Lecture 1
Introduction

Dr. Dave Parker

Department of Computer Science
University of Oxford

Probabilistic model checking

- Probabilistic model checking…
  - is a formal verification technique for modelling and analysing systems that exhibit probabilistic behaviour

- Formal verification…
  - is the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems

Outline

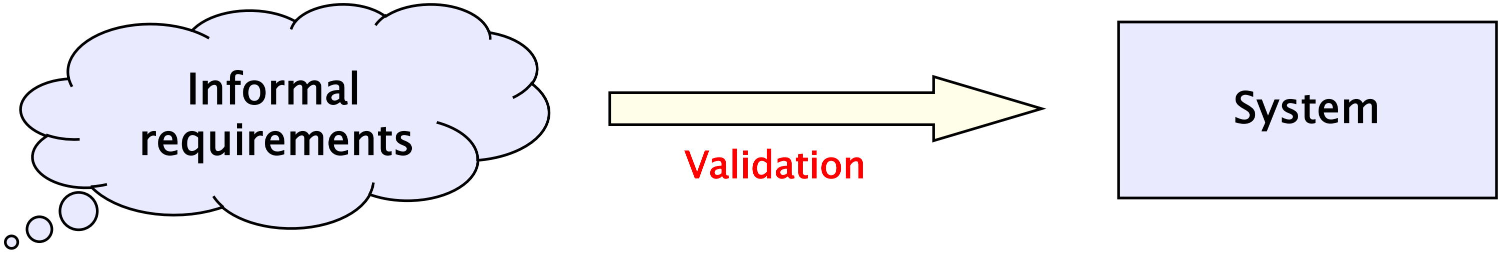
- Introducing probabilistic model checking…

- Topics for this lecture
  - the role of automatic verification
  - what is probabilistic model checking?
  - why is it important?
  - where is it applicable?
  - what does it involve?

- About this course
  - aims and organisation
  - information and links

Conventional software engineering

- From requirements to software system
  - apply design methodologies
  - code directly in programming language
  - validation via testing, code walkthroughs



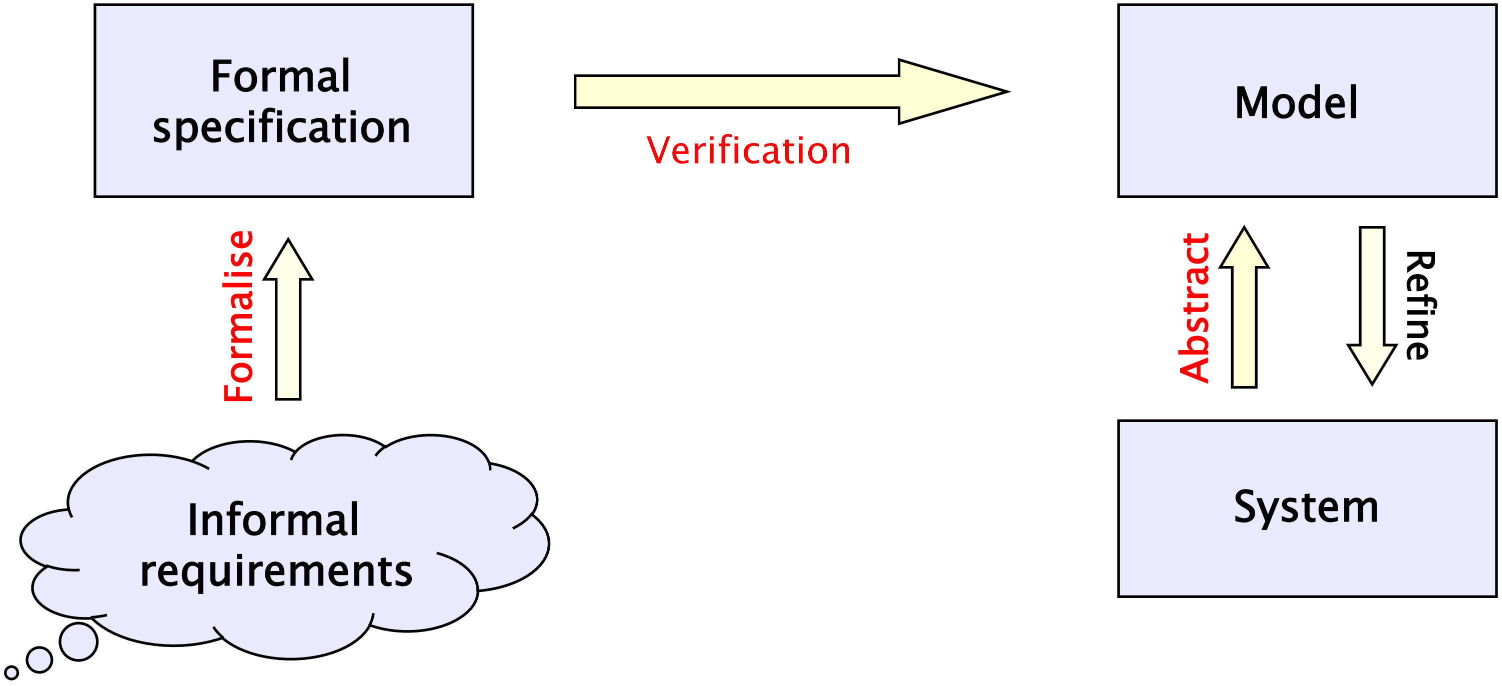
Informal requirements

Validation

System

Formal verification

- From requirements to formal specification
  - formalise specification, derive model
  - formally verify correctness



Formal specification

Verification

Model

Formalise

Abstract

Refine

Informal requirements

System

But my program works!

- True, there are many successful large-scale complex computer systems…
    - online banking, electronic commerce
    - information services, online libraries, business processes
    - supply chain management
    - mobile phone networks

- Yet many new potential application domains with far greater complexity and higher expectations
    - automotive drive-by-wire
    - medical sensors: heart rate & blood pressure monitors
    - intelligent buildings and spaces, environmental sensors

- Learning from mistakes costly…

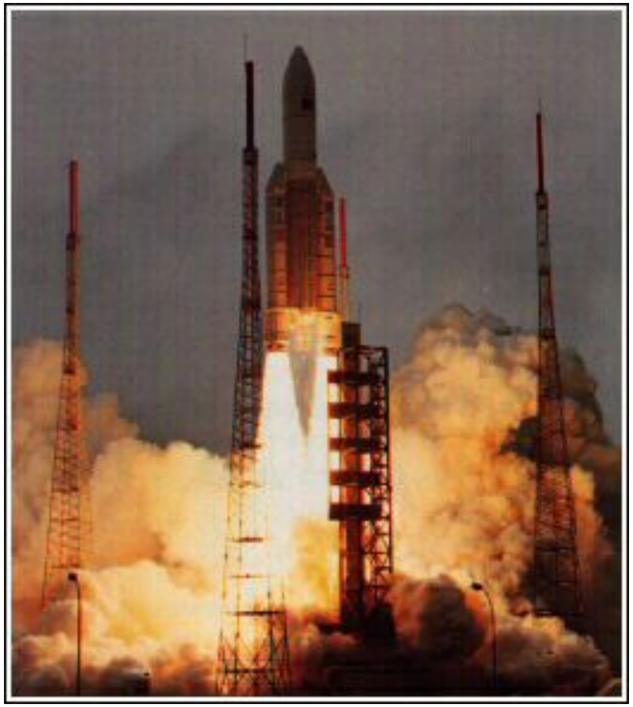
Toyota Prius

- Toyota Prius
  - first mass-produced hybrid vehicle
- February 2010
  - software “glitch” found in anti-lock braking system
  - in response to numerous complaints/accidents

- Eventually fixed via software update
  - in total 185,000 cars recalled, at huge cost
  - handling of the incident prompted much criticism, bad publicity

Ariane 5

- ESA (European Space Agency) Ariane 5 launcher
  - shown here in maiden flight on 4th June 1996



- 37secs later self-destructs
  - uncaught exception: numerical overflow in a conversion routine results in incorrect altitude sent by the on-board computer
- Expensive, embarrassing...

The London Ambulance Service

- London Ambulance Service computer aided despatch system
  - Area 600sq miles
  - Population 6.8million
  - 5000 patients per day
  - 2000-2500 calls per day
  - 1000-1200 999 calls per day

- Introduced October 1992
- Severe system failure:
  - position of vehicles incorrectly recorded
  - multiple vehicles sent to the same location
  - 20-30 people estimated to have died as a result

What do these stories have in common?

- Programmable computing devices
  - conventional computers and networks
  - software embedded in devices
    - airbag controllers, mobile phones, etc
- Programming error direct cause of failure

- Software critical
  - for safety
  - for business
  - for performance
- High costs incurred: not just financial

- Failures avoidable…

Why must we verify?

"Testing can only show the presence of errors, not their absence."

To rule out errors need to consider all possible executions often not feasible mechanically!

- need formal verification…

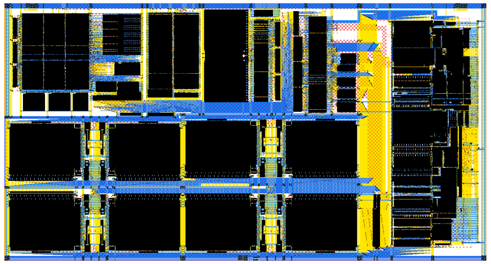
"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, computers are without precedent in the cultural history of mankind."

Edsger Dijkstra

1930-2002

Automatic verification

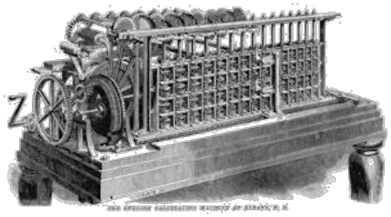
- Formal verification…
  - the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
  - essentially: proving that a program satisfies it specification
  - many techniques: manual proof, automated theorem proving, static analysis, model checking, …



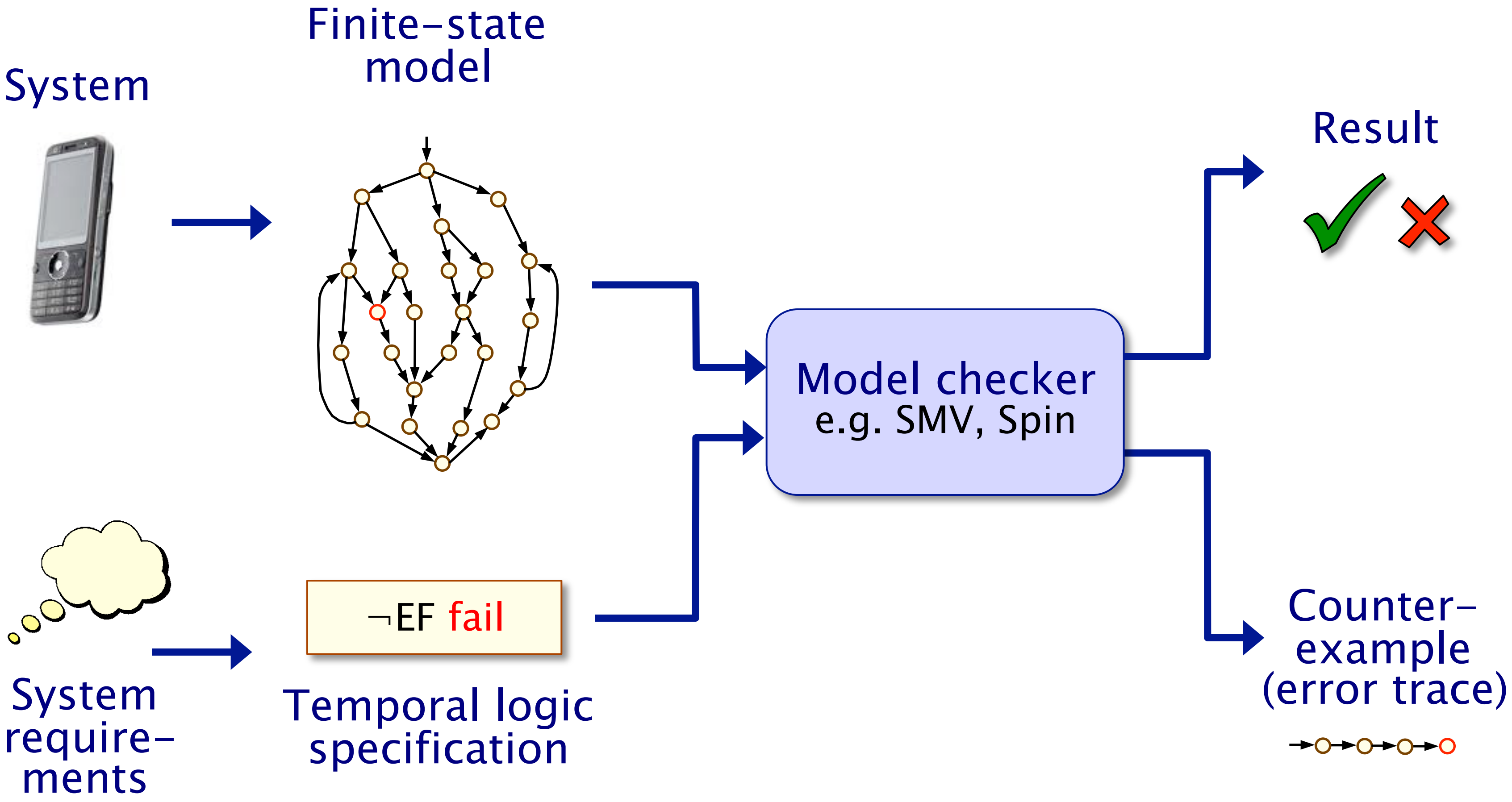
$10^{500,000}$ states

10^{70} atoms

- Automatic verification =
  - mechanical, push-button technology
  - performed without human intervention



Verification via model checking



System

Finite-state model

System require-ments

$\neg$EF fail

Temporal logic specification

Model checker
e.g. SMV, Spin

Result

Counter-example (error trace)

Model checking in practice

- Model checking now routinely applied to real-life systems
  - not just “verification”…
  - model checkers used as a debugging tool
  - at IBM, bugs detected in arbiter that could not be found with simulations
- Now widely accepted in industrial practice
  - Microsoft, Intel, Cadence, Bell Labs, IBM,...
- Many software tools, both commercial and academic
  - smv, SPIN, SLAM, FDR2, FormalCheck, RuleBase, ...
  - software, hardware, protocols, …
- Extremely active research area
  - 2008 Turing Award won by Edmund Clarke, Allen Emerson and Joseph Sifakis for their work on model checking

New challenges for verification

- Devices, ever smaller
    - laptops, phones, sensors…
- Networking, wireless, wired & global
    - wireless & internet everywhere
- New design and engineering challenges
    - adaptive computing, ubiquitous/pervasive computing, context-aware systems
    - trade-offs between e.g. performance, security, power usage, battery life, …

New challenges for verification

- Many properties other than correctness are important
- Need to guarantee…
  - safety, reliability, performance, dependability
  - resource usage, e.g. battery life
  - security, privacy, trust, anonymity, fairness
  - and much more…

- Quantitative, as well as qualitative requirements:
  - "how reliable is my car's Bluetooth network?"
  - "how efficient is my phone's power management policy?"
  - "how secure is my bank's web-service?"

