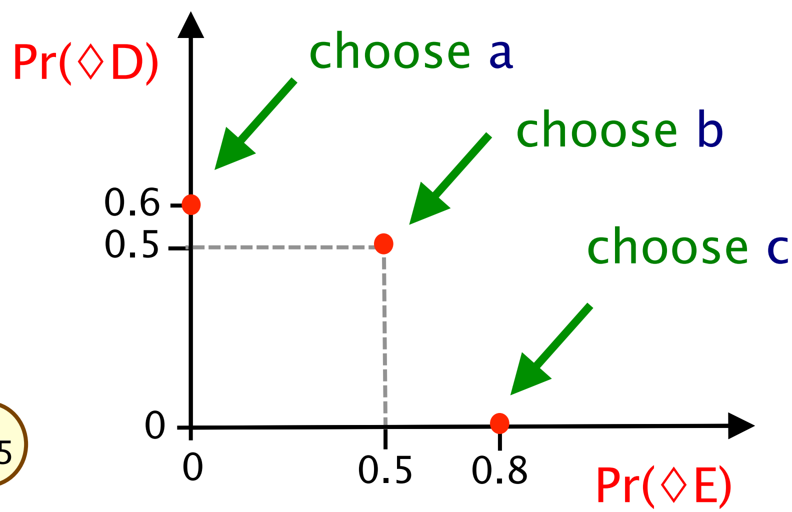
- Motivating example:
  - $Pr_M^{\sigma}(\Box(\text{queue\_size}<10)) > 0.99 \wedge Pr_M^{\sigma}(\Diamond \text{flat\_battery}) < 0.01$

- Multi-objective MDP model checking [EKVY07]
  - construct product of automata for M, $\Phi_1,\ldots,\Phi_k$
  - then solve linear programming (LP) problem
  - the resulting adversary σ can obtained from LP solution
  - note: σ may be randomised (unlike the single objective case)

Multi-objective MDP model checking

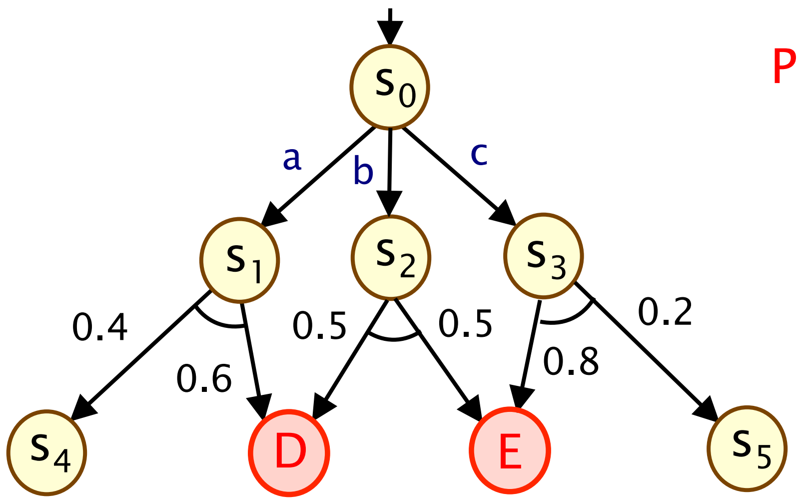
- Consider the two objectives $\Diamond D$ and $\Diamond E$ in the MDP below
  - i.e. the trade-off between the probabilities $Pr(\Diamond D)$ and $Pr(\Diamond E)$
  - an adversary resolves the choice between a/b/c
  - increasing the probability of reaching one target decreases the probability of reaching the other

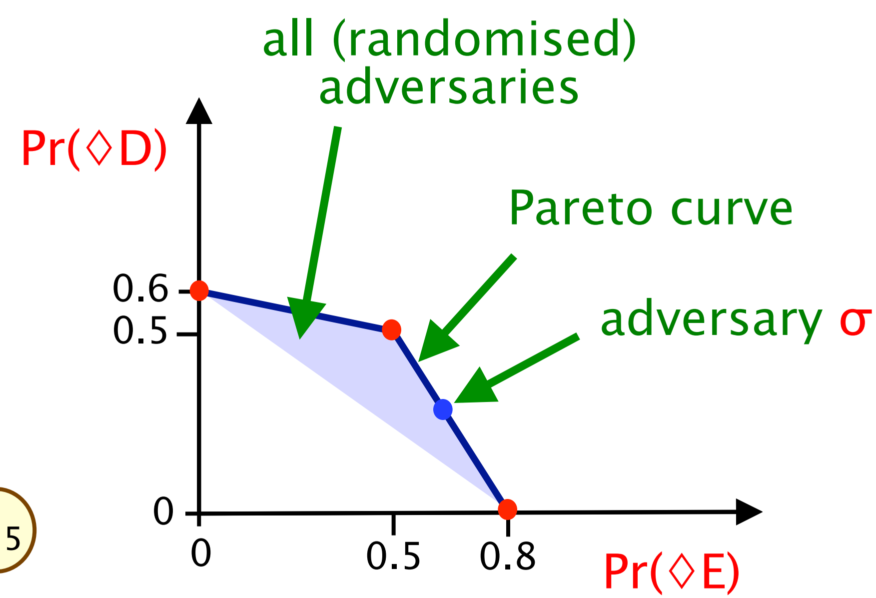




Multi-objective MDP model checking

- Need to consider all randomised adversaries
  - for example, is there an adversary σ such that:
  - $Pr(\Diamond D) > 0.2 \wedge Pr(\Diamond E) > 0.6$