- This course: probabilistic verification

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
    - IEEE 802.3 CSMA/CD, IEEE 802.11 Wireless LAN
  - Random choice of waiting time
    - IEEE 1394 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

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

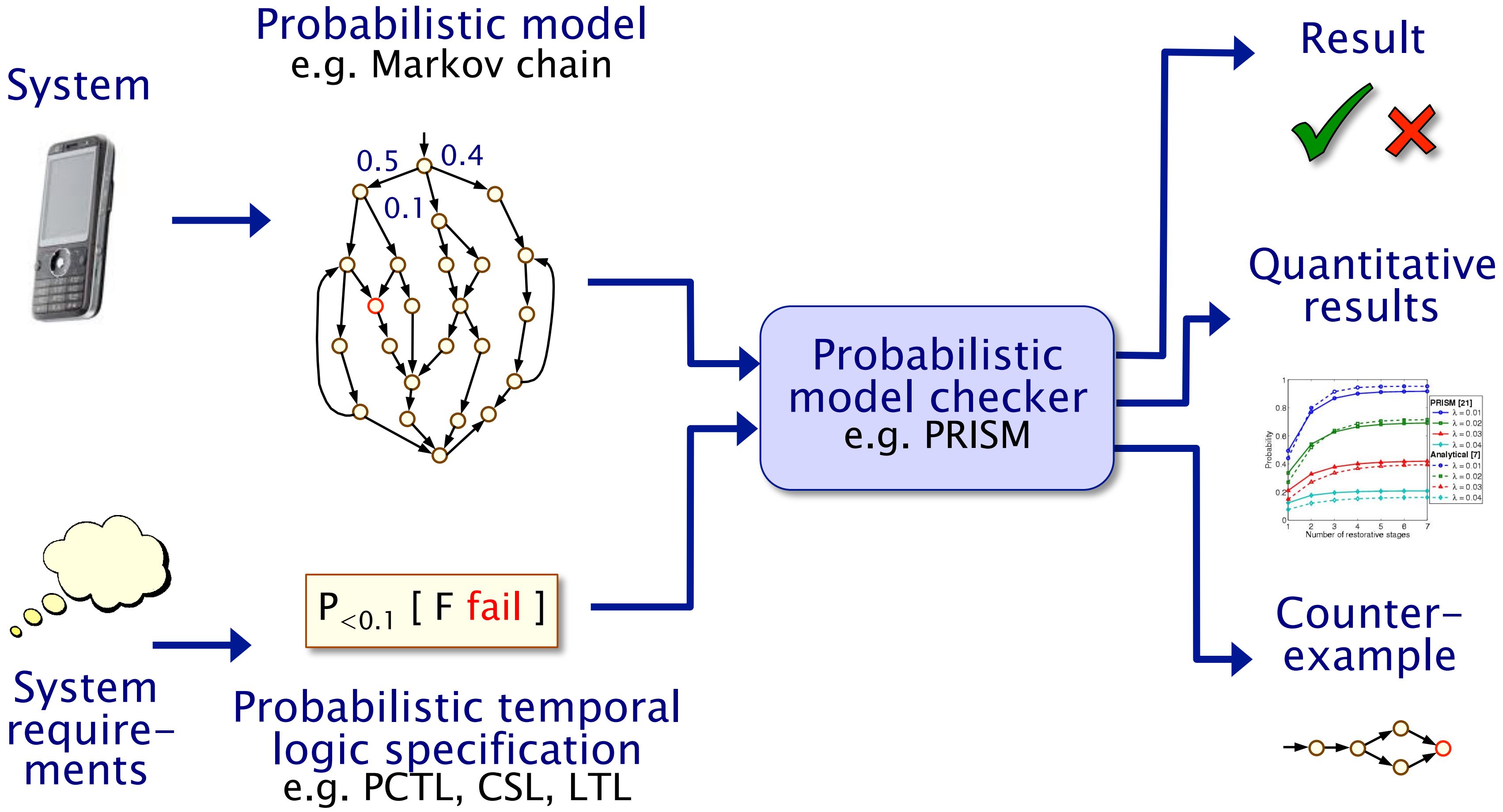
- Examples:
  - computer networks, embedded systems
  - power management policies
  - nano-scale circuitry: reliability through defect-tolerance

Why probability?

- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- Modelling uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- For quantitative analysis of software and systems
  - to quantify resource usage given a policy

    "the minimum expected battery capacity for a scenario…"
- And many others, e.g. biological processes

Probabilistic model checking



System

Probabilistic model
e.g. Markov chain

Probabilistic
model checker
e.g. PRISM

Result

Quantitative
results

Counter-
example

System
require-
ments

$P_{<0.1}$ [F fail]

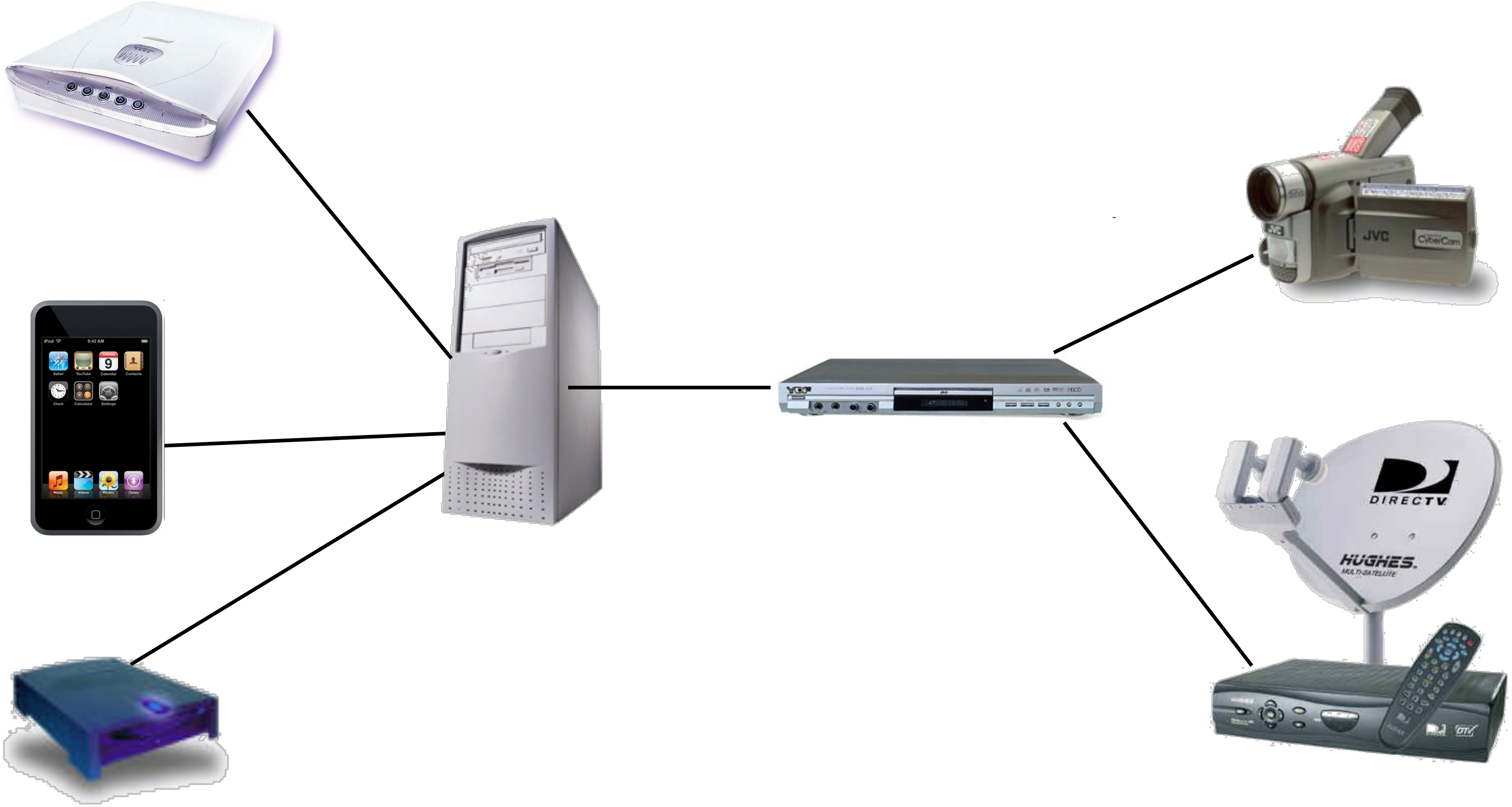
Probabilistic temporal
logic specification
e.g. PCTL, CSL, LTL

Case study: FireWire protocol

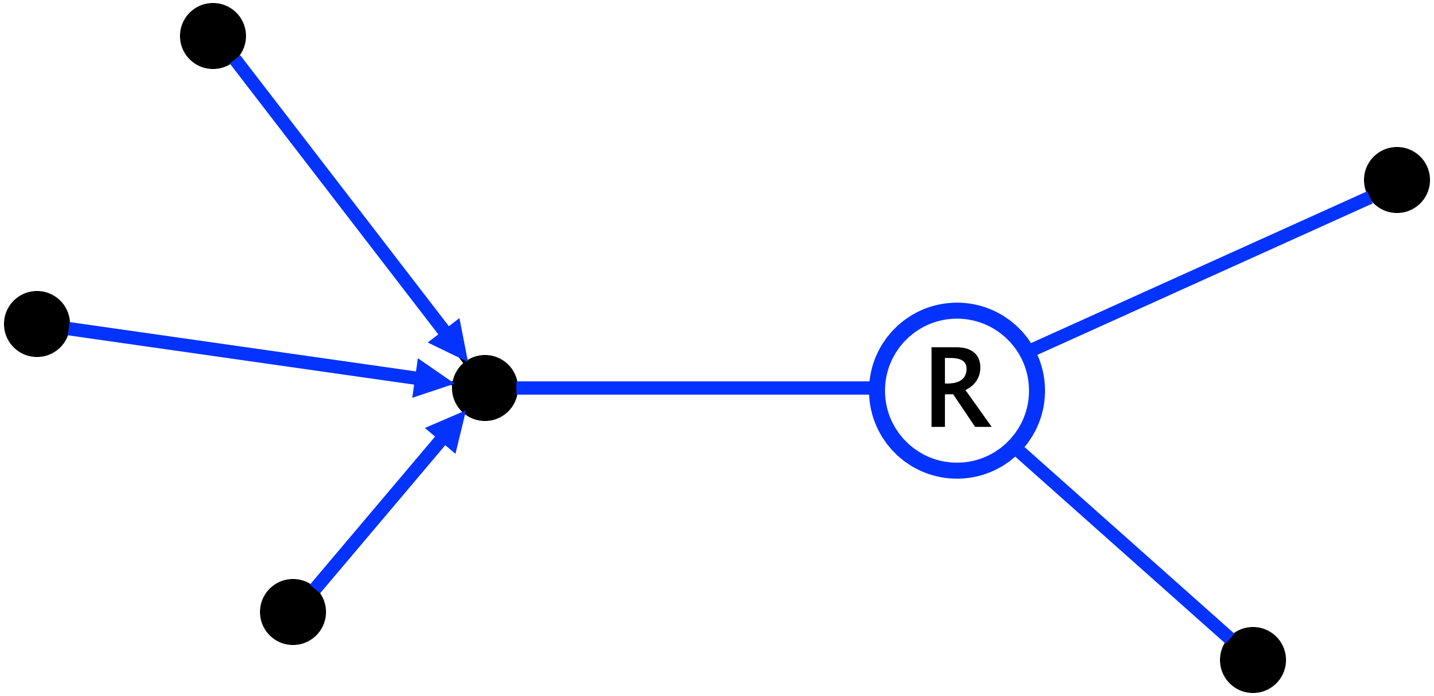
- FireWire (IEEE 1394)
  - high-performance serial bus for networking multimedia devices; originally by Apple
  - "hot-pluggable" - add/remove devices at any time
  - no requirement for a single PC (need acyclic topology)