Overview (Part 4)

- Compositional verification
  - assume-guarantee reasoning
- Markov decision processes
  - probabilistic safety properties
  - multi-objective model checking
- Probabilistic assume guarantee
  - semantics, model checking
  - assume-guarantee proof rules
  - quantitative approaches
  - implementation & experimental results
  - assumption generation with learning

Probabilistic assume guarantee

- Assume-guarantee triples $\langle A \rangle_{\geq p_A}$ M $\langle G \rangle_{\geq p_G}$ where:
  - M is an MDP
  - $P_{\geq p_A}[A]$ and $P_{\geq p_G}[G]$ are probabilistic safety properties

- Informally:
  - "whenever M is part of a system satisfying A with probability at least p_A, then the system is guaranteed to satisfy G with probability at least p_G"

- Formally:
  - $\forall \sigma \in Adv_{M'} \; (Pr_{M'}^{\sigma}(A) \geq p_A \rightarrow Pr_{M'}^{\sigma}(G) \geq p_G)$
  - where M' is M with its alphabet extended to include α_A
  - reduces to multi-objective model checking on M'
  - look for adversary satisfying assumption but not guarantee
  - i.e. can check $\langle A \rangle_{\geq p_A}$ M $\langle G \rangle_{\geq p_G}$ efficiently via LP problem

An assume-guarantee rule

- The following asymmetric proof rule holds
  - (asymmetric = uses one assumption about one component)

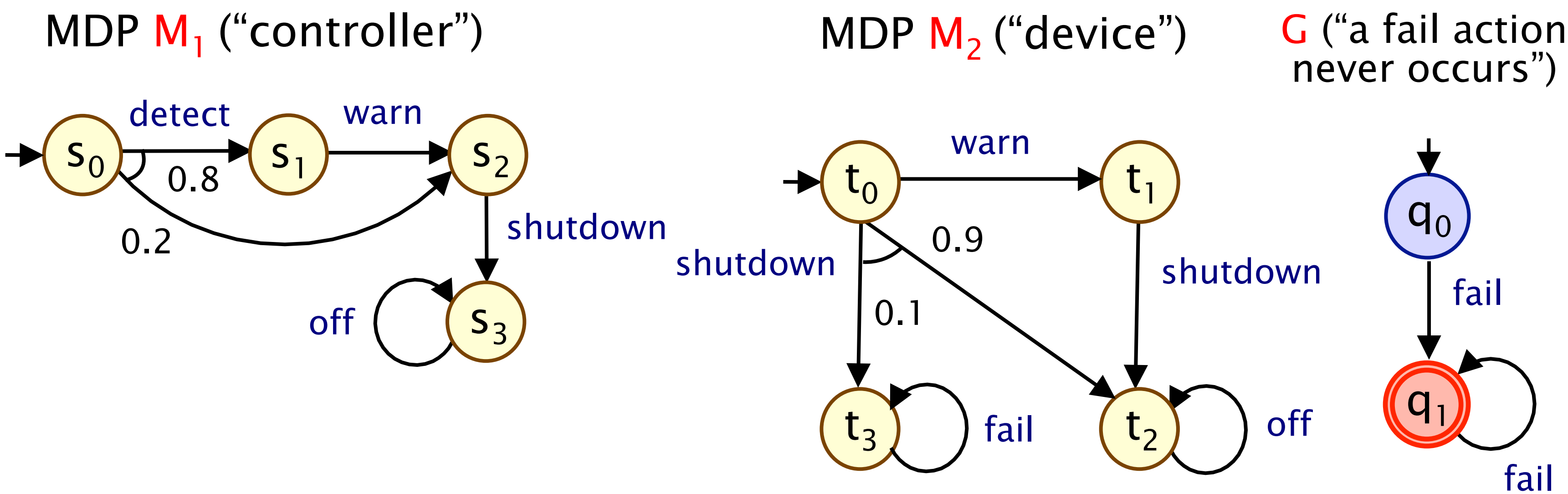
$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(ASYM)}$$

- So, verifying $M_1 \,||\, M_2 \models P_{\geq p_G}[G]$ requires:
  - premise 1: $M_1 \models P_{\geq p_A}[A]$ (standard model checking)
  - premise 2: $\langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}$ (multi-objective model checking)