- 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

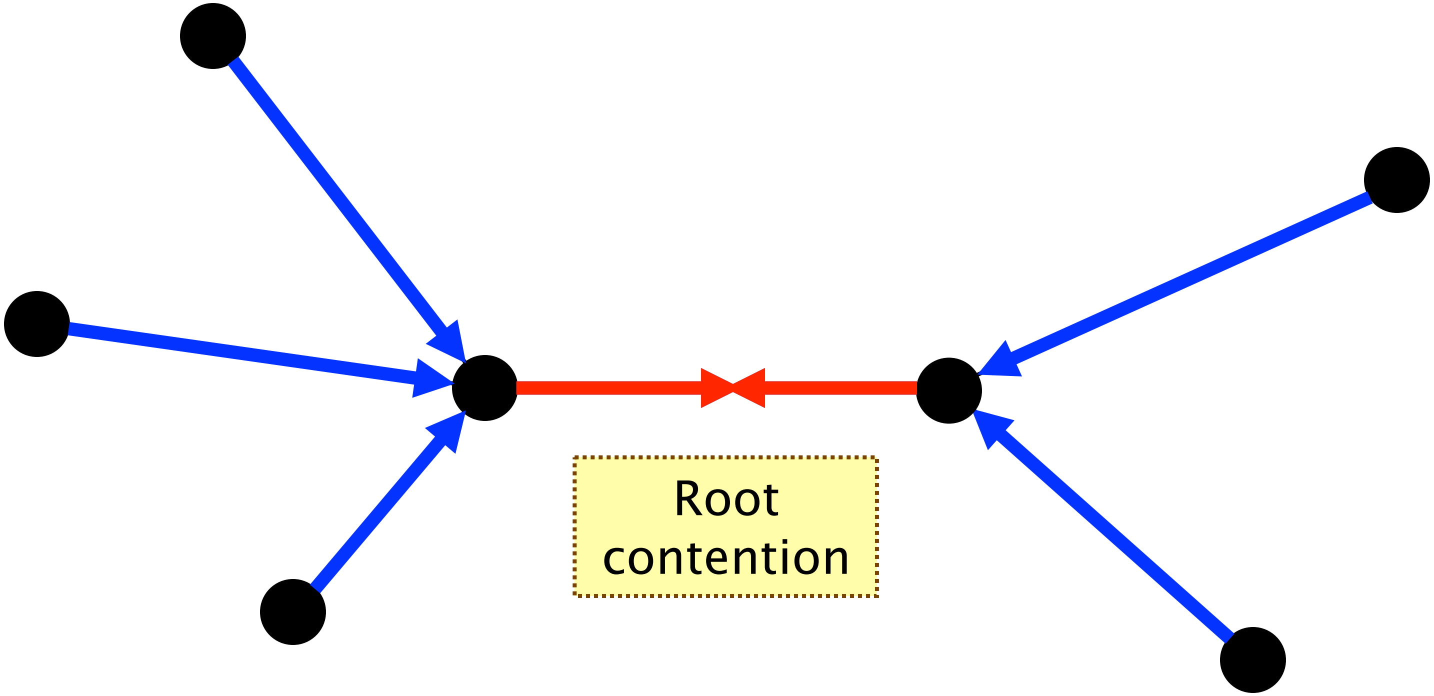
FireWire example



FireWire leader election

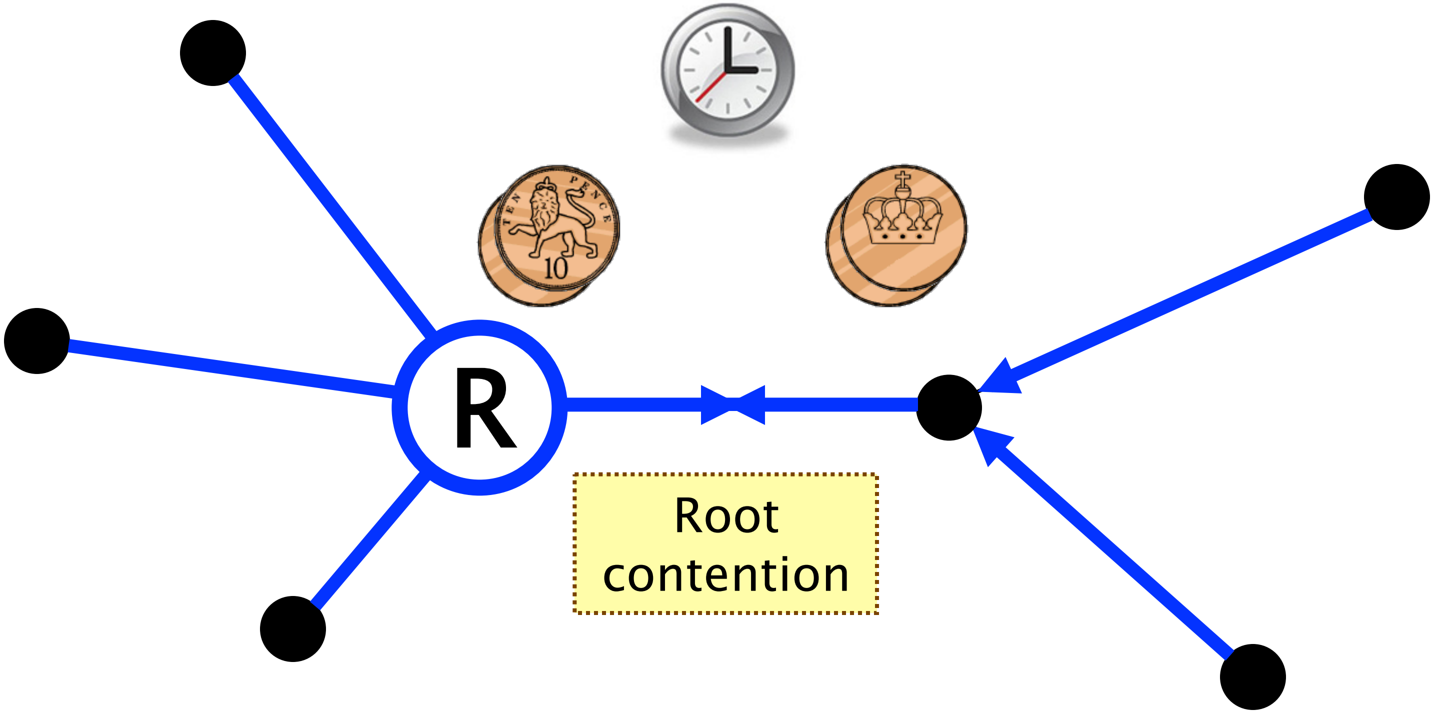


FireWire root contention



Root contention

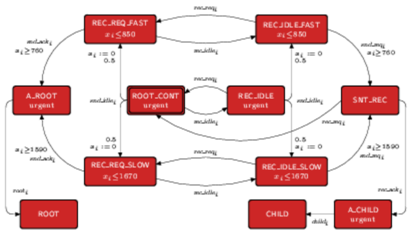
FireWire root contention

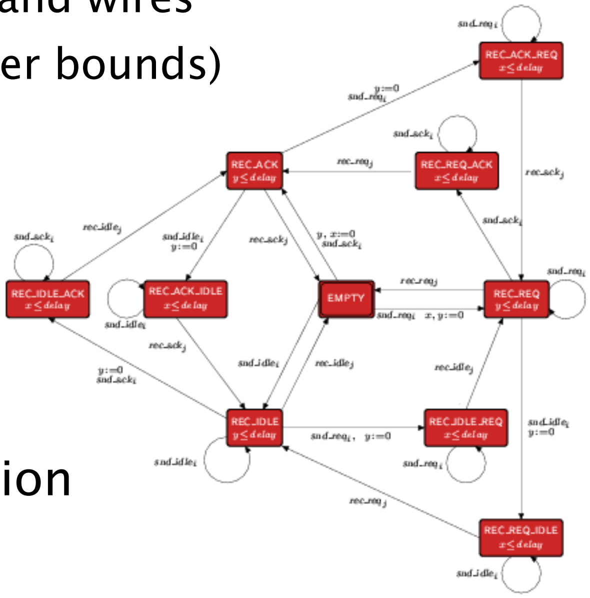


Root contention

FireWire analysis

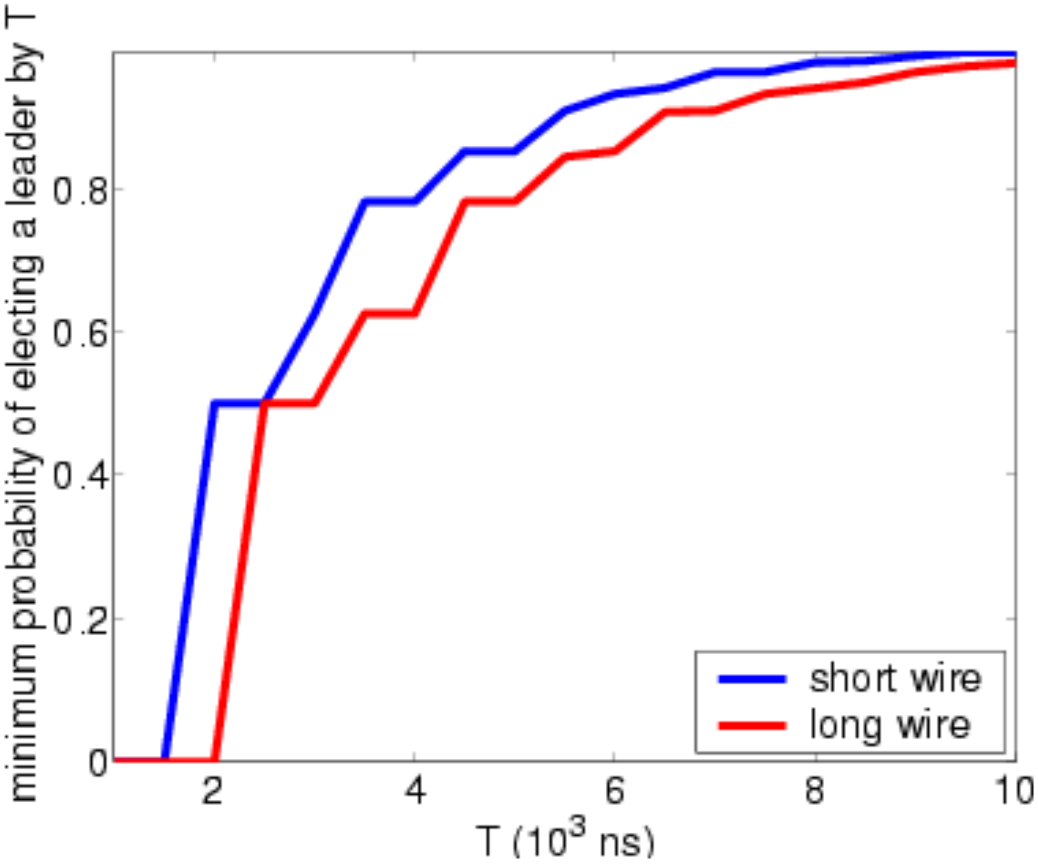
- Probabilistic model checking
  - model constructed and analysed using PRISM
  - timing delays taken from IEEE standard
  - model includes:
    - concurrency: messages between nodes and wires
    - underspecification of delays (upper/lower bounds)
  - max. model size: 170 million states





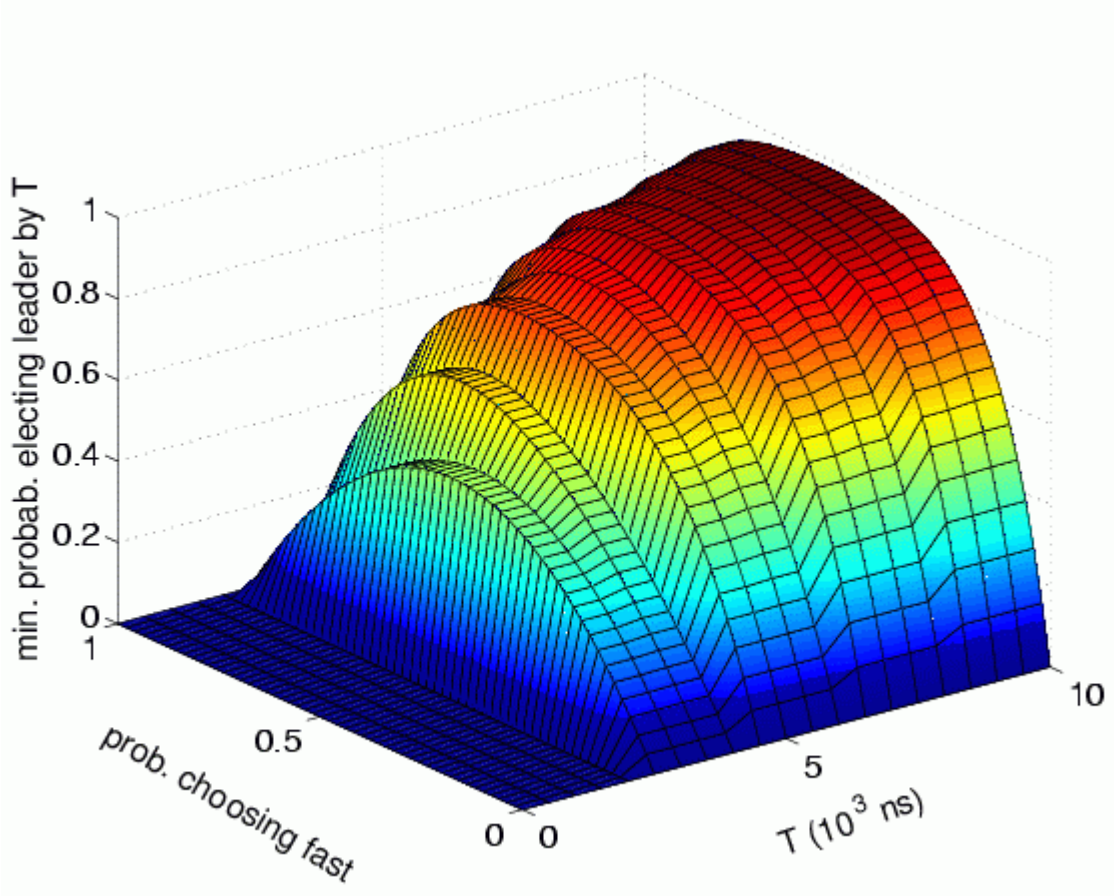
- Analysis:
  - verified that root contention always resolved with probability 1
  - investigated time taken for leader election
  - and the effect of using biased coin
    - based on a conjecture by Stoelinga

FireWire: Analysis results



"minimum probability of electing leader by time T"

FireWire: Analysis results

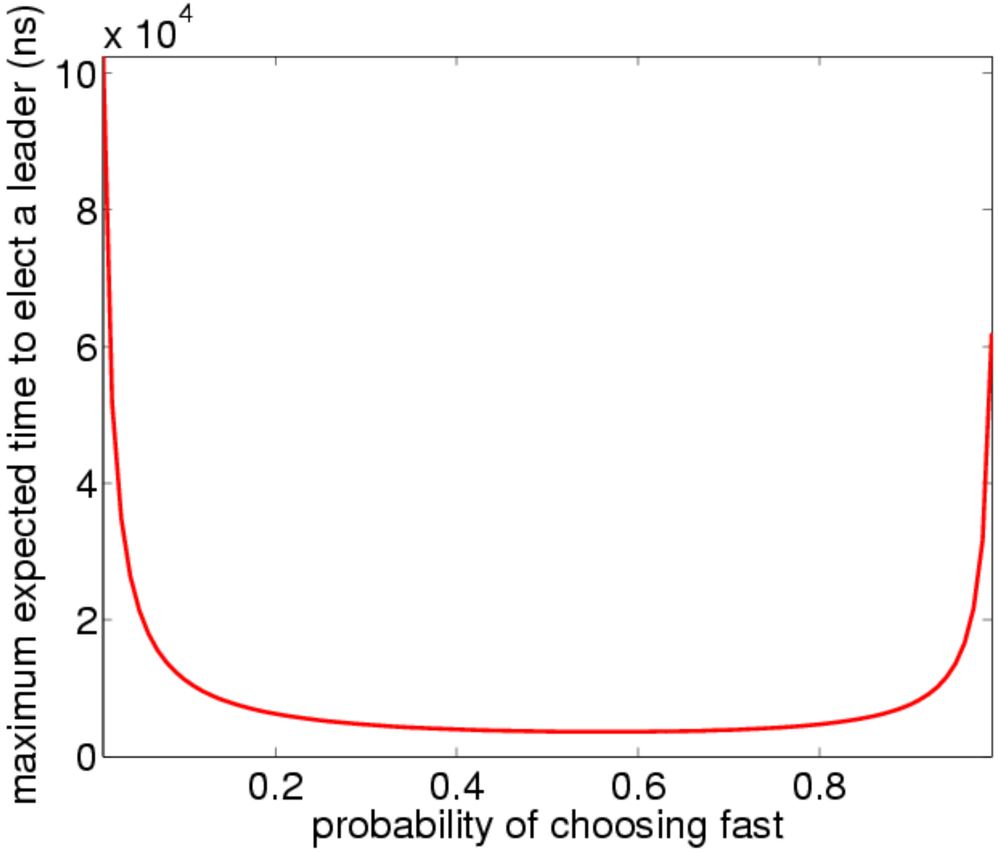


“minimum probability of electing leader by time T”

(short wire length)

Using a biased coin

FireWire: Analysis results

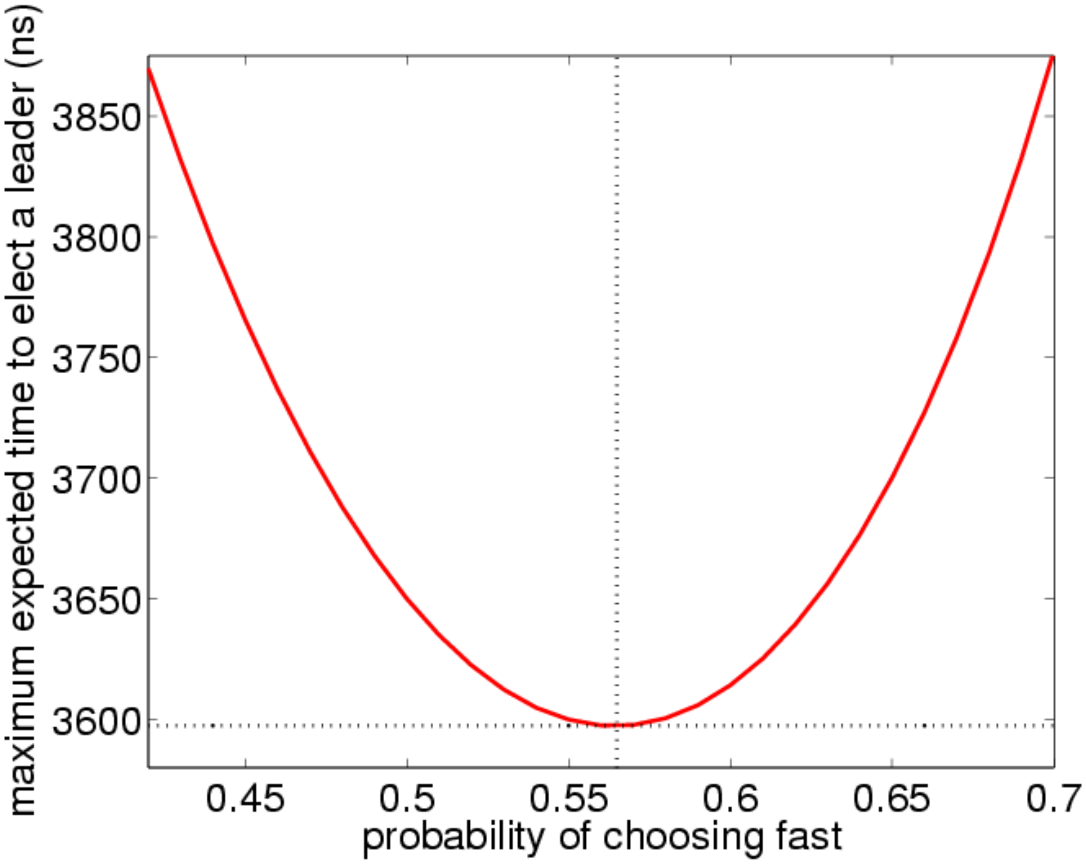


"maximum expected time to elect a leader"