- Potentially much cheaper if $|A|$ much smaller than $|M_1|$

Running example

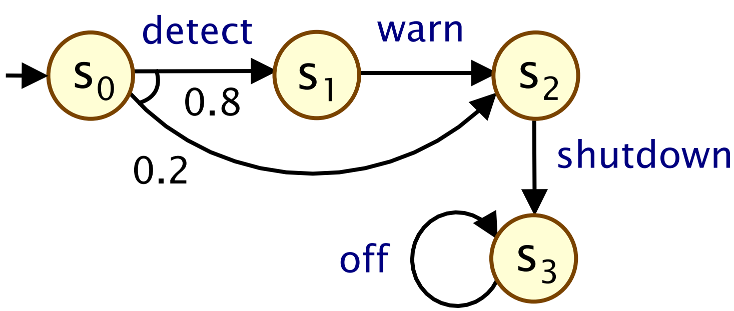
- Does probabilistic safety property $P_{\geq 0.98}$ [G] hold in $M_1||M_2$?



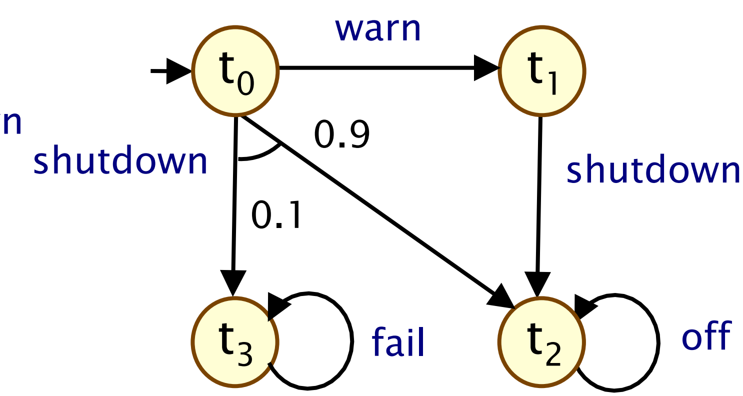
Running example

- Does probabilistic safety property $P_{\geq 0.98}$ [G] hold in $M_1||M_2$?

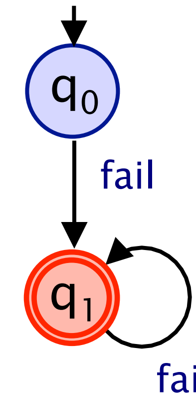
MDP M_1 ("controller")



MDP M_2 ("device")



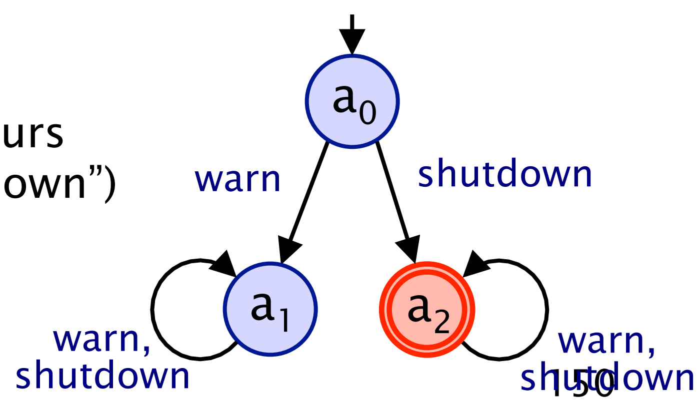
G ("a fail action never occurs")



- Use AG with assumption $\langle A \rangle_{\geq 0.8}$ about M_1

$$\frac{\langle true \rangle\ M_1\ \langle A \rangle_{\geq 0.8} \qquad \langle A \rangle_{\geq 0.8}\ M_2\ \langle G \rangle_{\geq 0.98}}{\langle true \rangle\ M_1 \mid\mid M_2\ \langle G \rangle_{\geq 0.98}}$$

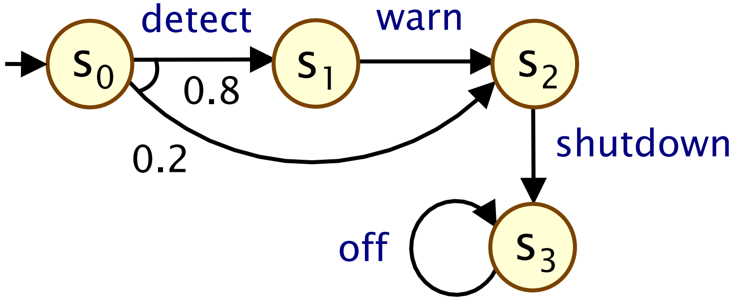
A ("warn occurs before shutdown")



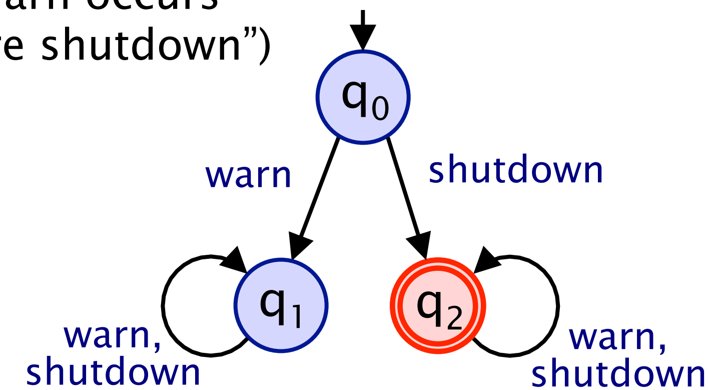
Running example

- Premise 1: Does $M_1 \vDash P_{\geq 0.8}$ [A] hold? (same as earlier ex.)