(short wire length)

Using a biased coin

FireWire: Analysis results

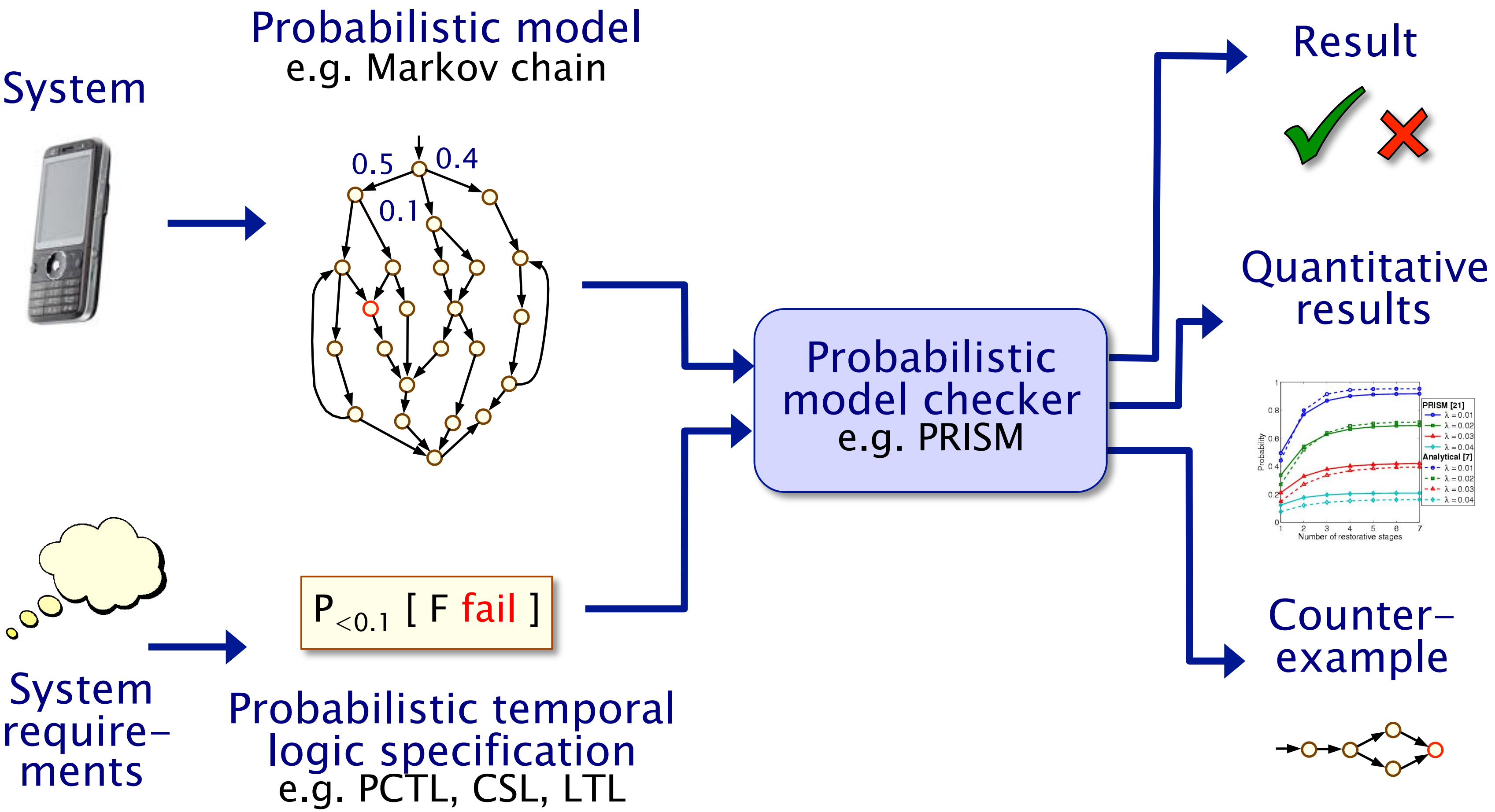


"maximum expected time to elect a leader"

(short wire length)

Using a biased coin is beneficial!

Probabilistic model checking



System

Probabilistic model
e.g. Markov chain

System require-ments

$P_{<0.1}$ [F fail]

Probabilistic temporal logic specification
e.g. PCTL, CSL, LTL

Probabilistic model checker
e.g. PRISM

Result

Quantitative results

Counter-example

Probabilistic model checking inputs

- Models: variants of Markov chains
  - discrete-time Markov chains (DTMCs)
    - discrete time, discrete probabilistic behaviours only
  - continuous-time Markov chains (CTMCs)
    - continuous time, continuous probabilistic behaviours
  - Markov decision processes (MDPs)
    - DTMCs, plus nondeterminism
- Specifications
  - informally:
    - "probability of delivery within time deadline is …"
    - "expected time until message delivery is …"
    - "expected power consumption is …"
  - formally:
    - probabilistic temporal logics (PCTL, CSL, LTL, PCTL*, …)
    - e.g. $P_{<0.05}$ [F err/total>0.1], $P_{=?}$ [$F^{\leq t}$ reply_count=k]

Probabilistic model checking involves…

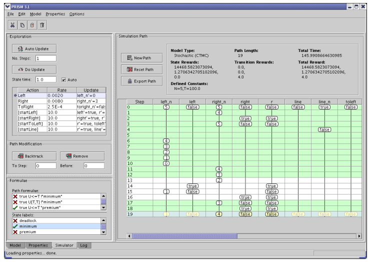
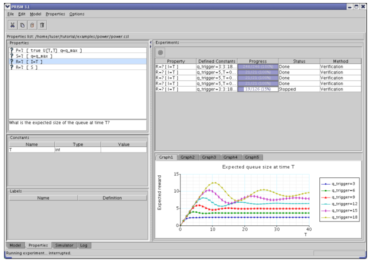
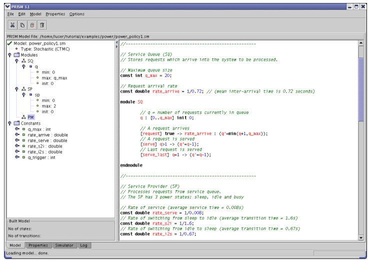
- Construction of models
  - from a description in a high-level modelling language

- Probabilistic model checking algorithms
  - graph-theoretical algorithms
    - e.g. for reachability, identifying strongly connected components
  - numerical computation
    - linear equation systems, linear optimisation problems
    - iterative methods, direct methods
    - uniformisation, shortest path problems
  - automata for regular languages
  - also sampling-based (statistical) for approximate analysis
    - e.g. hypothesis testing based on simulation runs

Probabilistic model checking involves…

- Efficient implementation techniques
  - essential for scalability to real-life systems
  - symbolic data structures based on binary decision diagrams
  - algorithms for bisimulation minimisation, symmetry reduction

- Tool support
  - PRISM: free, open-source probabilistic model checker
  - currently based at Oxford University
  - supports all probabilistic models discussed here

Course aims

- Introduce main types of probabilistic models and specification notations
  - theory, syntax, semantics, examples
  - probability, expectation, costs/rewards
- Explain the working of probabilistic model checking
  - algorithms & (symbolic) implementation
- Introduce software tools
  - probabilistic model checker PRISM
- Examples from wide range of application domains
  - communication & coordination protocols, performance & reliability modelling, biological systems, …

- Mix of theory and practice

Course outline

- Discrete-time Markov chains (DTMCs) and their properties
- Probabilistic temporal logics: PCTL, LTL, etc.
- PCTL model checking for DTMCs
- The PRISM model checker
- Costs & rewards
- Continuous-time Markov chains (CTMCs)
- Counterexamples & bisimulation
- Markov decision processes (MDPs)
- Probabilistic LTL model checking
- Implementation and data structures: symbolic techniques

Course information

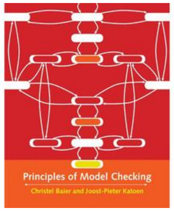
- Prerequisites/background
  - basic computer science/maths background
  - no probability knowledge assumed

- Lectures
  - 20 lectures: Mon 2pm, Wed 3pm, Thur 12pm (wks 1-4)
- Classes/practicals (please sign up on-line)
  - 4 problem sheets + 1 hr classes
    (Tue 3pm, Wed 12pm, wks 3, 5, 7, 8)
  - 4 practical exercises, based on PRISM,
    4 scheduled 2 hr practical sessions (Tue 4pm, wks 3, 4, 6, 7),
    + work outside lab sessions
- Assessment
  - take-home assignment

Further information

- Course lecture notes are self-contained
  - www.cs.ox.ac.uk/teaching/materials11-12/probabilistic/

- For further reading material…
  - two online tutorial papers also cover a lot of the material
    - Stochastic Model Checking
      Marta Kwiatkowska, Gethin Norman and David Parker
    - Automated Verification Techniques for Probabilistic Systems
      Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, David Parker
  - DTMC/MDP material also based on Chapter 10 of:



Principles of Model Checking
Christel Baier and Joost-Pieter Katoen
MIT Press

  - PRISM web site: http://www.prismmodelchecker.org/

Next lecture(s)

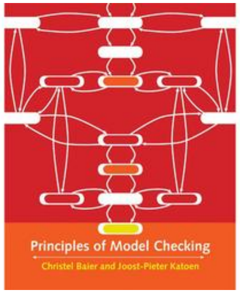
- Wed 3pm
- Thur 12pm

- Discrete-time Markov chains

Acknowledgements

- Much of the material in the course is based on an existing lecture course prepared by:
  - Marta Kwiatkowska
  - Gethin Norman
  - Dave Parker

- Various material and examples also appear courtesy of:
  - Christel Baier
  - Joost-Pieter Katoen



Lecture 2
Discrete-time Markov Chains

Dr. Dave Parker

Department of Computer Science
University of Oxford

Probabilistic Model Checking

- Formal verification and analysis of systems that exhibit probabilistic behaviour
  - e.g. randomised algorithms/protocols
  - e.g. systems with failures/unreliability

- Based on the construction and analysis of precise mathematical models

- This lecture: discrete-time Markov chains

Overview

- Probability basics
- Discrete-time Markov chains (DTMCs)
  - definition, properties, examples
- Formalising path-based properties of DTMCs
  - probability space over infinite paths
- Probabilistic reachability
  - definition, computation
- Sources/further reading: Section 10.1 of [BK08]

Probability basics

- First, need an experiment
  - The sample space Ω is the set of possible outcomes
  - An event is a subset of Ω, can form events $A \cap B$, $A \cup B$, $\Omega \setminus A$

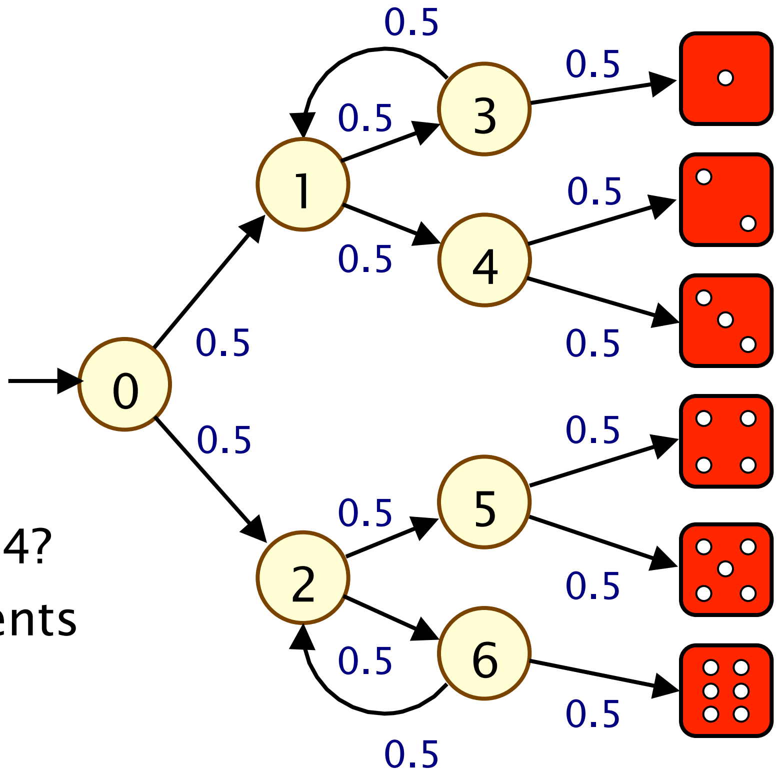
- Examples:

– toss a coin:	$\Omega = \{H,T\}$, events: "H", "T"
– toss two coins:	$\Omega = \{(H,H),(H,T),(T,H),(T,T)\}$, event: "at least one H"
– toss a coin ∞-often:	Ω is set of infinite sequences of H/T event: "H in the first 3 throws"

- Probability is:
  - Pr("H") = Pr("T") = 1/2,   Pr("at least one H") = 3/4
  - Pr("H in the first 3 throws") = 1/2 + 1/4 + 1/8 = 7/8

Probability example

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

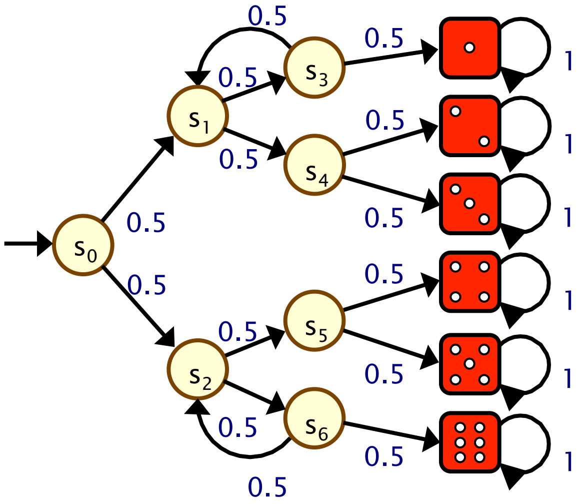


- Is this algorithm correct?
  - e.g. probability of obtaining a 4?
  - Obtain as disjoint union of events
  - THH, TTTHH, TTTTTHH, …
  - Pr("eventually 4")

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

Example…

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

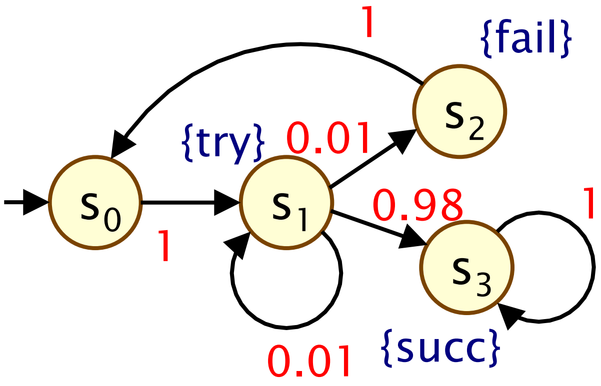


- Probabilistic model checking provides a framework for these kinds of properties…
  - modelling languages
  - property specification languages
  - model checking algorithms, techniques and tools

Discrete-time Markov chains

- State-transition systems augmented with probabilities

- States
  - set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states model evolution of system's state; occur in discrete time-steps
- Probabilities
  - probabilities of making transitions between states are given by discrete probability distributions