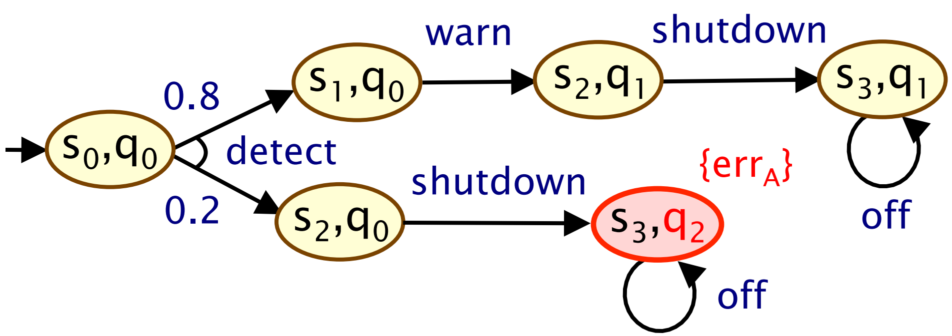
MDP M_1 ("controller")



A ("warn occurs before shutdown")



Product MDP $M_1 \otimes A_{err}$



$Pr_{M_1}^{min}(A)$

$= 1 - Pr_{M_1 \otimes A_{err}}^{max}(\Diamond err_A)$

$= 1 - 0.2$

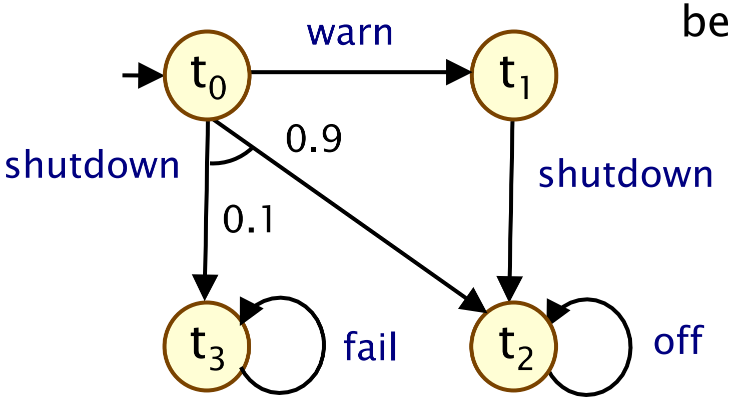
$= 0.8$

$\rightarrow M_1 \vDash P_{\geq 0.8}$ [A]

Running example

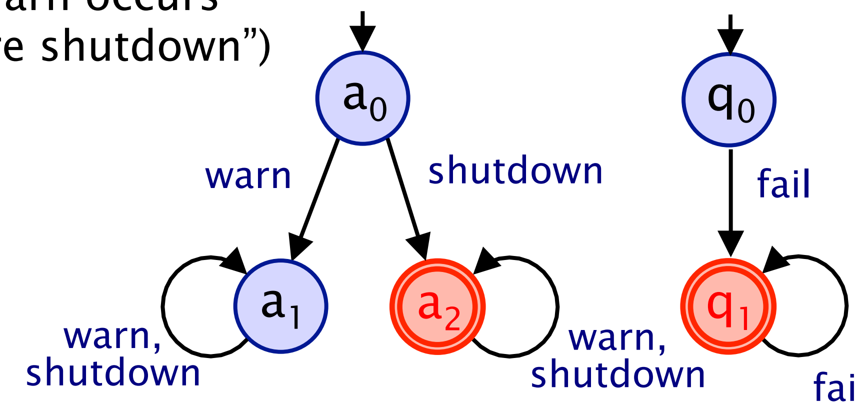
- Premise 2: Does $\langle A\rangle_{\geq 0.8}\ M_2\ \langle G\rangle_{\geq 0.98}$ hold?