Markov property

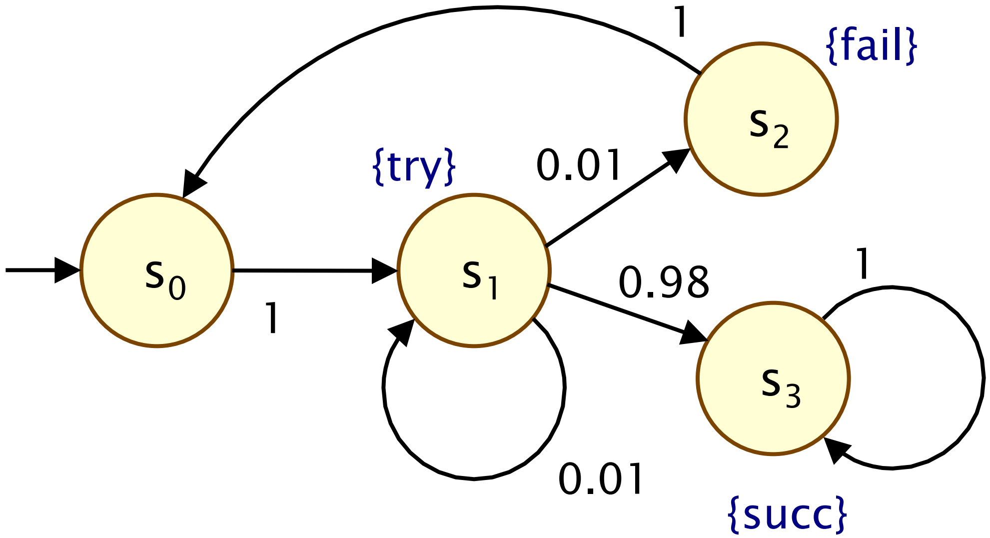
- If the current state is known, then the future states of the system are independent of its past states

- i.e. the current state of the model contains all information that can influence the future evolution of the system

- also known as “memorylessness”

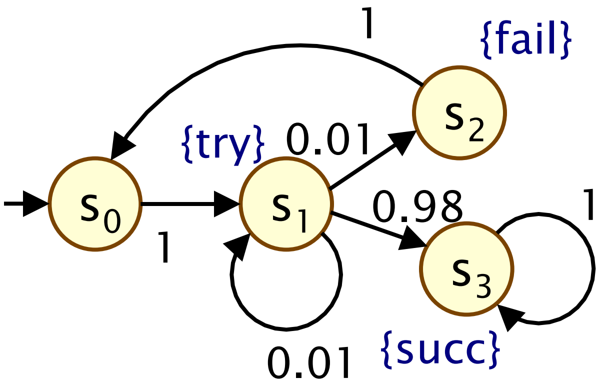
Simple DTMC example

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



Discrete-time Markov chains

- Formally, a DTMC D is a tuple $(S,s_{init},\mathbf{P},L)$ where:
  - S is a 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 (taken from a set AP)



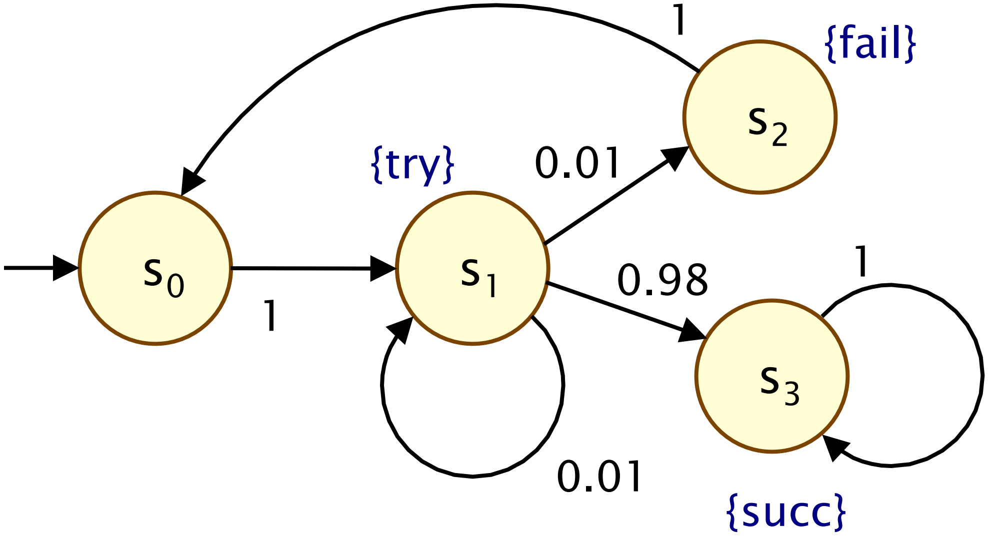
Simple DTMC example

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

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

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

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



Some more terminology

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

DTMCs: An alternative definition

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

Other assumptions made here

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

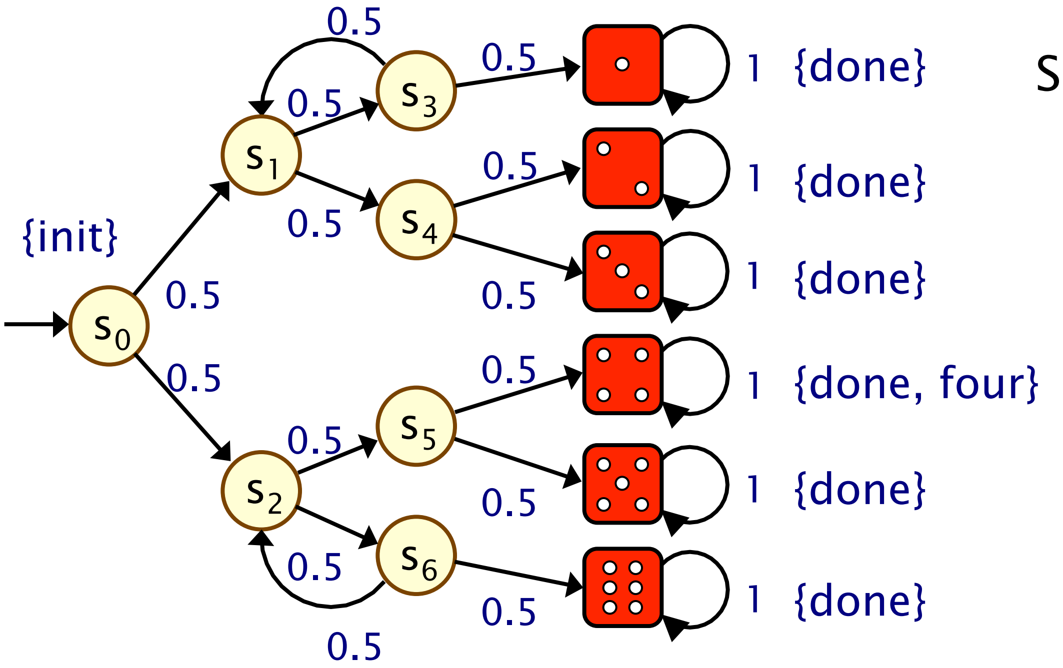
- We will (mostly) assume that the state space S is finite
  - in general, S can be any countable set

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

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

DTMC example 2 – Coins and dice

- Recall Knuth/Yao's die algorithm from earlier:



$S = \{ s_0, s_1, \ldots, s_6, 1, 2, \ldots, 6 \}$

$s_{init} = s_0$

$P(s_0,s_1)=0.5$
$P(s_0,s_2)=0.5$
etc.

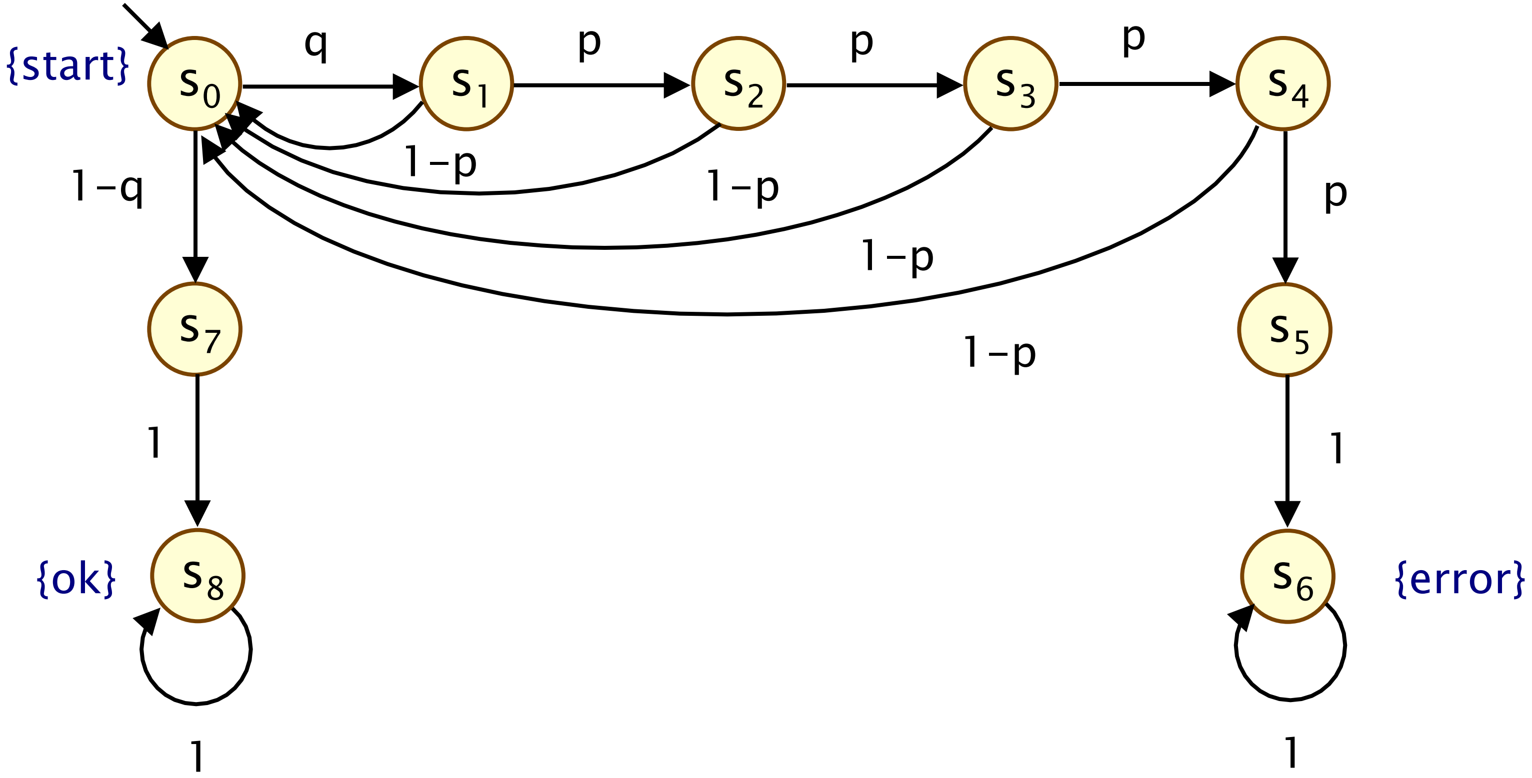
$L(s_0) = \{init\}$
etc.

DTMC example 3 – Zeroconf

- Zeroconf = "Zero configuration networking"
  - self-configuration for local, ad-hoc networks
  - automatic configuration of unique IP for new devices
  - simple; no DHCP, DNS, …
- Basic idea:
  - 65,024 available IP addresses (IANA-specified range)
  - new node picks address U at random
  - broadcasts "probe" messages: "Who is using U?"
  - a node already using U replies to the probe
  - in this case, protocol is restarted
  - messages may not get sent (transmission fails, host busy, …)
  - so: nodes send multiple (n) probes, waiting after each one

DTMC for Zeroconf

- $n=4$ probes, m existing nodes in network
- probability of message loss: p
- probability that new address is in use: $q = m/65024$



Properties of DTMCs

- Path-based properties
  - what is the probability of observing a particular behaviour (or class of behaviours)?
  - e.g. “what is the probability of throwing a 4?”

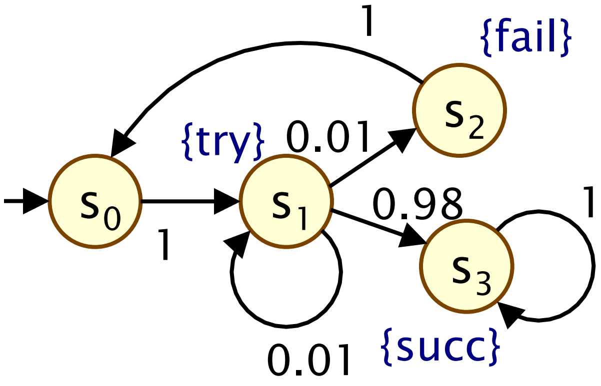
- Transient properties
  - probability of being in state s after t steps?

- Steady-state
  - long-run probability of being in each state

- Expectations
  - e.g. “what is the average number of coin tosses required?”

DTMCs and paths

- A path in a DTMC represents an execution (i.e. one possible behaviour) of the system being modelled
- Formally:
  - infinite sequence of states $s_0 s_1 s_2 s_3 \ldots$ such that $P(s_i, s_{i+1}) > 0 \;\forall i \geq 0$
  - infinite unfolding of DTMC

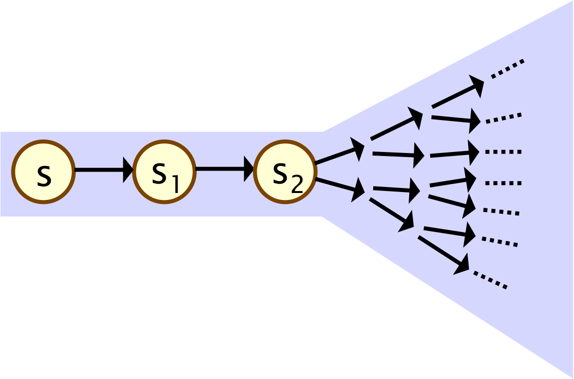


- Examples:
  - never succeeds: $(s_0 s_1 s_2)^\omega$
  - tries, waits, fails, retries, succeeds: $s_0 s_1 s_1 s_2 s_0 s_1 (s_3)^\omega$
- Notation:
  - Path(s) = set of all infinite paths starting in state s
  - also sometimes use finite (length) paths
  - $\mathrm{Path}_{\mathrm{fin}}(s)$ = set of all finite paths starting in state s

Paths and probabilities

- 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 Cyl(ω), for a finite path ω = set of infinite paths with the common finite prefix ω
  - for example: Cyl(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 ∅ is in Σ

- Elements of Σ are called measurable sets or events

- Theorem: For any family F of subsets of Ω, there exists a unique smallest σ–algebra on Ω containing F

Probability spaces

- 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 – Simple example

- Sample space Ω
  - Ω = {1,2,3}

- Event set Σ
  - e.g. powerset of Ω
  - Σ = { ∅, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3} }
  - (closed under complement/countable union, contains ∅)

- Probability measure Pr
  - e.g. Pr(1) = Pr(2) = Pr(3) = 1/3
  - Pr({1,2}) = 1/3+1/3 = 2/3, etc.

Probability space – Simple example 2

- Sample space Ω
  - $\Omega = \mathbb{N} = \{ 0,1,2,3,4,\ldots \}$
- Event set Σ
  - e.g. $\Sigma = \{ \emptyset$, “odd”, “even”, $\mathbb{N} \}$
  - (closed under complement/countable union, contains ∅)
- Probability measure Pr
  - e.g. Pr(“odd”) = 0.5, Pr(“even”) = 0.5

Probability space over paths

- Sample space Ω = Path(s)

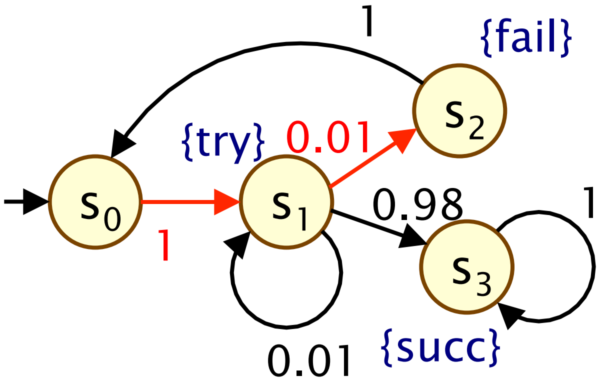
  set of infinite paths with initial state s
- Event set $\Sigma_{Path(s)}$
  - the cylinder set Cyl(ω) = { ω' ∈ Path(s) | ω is prefix of ω' }
  - $\Sigma_{Path(s)}$ is the least σ-algebra on Path(s) containing Cyl(ω) for all finite paths ω starting in s
- Probability measure Pr_s
  - define probability $\mathbf{P}_s(\omega)$ for finite path $\omega = s s_1 \ldots s_n$ 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 $Pr_s(Cyl(\omega)) = \mathbf{P}_s(\omega)$ 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

Paths and probabilities – Example

- Paths where sending fails immediately
  - $\omega = s_0 s_1 s_2$
  - $Cyl(\omega)$ = all paths starting $s_0 s_1 s_2 \ldots$
  - $P_{s0}(\omega) = P(s_0,s_1) \cdot P(s_1,s_2)$

    $= 1 \cdot 0.01 = 0.01$
  - $Pr_{s0}(Cyl(\omega)) = P_{s0}(\omega) = 0.01$



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

    $= P_{s0}(s_0 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_1 s_3) + \ldots$

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

    $= 0.9898989898\ldots$

    $= 98/99$

Reachability

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

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

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

Reachability probabilities

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

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

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

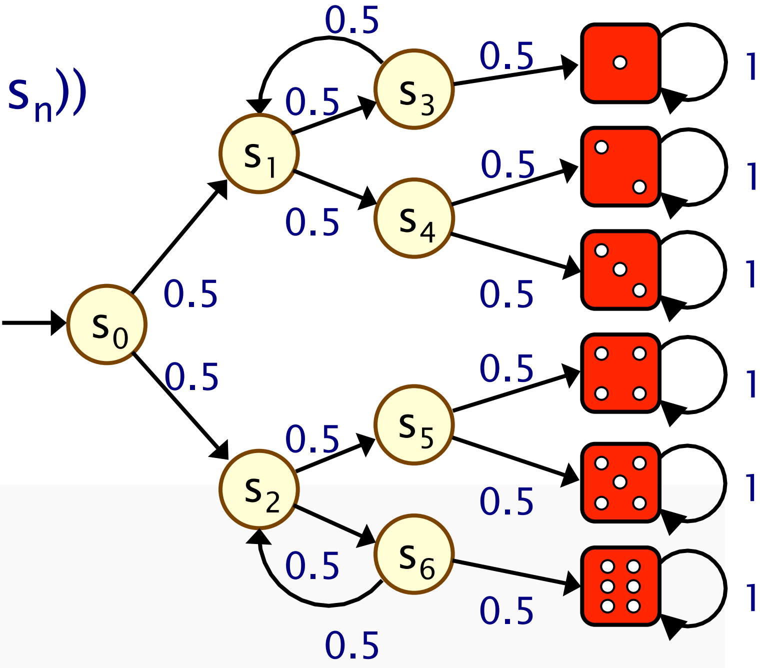
Computing reachability probabilities

- Compute as (infinite) sum…

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

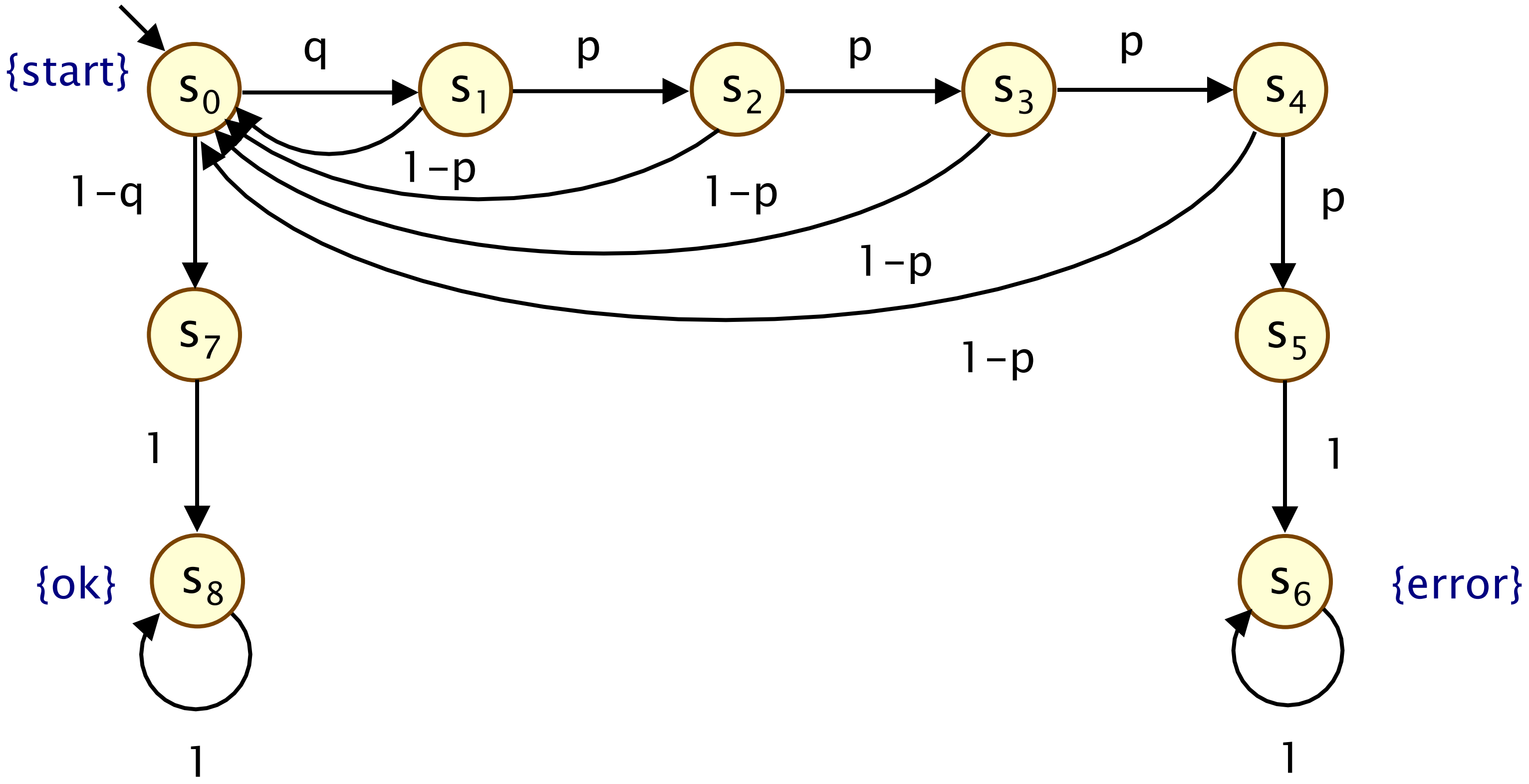
  $= \Sigma_{s_0,\ldots,s_n \in \text{Reachfin}(s,\,T)} \mathbf{P}(s_0,\ldots,s_n)$

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



Computing reachability probabilities

- ProbReach(s_0, {s_6}) : compute as infinite sum?
  - doesn't scale…



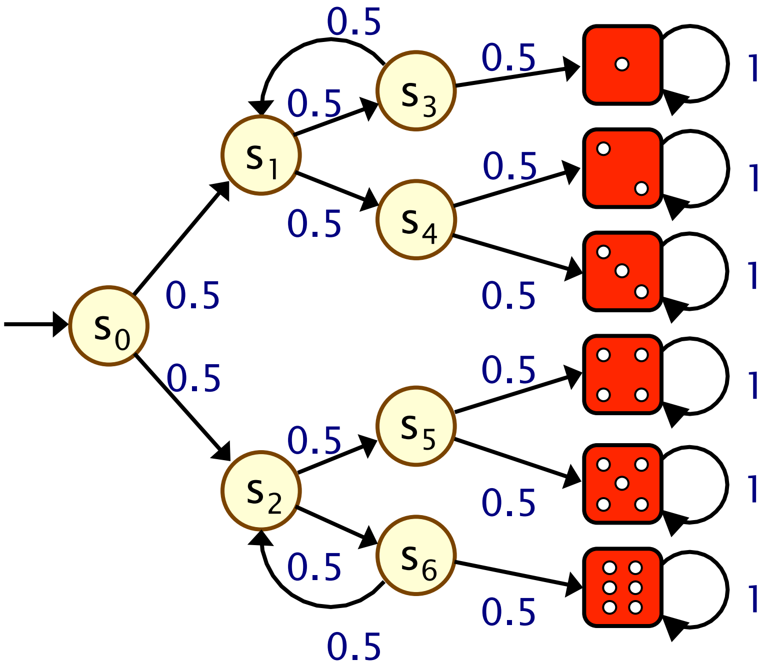
Computing reachability probabilities

- Alternative: derive a linear equation system
  - solve for all states simultaneously
  - i.e. compute vector ProbReach(T)
- Let x_s denote ProbReach(s, T)
- Solve:

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

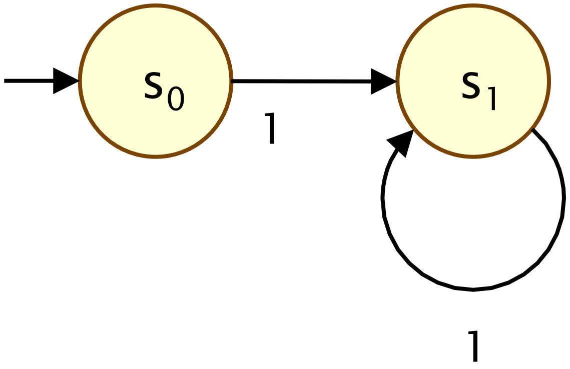
Example

- Compute ProbReach(s_0, {4})



Unique solutions

- Why the need to identify states that cannot reach T?
- Consider this simple DTMC:
  - compute probability of reaching $\{s_0\}$ from s_1



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

Computing reachability probabilities

- Another alternative: least fixed point characterisation

- Consider functions of the form:
  - $F : [0,1]^S \rightarrow [0,1]^S$

vectors of probabilities for each state

- And define:
  - $\underline{y} \leq \underline{y}'$ iff $\underline{y}(s) \leq \underline{y}'(s)$ for all s

- $\underline{y}$ is a fixed point of F if $F(\underline{y}) = \underline{y}$

- A fixed point $\underline{x}$ of F is the least fixed point of F if $\underline{x} \leq \underline{y}$ for any other fixed point $\underline{y}$

Least fixed point

- $\underline{\text{ProbReach}}(T)$ is the least fixed point of the function F:

$$F(\underline{y})(s) \quad = \quad \begin{cases} 1 & \text{if } s \in T \\ \sum_{s' \in S} P(s,s') \cdot \underline{y}(s') & \text{otherwise.} \end{cases}$$

- This yields a simple iterative algorithm to approximate $\underline{\text{ProbReach}}(T)$:
  - $\underline{x}^{(0)} = \underline{0}$ (i.e. $\underline{x}^{(0)}(s) = 0$ for all s)
  - $\underline{x}^{(n+1)} = F(\underline{x}^{(n)})$
  - $\underline{x}^{(0)} \leq \underline{x}^{(1)} \leq \underline{x}^{(2)} \leq \underline{x}^{(3)} \leq \ldots$
  - $\underline{\text{ProbReach}}(T) = \lim_{n \to \infty} \underline{x}^{(n)}$

in practice, terminate when for example:

$$\max_s \left| \underline{x}^{(n+1)}(s) - \underline{x}^{(n)}(s)) \right| < \varepsilon$$

for some user-defined tolerance value ε

Least fixed point

- Expressing ProbReach as a least fixed point…
  - corresponds to solving the linear equation system using the power method
    - other iterative methods exist (see later)
    - power method is guaranteed to converge
  - generalises non-probabilistic reachability
  - can be generalised to:
    - constrained reachability (see PCTL “until”)
    - reachability for Markov decision processes
  - also yields bounded reachability probabilities…

Bounded reachability probabilities

- Probability of reaching T from s within k steps

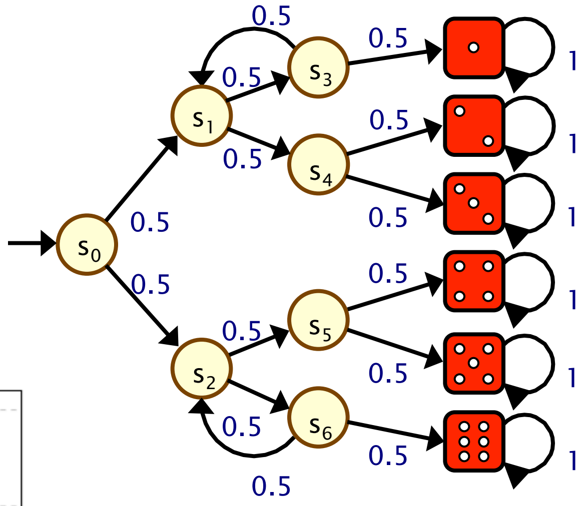
- Formally: ProbReach$^{\leq k}$(s, T) = Pr$_s$(Reach$^{\leq k}$(s, T)) where:
  - Reach$^{\leq k}$(s, T) = { $s_0 s_1 s_2 \ldots \in$ Path(s) | s_i in T for some i≤k }

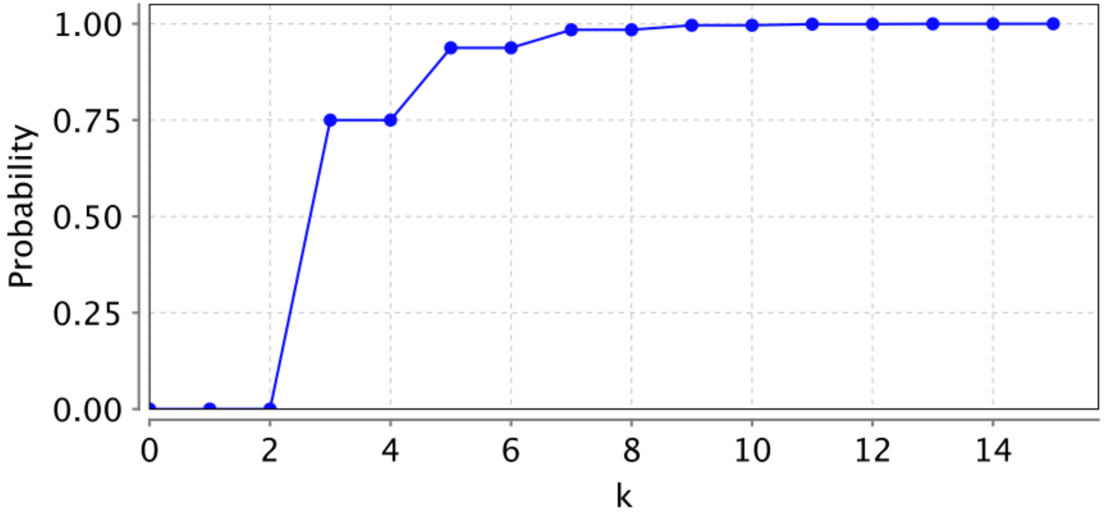
- ProbReach$^{\leq k}$(T) = $\underline{x}^{(k+1)}$ from the previous fixed point
  - which gives us…