MDP M_2 ("device")



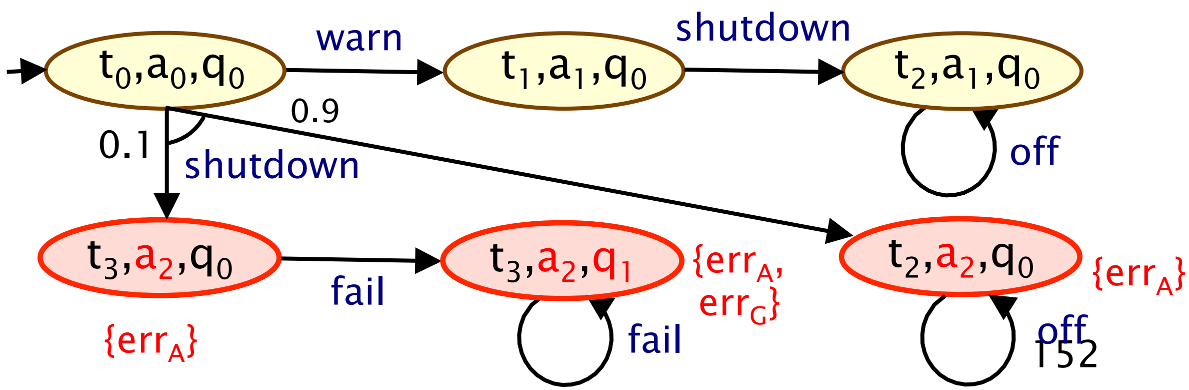
A ("warn occurs before shutdown")

G ("a fail action never occurs")



Product MDP

$M' = M_2[\alpha_A]\otimes A_{err}\otimes G_{err}$

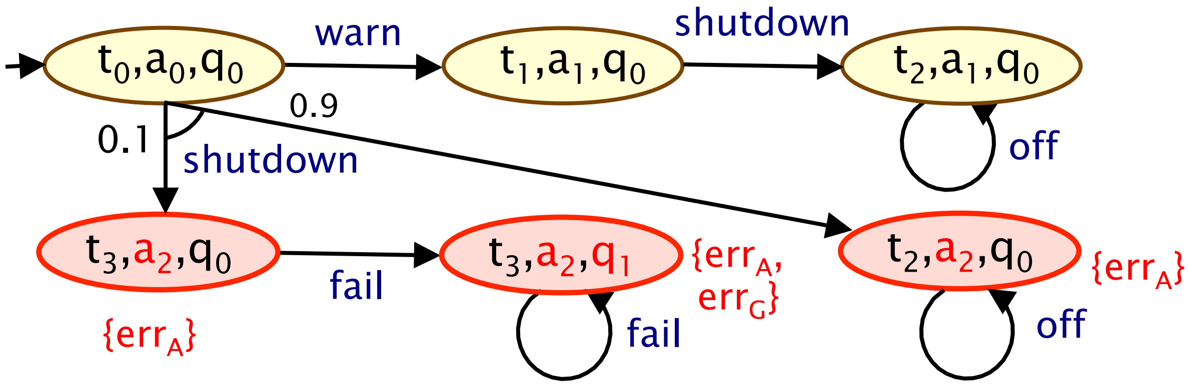


Running example

- Premise 2: Does $\langle A\rangle_{\geq 0.8}$ M_2 $\langle G\rangle_{\geq 0.98}$ hold?

Product MDP

$M' = M_2[\alpha_A] \otimes A_{err} \otimes G_{err}$



- ∃ an adversary of M_2 satisfying $Pr_M^\sigma(A) \geq 0.8$ but not $Pr_M^\sigma(G) \geq 0.98$?

  ⇔

- ∃ an an adversary of M' with $Pr_{M'}^{\sigma'}(\Diamond err_A) \leq 0.2$ and $Pr_{M'}^{\sigma'}(\Diamond err_G) > 0.02$?

- To satisfy $Pr_{M'}^{\sigma'}(\Diamond err_A) \leq 0.2$, adversary σ' must choose shutdown in initial state with probability ≤ 0.2, which means $Pr_{M'}^{\sigma'}(\Diamond err_G) \leq 0.02$

- So, there is no such adversary and $\langle A\rangle_{\geq 0.8}$ M_2 $\langle G\rangle_{\geq 0.98}$ <u>does</u> hold

Other assume-guarantee rules

- Multiple assumptions:

$$\frac{M_1 \models P_{\geq p_1}[A_1] \wedge \ldots \wedge P_{\geq p_k}[A_k] \qquad \langle A_1,\ldots,A_k\rangle_{\geq p_1,\ldots,p_k}\, M_2\, \langle G\rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(ASYM-MULT)}$$

Multiple components (chain):

$$\frac{M_1 \models P_{\geq p_1}[A_1] \qquad \langle A_1\rangle_{\geq p_1}\, M_2\, \langle A_2\rangle_{\geq p_2} \qquad \ldots \qquad \langle A_n\rangle_{\geq p_n}\, M_n\, \langle G\rangle_{\geq p_G}}{M_1 \,||\, \ldots \,||\, M_n \models P_{\geq p_G}[G]} \quad \text{(ASYM-N)}$$

- Circular rule:

$$\frac{M_2 \models P_{\geq p_2}[A_2] \qquad \langle A_2\rangle_{\geq p_2}\, M_1\, \langle A_1\rangle_{\geq p_1} \qquad \langle A_1\rangle_{\geq p_1}\, M_2\, \langle G\rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(CIRC)}$$