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

(Bounded) reachability

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





Summing up…

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

- Formalising path-based properties of DTMCs
  - probability space over infinite paths

- Probabilistic reachability
  - infinite sum
  - linear equation system
  - least fixed point characterisation
  - bounded reachability

Next lecture

- Thur 12pm

- Discrete-time Markov chains…
  - transient
  - steady-state
  - long-run behaviour

Lecture 3
Discrete-time Markov Chains…

Dr. Dave Parker

Department of Computer Science
University of Oxford

Next few lectures…

- Today:
  - Discrete-time Markov chains (continued)

- Mon 2pm:
  - Probabilistic temporal logics

- Wed 3pm:
  - PCTL model checking for DTMCs

- Thur 12pm:
  - PRISM

Overview

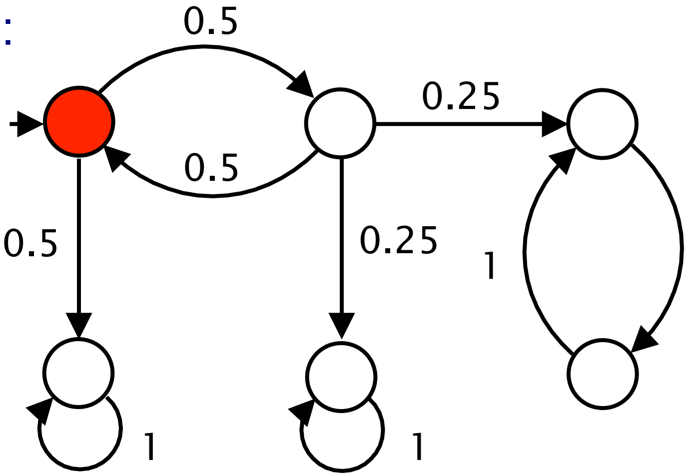
- Transient state probabilities
- Long-run / steady-state probabilities
- Qualitative properties
  - repeated reachability
  - persistence

Transient state probabilities

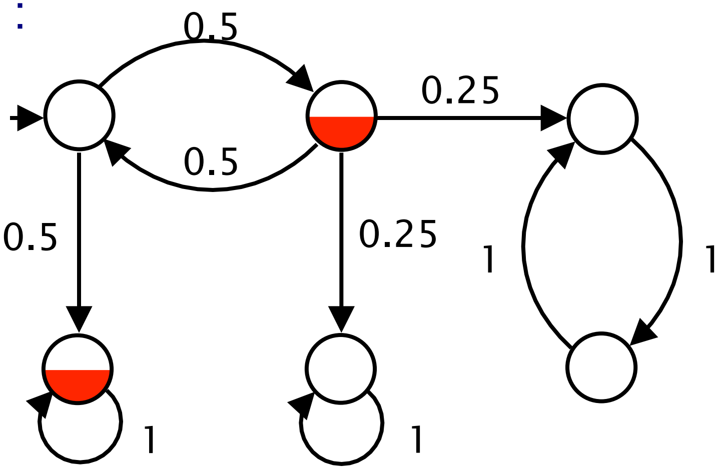
- What is the probability, having started in state s, of being in state s’ at time k?
  - i.e. after exactly k steps/transitions have occurred
  - this is the transient state probability: $\pi_{s,k}(s')$
- Transient state distribution: $\underline{\pi}_{s,k}$
  - vector $\underline{\pi}_{s,k}$ i.e. $\pi_{s,k}(s')$ for all states s’
- Note: this is a discrete probability distribution
  - so we have $\underline{\pi}_{s,k} : S \rightarrow [0,1]$
  - rather than e.g. $Pr_s : \Sigma_{Path(s)} \rightarrow [0,1]$ where $\Sigma_{Path(s)} \subseteq 2^{Path(s)}$

Transient distributions

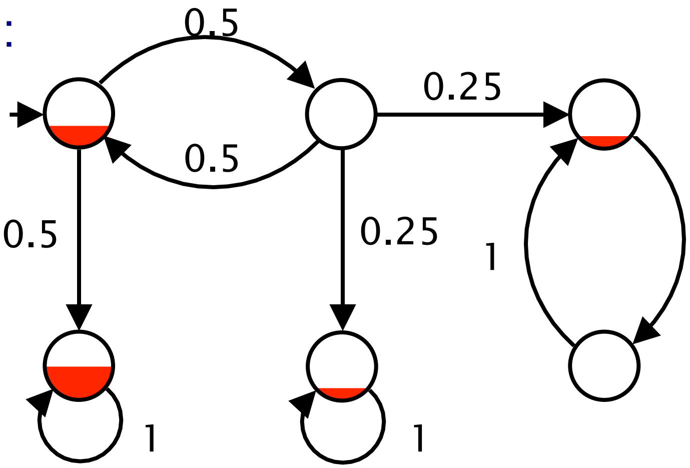
k=0:



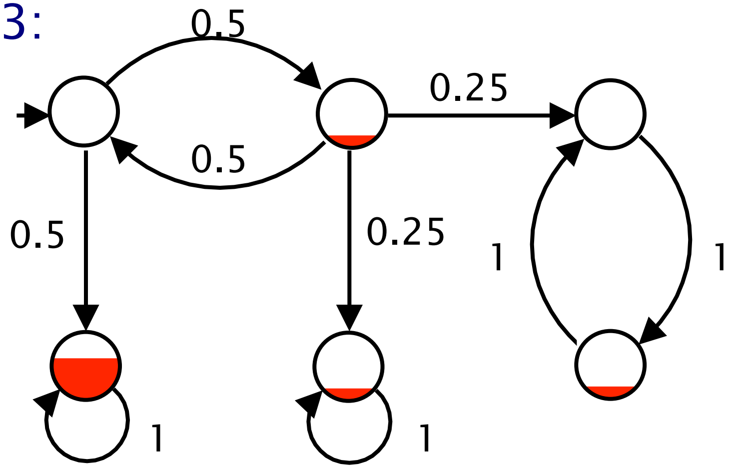
k=1:



k=2:



k=3:



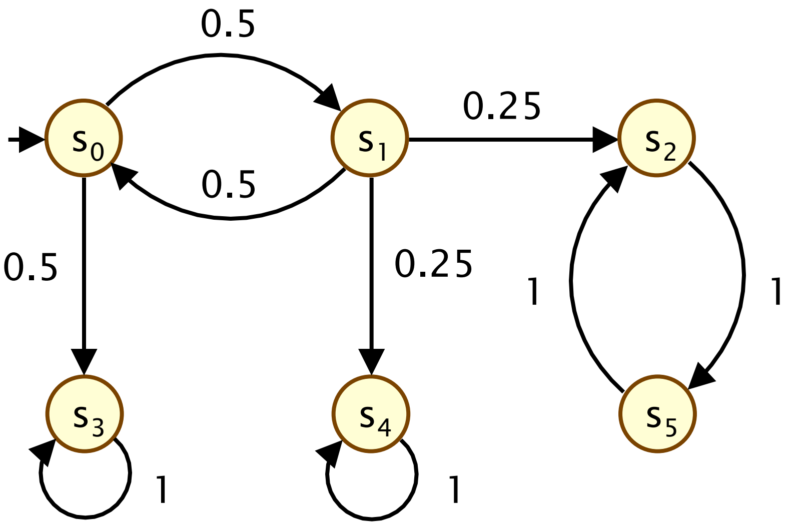
Computing transient probabilities

- Transient state probabilities:
  - $\pi_{s,k}(s') = \Sigma_{s''\in S}\ P(s'',s') \cdot \pi_{s,k-1}(s'')$
  - (i.e. look at incoming transitions)

- Computation of transient state distribution:
  - $\underline{\pi}_{s,0}$ is the initial probability distribution
  - e.g. in our case $\underline{\pi}_{s,0}(s') = 1$ if $s'=s$ and $\underline{\pi}_{s,0}(s') = 0$ otherwise
  - $\underline{\pi}_{s,k} = \underline{\pi}_{s,k-1} \cdot \mathbf{P}$

- i.e. successive vector-matrix multiplications

Computing transient probabilities



$$\underline{\pi}_{s0,0} = \left[1,0,0,0,0,0 \right]$$

$$\underline{\pi}_{s0,1} = \left[0,\tfrac{1}{2},0,\tfrac{1}{2},0,0 \right]$$

$$\underline{\pi}_{s0,2} = \left[\tfrac{1}{4},0,\tfrac{1}{8},\tfrac{1}{2},\tfrac{1}{8},0 \right]$$

$$\underline{\pi}_{s0,3} = \left[0,\tfrac{1}{8},0,\tfrac{5}{8},\tfrac{1}{8},\tfrac{1}{8} \right]$$

…

$$\mathbf{P} = \begin{bmatrix} 0 & 0.5 & 0 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Computing transient probabilities

- $\underline{\pi}_{s,k} = \underline{\pi}_{s,k-1} \cdot \mathbf{P} = \underline{\pi}_{s,0} \cdot \mathbf{P}^k$

- k^{th} matrix power: $\mathbf{P}^k$
  - **P** gives one-step transition probabilities
  - $\mathbf{P}^k$ gives probabilities of k-step transition probabilities
  - i.e. $\mathbf{P}^k(s,s') = \pi_{s,k}(s')$

- A possible optimisation: iterative squaring
  - e.g. $\mathbf{P}^8 = ((\mathbf{P}^2)^2)^2$
  - only requires log k multiplications
  - but potentially inefficient, e.g. if **P** is large and sparse

  - in practice, successive vector-matrix multiplications preferred

Notion of time in DTMCs

- Two possible views on the timing aspects of a system modelled as a DTMC:

- Discrete time-steps model time accurately
  - e.g. clock ticks in a model of an embedded device
  - or like dice example: interested in number of steps (tosses)

- Time-abstract
  - no information assumed about the time transitions take
  - e.g. simple Zeroconf model

- In the latter case, transient probabilities are not very useful
- In both cases, often beneficial to study long-run behaviour

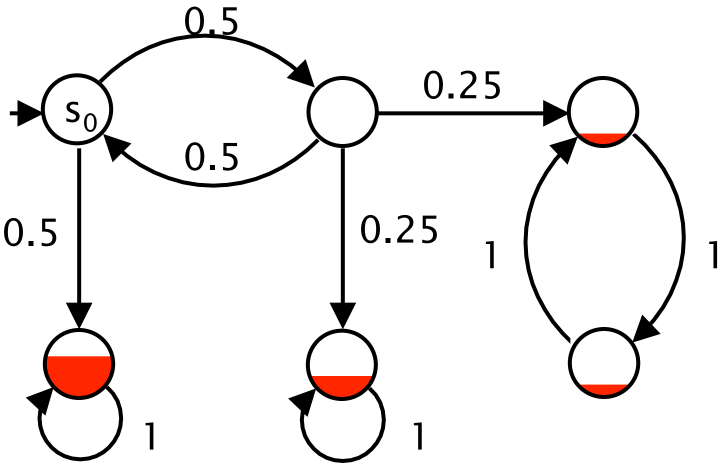
Long-run behaviour

- Consider the limit: $\underline{\pi}_s = \lim_{k\to\infty} \underline{\pi}_{s,k}$
  - where $\underline{\pi}_{s,k}$ is the transient state distribution at time k having starting in state s
  - this limit, where it exists, is called the limiting distribution

- Intuitive idea
  - the percentage of time, in the long run, spent in each state
  - e.g. reliability: "in the long-run, what percentage of time is the system in an operational state"

Limiting distribution

- Example:



$$\underline{\pi}_{s0,0} = \left[1,0,0,0,0,0 \right]$$

$$\underline{\pi}_{s0,1} = \left[0,\tfrac{1}{2},0,\tfrac{1}{2},0,0 \right]$$

$$\underline{\pi}_{s0,2} = \left[\tfrac{1}{4},0,\tfrac{1}{8},\tfrac{1}{2},\tfrac{1}{8},0 \right]$$

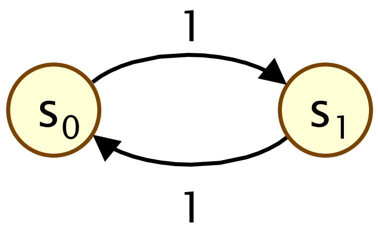
$$\underline{\pi}_{s0,3} = \left[0,\tfrac{1}{8},0,\tfrac{5}{8},\tfrac{1}{8},\tfrac{1}{8} \right]$$

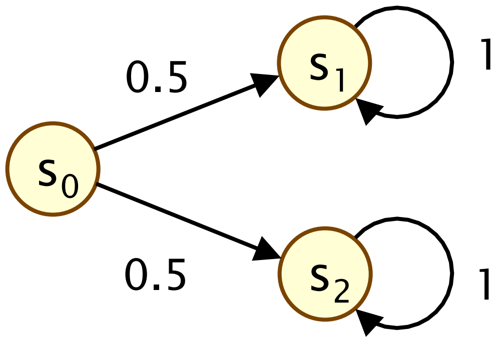
…

$$\underline{\pi}_{s0} = \left[0,0,\tfrac{1}{12},\tfrac{2}{3},\tfrac{1}{6},\tfrac{1}{12} \right]$$

Long-run behaviour

- Questions:
    - when does this limit exist?
    - does it depend on the initial state/distribution?





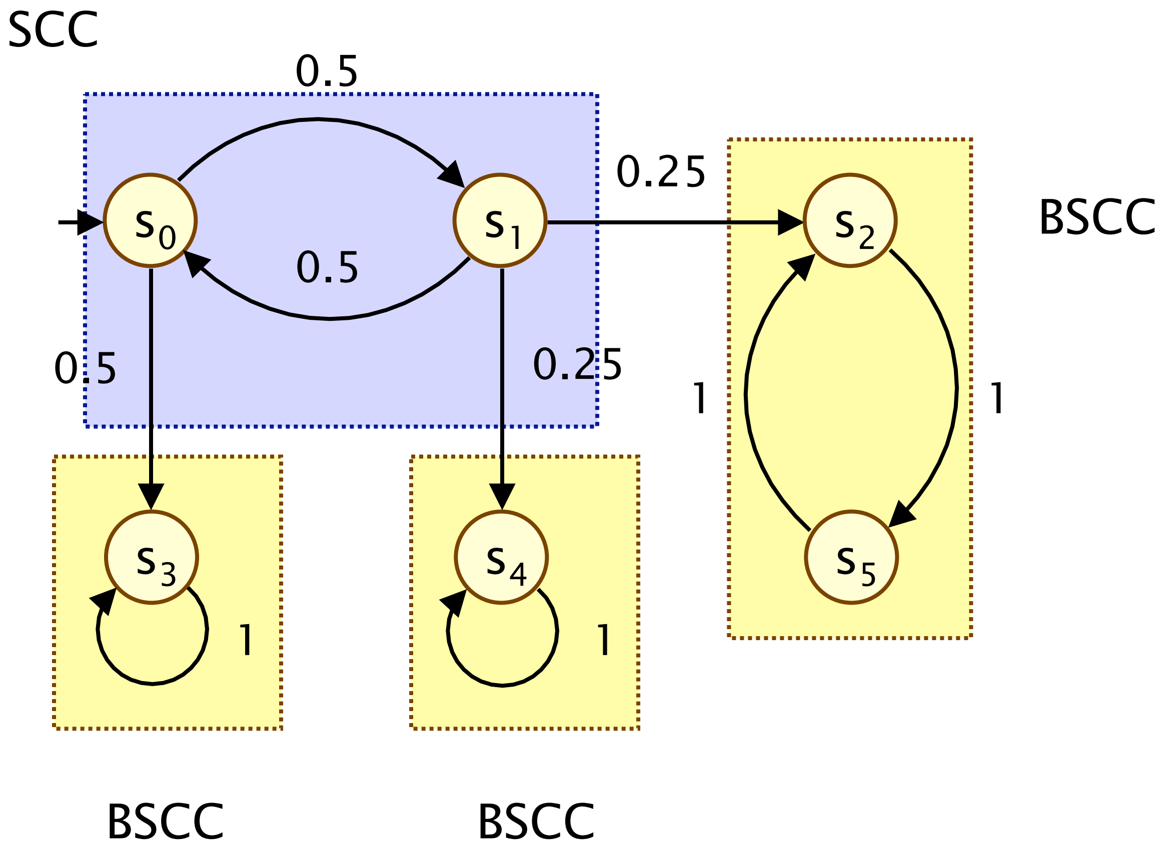
- Need to consider underlying graph
    - (V,E) where V are vertices and E $\subseteq$ VxV are edges
    - V = S and E = { (s,s') s.t. **P**(s,s') > 0 }