Asynchronous components:

$$\frac{\langle A_1\rangle_{\geq p_1}\, M_1\, \langle G_1\rangle_{\geq q_1} \qquad \langle A_2\rangle_{\geq p_2}\, M_2\, \langle G_2\rangle_{\geq q_2}}{\langle A_1,A_2\rangle_{\geq p_1 p_2}\, M_1 \,||\, M_2\, \langle G_1 \vee G_2\rangle_{\geq (q_1+q_2-q_1 q_2)}} \quad \text{(ASYNC)}$$

A quantitative approach

- For (non-compositional) probabilistic verification
  - prefer quantitative properties: $Pr_M^{min}(G)$, not $M \models P_{\geq p_G}$ [G]
  - can we do this for compositional verification?

$$\frac{\langle true \rangle\ M_1\ \langle A \rangle_{\geq p_A} \qquad \langle A \rangle_{\geq p_A}\ M_2\ \langle G \rangle_{\geq p_G}}{\langle true \rangle\ M_1 \ ||\ M_2\ \langle G \rangle_{\geq p_G}}$$

- Consider, for example, AG rule (ASym)
  - this proves $Pr_{M_1 \| M_2}^{min}(G) \geq p_G$ for certain values of p_G
  - i.e. gives lower bound for $Pr_{M_1 \| M_2}^{min}(G)$
  - for a fixed assumption A, we can compute the maximal lower bound obtainable, through a simple adaption of the multi-objective model checking problem
  - we can also compute upper bounds using generated adversaries as witnesses
  - furthermore: can explore trade-offs in parameterised models by approximating Pareto curves

Implementation + Case studies

- Prototype extension of PRISM model checker
  - already supports LTL for Markov decision processes
  - automata can be encoded in modelling language
  - added support for multi-objective LTL model checking, using LP solvers (ECLiPSe/COIN-OR CBC)

- Two large case studies
  - randomised consensus algorithm (Aspnes & Herlihy)
    - minimum probability consensus reached by round R
  - Zeroconf network protocol
    - maximum probability network configures incorrectly
    - minimum probability network configured by time T

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

- Faster than conventional model checking in a number of cases

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

- Verified instances where conventional model checking is infeasible

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

- LP problem generally much smaller than full state space (but still the limiting factor)

Overview (Part 4)

- Compositional verification
  - assume-guarantee reasoning

- Markov decision processes
  - probabilistic safety properties
  - multi-objective model checking

- Probabilistic assume guarantee
  - semantics, model checking
  - assume-guarantee proof rules
  - quantitative approaches
  - implementation & experimental results
  - assumption generation with learning

Generating assumptions

- Can model check $M_1 || M_2$ compositionally
  - but this relies on the existence of a suitable assumption $P_{\geq p_A}[A]$

$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A} \; M_2 \; \langle G \rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]}$$

- 1. Does such an assumption always exist?

- 2. When it does exist, can we generate it automatically?

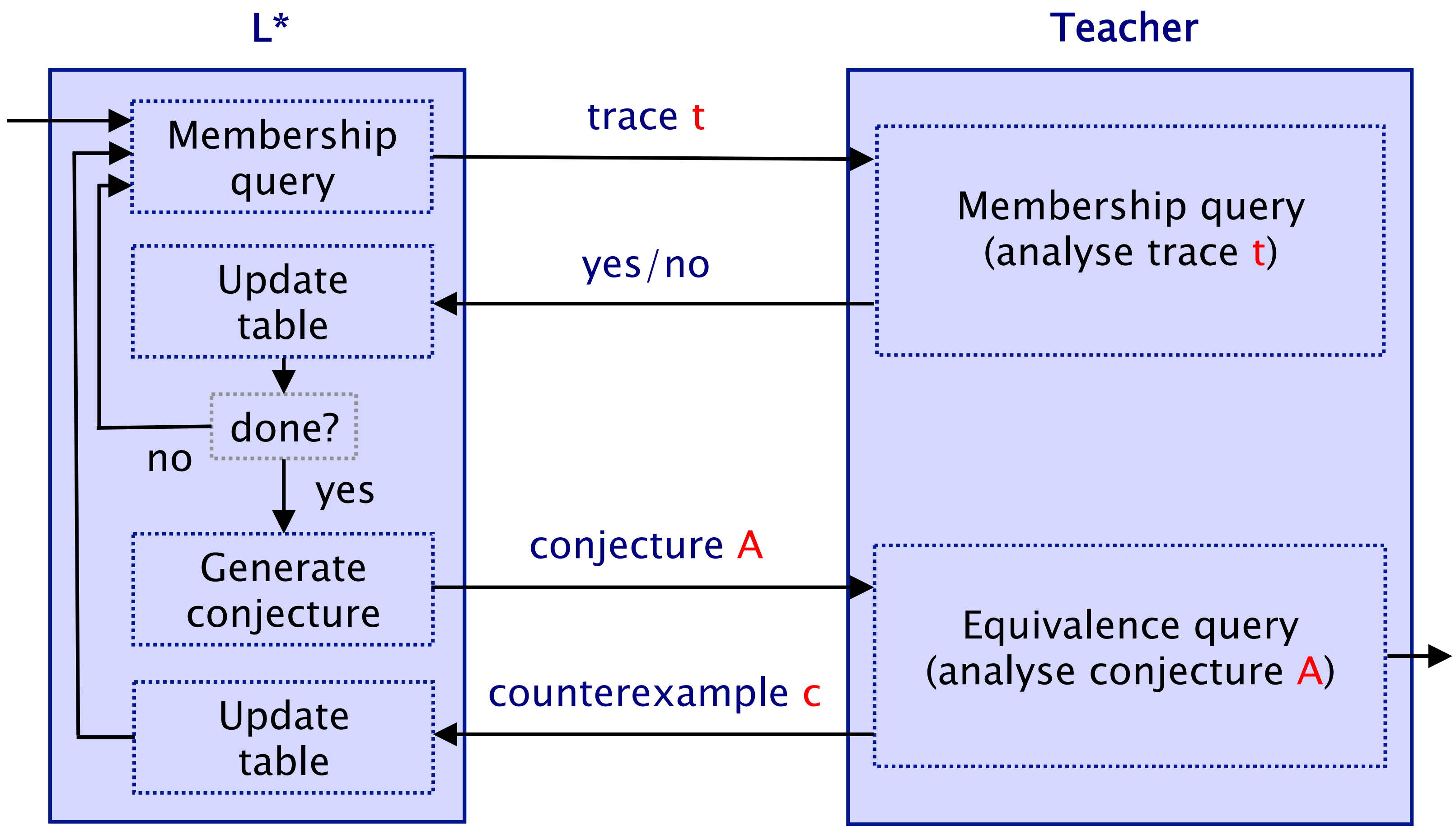
- Our approach: use algorithmic learning techniques
  - inspired by non-probabilistic AG work of [Pasareanu et al.]
  - uses L* algorithm to learn finite automata for assumptions
  - we use a modified version of L*
  - to learn probabilistic assumptions for rule (ASYM) [QEST'10]

The L* learning algorithm

- The L* algorithm [Angluin]
  - learns an unknown regular language L, as a (minimal) DFA

- Based on "active" learning
  - relies on existence of a "teacher" to guide the learning
  - answers two type of queries: "membership" and "equivalence"
  - membership: "is trace (word) t in the target language L?"
    - stores results of membership queries in observation table
    - based on these, generates conjectures A for the automata
  - equivalence: "does automata A accept the target language L"?
    - if not, teacher must return counterexample c
    - (c is a word in the symmetric difference of L and L(A))

The L* learning algorithm



L* for assume-guarantee

- Breakthrough in automated compositional verification
  - use of L* to learn assumptions for A/G reasoning
  - [Pasareanu/Giannakopoulou/et al.]
  - uses notion of "weakest assumption" about a component that suffices for compositional verification (always exists)
  - weakest assumption is the target regular language
- Fully automated L* learning loop
  - model checker plays role of teacher, returns counterexamples
  - in practice, can usually stop early: either with a simpler (stronger) assumption or by refuting the property
- Successfully applied to several large case studies
  - does particularly well when assumption/alphabet are small
  - much recent interest in learning for verification…

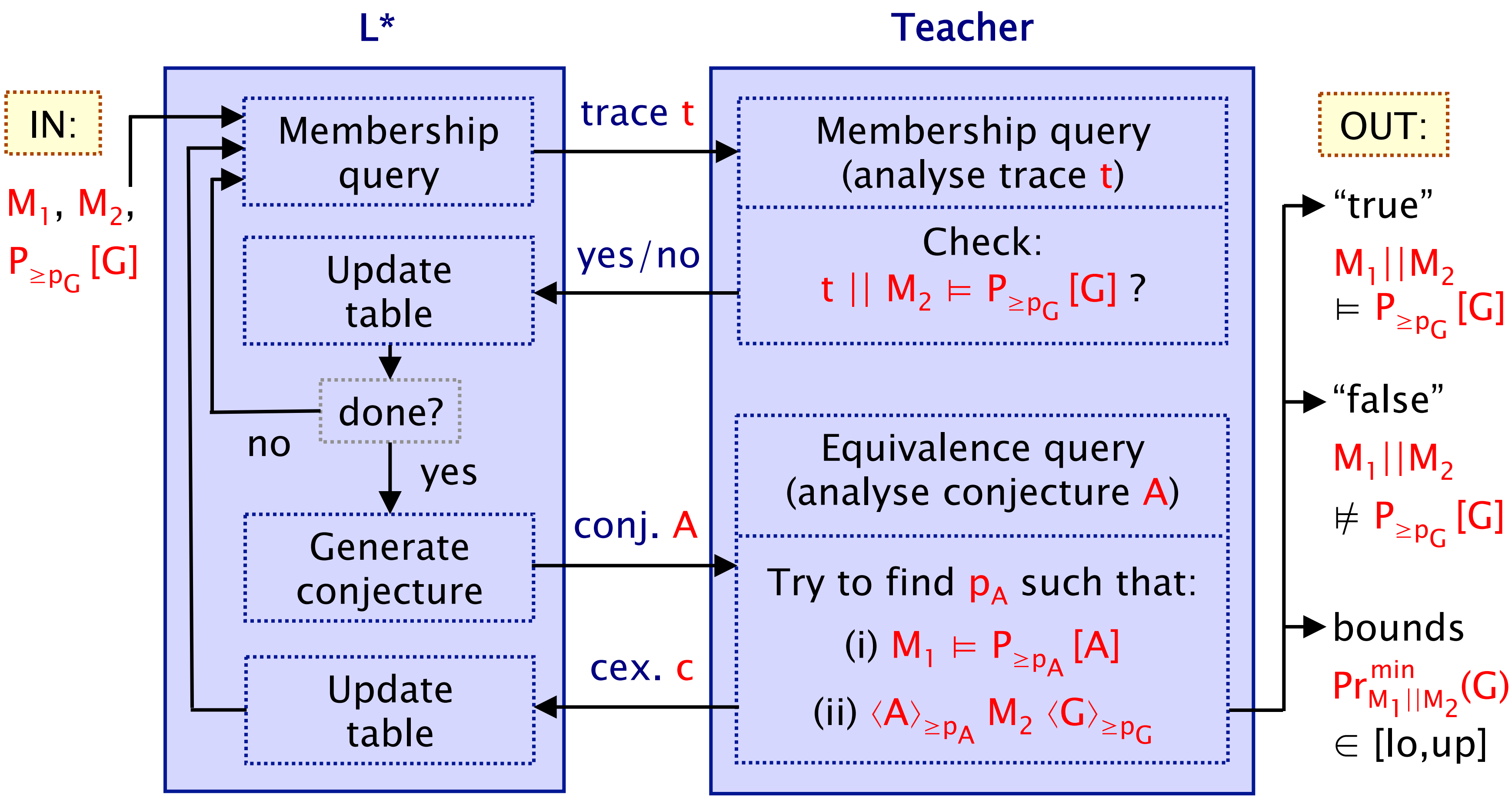
Probabilistic assumption generation

- Goal: automate A/G rule (ASYM)
  - generate probabilistic assumption $P_{\geq p_A}[A]$
  - for checking property $P_{\geq p_G}[G]$ on $M_1 \,||\, M_2$

$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A}\, M_2\, \langle G \rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]}$$

- Reduce problem to generation of non-probabilistic assumption A
  - then (if possible) find lowest p_A such that premises 1 & 2 hold
  - in fact, for fixed A, we can generate lower and upper bounds on $Pr_{M_1||M_2}^{min}(G)$, which may suffice to verify/refute $P_{\geq p_G}[G]$
- Use adapted L* to learn non-probabilistic assumption A
  - note: there is no "weakest assumption" (AG rule is incomplete)
  - but can generate sequence of conjectures for A in similar style
  - "teacher" based on a probabilistic model checker (PRISM), feedback is from probabilistic counterexamples [Han/Katoen]
  - three outcomes of loop: "true", "false", lower/upper bounds

Probabilistic assumption generation



Implementation + Case studies

- Implemented using:
  - extension of PRISM model checker
  - libalf learning library [Bollig et al.]

- Several case studies
  - client-server (A/G model checking benchmark + failures)
    - minimum probability mutual exclusion not violated
  - randomised consensus algorithm [Aspnes & Herlihy]
    - minimum probability consensus reached by round R
  - sensor network [QEST'10]
    - minimum probability of processor error occurring
  - Mars Exploration Rovers (MER) [NASA]
    - minimum probability mutual exclusion not violated in k cycles

Experimental results (learning)

Case study [parameters]		Component sizes		Compositional	
		$\|M_2 \otimes G_{err}\|$	$\|M_1\|$	$\|A^{err}\|$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

Experimental results (learning)

Case study [parameters]		Component sizes		Compositional	
		$\|M_2 \otimes G_{err}\|$	$\|M_1\|$	$\|A^{err}\|$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

- Successfully learnt (small) assumptions in all cases

Experimental results (learning)

Case study [parameters]		Component sizes		Compositional	
		$\lvert M_2 \otimes G_{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

- In some cases, learning + compositional verification is faster (than non-compositional verification, using PRISM)

Summary (Part 4)

- Compositional verification, e.g. assume-guarantee
  - decompose verification problem based on system structure
- Compositional probabilistic verification based on:
  - Markov decision processes, with arbitrary parallel composition
  - assumptions/guarantees are probabilistic safety properties
  - reduction to multi-objective model checking
  - multiple proof rules; adapted to quantitative approach
  - automatic generation of assumptions: L* learning
- Can work well in practice
  - verified safety/performance on several large case studies
  - cases where infeasible using non-compositional verification
- For further detail, see [KNPQ10], [FKP10], [FKN+11]

- Next: PRISM lab session…

Thanks for your attention

More info here:
www.prismmodelchecker.org