Graph terminology

- A state s' is reachable from s if there is a finite path starting in s and ending in s'
- A subset T of S is strongly connected if, for each pair of states s and s' in T, s' is reachable from s passing only through states in T
- A strongly connected component (SCC) is a maximally strongly connected set of states (i.e. no superset of it is also strongly connected)
- A bottom strongly connected component (BSCC) is an SCC T from which no state outside T is reachable from T

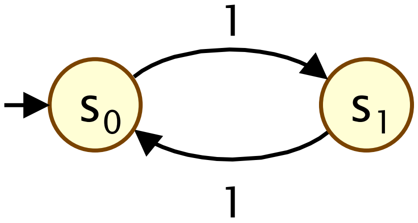
- Alternative terminology: "s communicates with s'", "communicating class", "closed communicating class"

Example – (B)SCCs



Graph terminology

- Markov chain is irreducible if all its states belong to a single BSCC; otherwise reducible



- A state s is periodic, with period d, if
  - the greatest common divisor of the set { n | $f_s^{(n)}>0$} equals d
  - where $f_s^{(n)}$ is the probability of, when starting in state s, returning to state s in exactly n steps

- A Markov chain is aperiodic if its period is 1

Steady-state probabilities

- For a finite, irreducible, aperiodic DTMC…
  - limiting distribution always exists
  - and is independent of initial state/distribution
- These are known as steady-state probabilities
  - (or equilibrium probabilities)
  - effect of initial distribution has disappeared, denoted $\underline{\pi}$
- These probabilities can be computed as the unique solution of the linear equation system:

$$\underline{\pi} \cdot P = \underline{\pi} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}(s) = 1$$

Steady-state – Balance equations

- Known as balance equations

$$\underline{\pi} \cdot P = \underline{\pi} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}(s) = 1$$

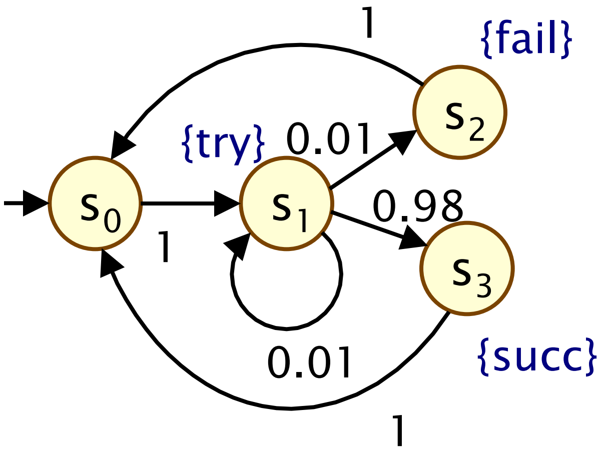
- That is:
  - $\underline{\pi}(s') = \Sigma_{s \in S} \, \underline{\pi}(s) \cdot P(s,s')$
  - $\Sigma_{s \in S} \, \underline{\pi}(s) = 1$

balance the probability of leaving and entering a state s’

normalisation

Steady-state – Example

- Let $\underline{x} = \underline{\pi}$
- Solve: $\underline{x} \cdot \mathbf{P} = \underline{x}$, $\Sigma_s \underline{x}(s) = 1$



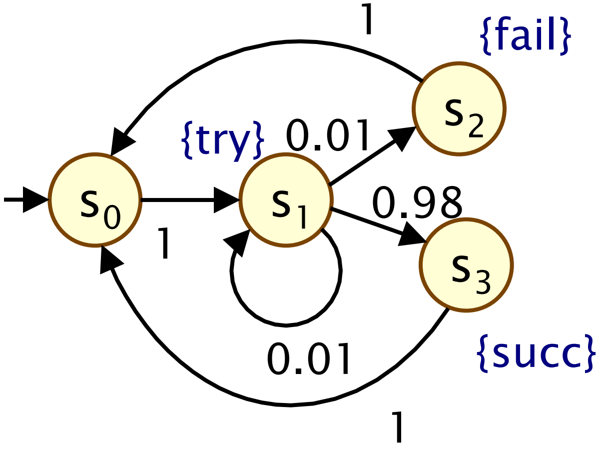
$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$\underline{x} \approx [\ 0.332215, 0.335570, 0.003356, 0.328859\ ]$

Steady-state – Example

- Let $\underline{x} = \underline{\pi}$
- Solve: $\underline{x} \cdot \mathbf{P} = \underline{x}$, $\Sigma_s \underline{x}(s) = 1$

$\underline{x} \approx [\ 0.332215, 0.335570,$
$0.003356, 0.328859\ ]$



Long-run percentage of time spent in the state “try” $\approx$ 33.6%

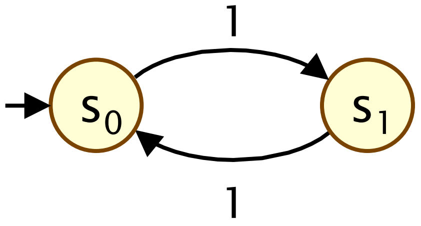
$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Long-run percentage of time spent in “fail”/”succ”
$\approx 0.003356 + 0.328859$
$\approx$ 33.2%

Periodic DTMCs

- For (finite, irreducible) periodic DTMCs, this limit:

$$\underline{\pi}_s(s') = \lim_{k \to \infty} \underline{\pi}_{s,k}(s')$$



- does not exist, but this limit does:

$$\lim_{n \to \infty} \frac{1}{n} \cdot \sum_{k=1}^{n} \underline{\pi}_{s,k}(s')$$

(and where both limits exist, e.g. for aperiodic DTMCs, these 2 limits coincide)

- Steady-state probabilities for these DTMCs can be computed by solving the same set of linear equations:

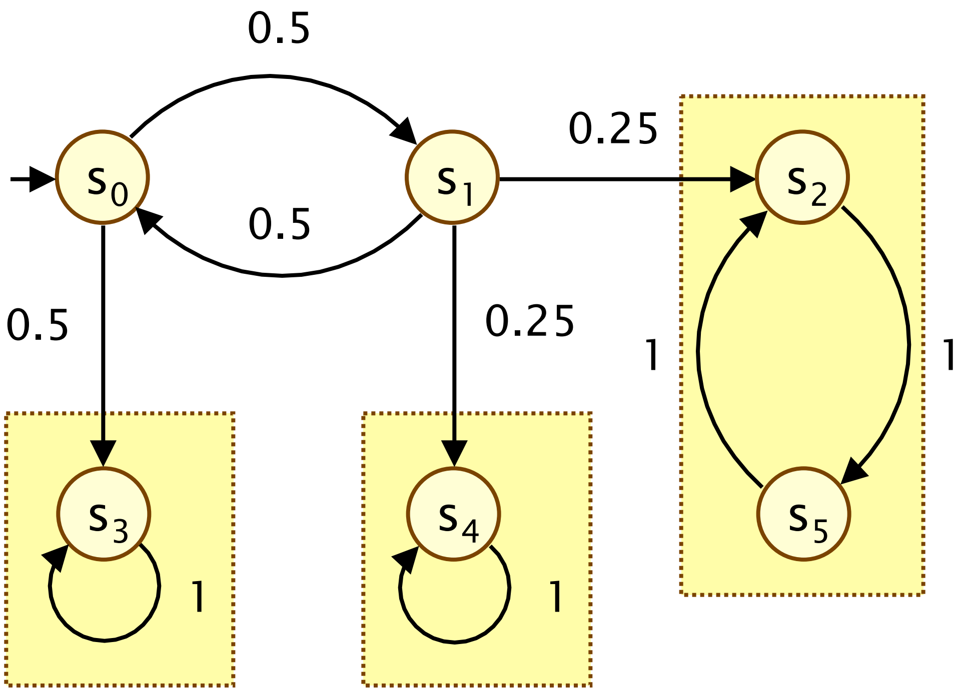
$$\underline{\pi} \cdot P = \underline{\pi} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}(s) = 1$$

Steady-state – General case

- General case: reducible DTMC
  - compute vector $\underline{\pi}_s$
  - (note: distribution depends on initial state s)
- Compute BSCCs for DTMC; then two cases to consider:
- (1) s is in a BSCC T
  - compute steady-state probabilities $\underline{x}$ in sub-DTMC for T
  - $\underline{\pi}_s(s') = \underline{x}(s')$ if s' in T
  - $\underline{\pi}_s(s') = 0$ if s' not in T
- (2) s is not in any BSCC
  - compute steady-state probabilities $\underline{x}_T$ for sub-DTMC of each BSCC T and combine with reachability probabilities to BSCCs
  - $\underline{\pi}_s(s') = \text{ProbReach}(s, T) \cdot \underline{x}_T(s')$ if s' is in BSCC T
  - $\underline{\pi}_s(s') = 0$ if s' is not in a BSCC

Steady-state – Example 2

- $\underline{\pi}_s$ depends on initial state s



$$\underline{\pi}_{s3} = [\ 0\ 0\ 0\ 1\ 0\ 0\ ]$$

$$\underline{\pi}_{s4} = [\ 0\ 0\ 0\ 0\ 1\ 0\ ]$$

$$\underline{\pi}_{s2} = \underline{\pi}_{s5} = \left[0,0,\tfrac{1}{2},0,0,\tfrac{1}{2}\right]$$

$$\underline{\pi}_{s0} = \left[0,0,\frac{1}{12},\frac{2}{3},\frac{1}{6},\frac{1}{12}\right]$$

$$\underline{\pi}_{s1} = \ldots$$

Qualitative properties

- Quantitative properties:
  - "what is the probability of event A?"

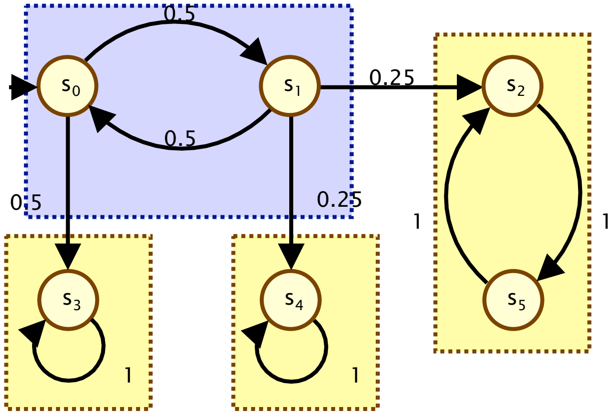
- Qualititative properties:
  - "the probability of event A is 1" ("almost surely A")
  - or: "the probability of event A is $>$ 0" ("possibly A")

- For finite DTMCs, qualititative properties do not depend on the transition probabilities – only need underlying graph
  - e.g. to determine "is target set T reached with probability 1?" (see DTMC model checking lecture)
  - computing BSCCs of a DTMCs yields information about long-run qualitative properties…

Fundamental property

- Fundamental property of (finite) DTMCs…

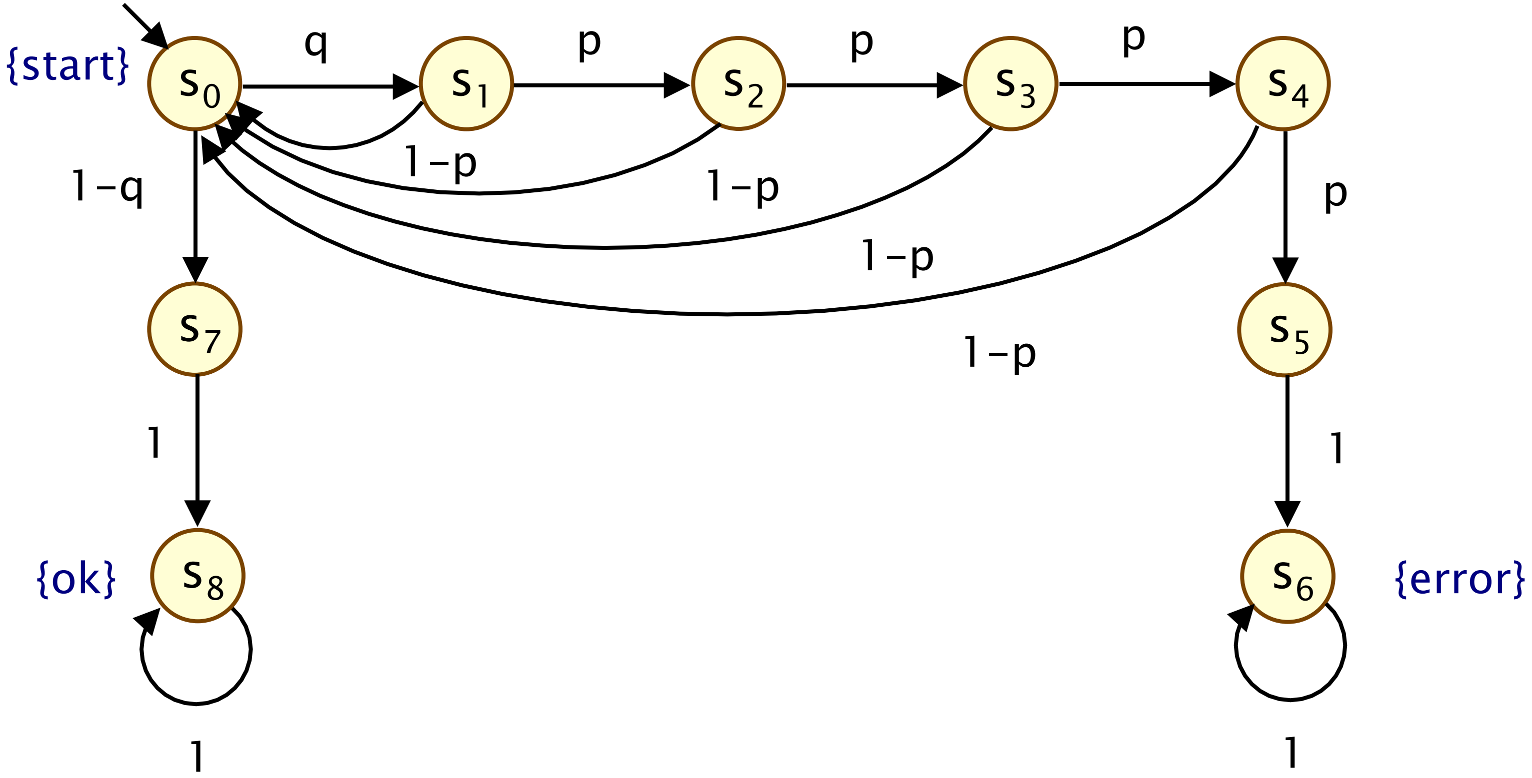
- With probability 1,
a BSCC will be reached
and all of its states
visited infinitely often



- Formally:
  - $\mathrm{Pr}_{s0}($ $s_0 s_1 s_2 \ldots$ | $\exists$ $i \geq 0$, $\exists$ BSCC T such that
$\forall$ $j \geq i$ $s_j \in$ T and
$\forall$ $s \in T$ $s_k = s$ for infinitely many k) $= 1$

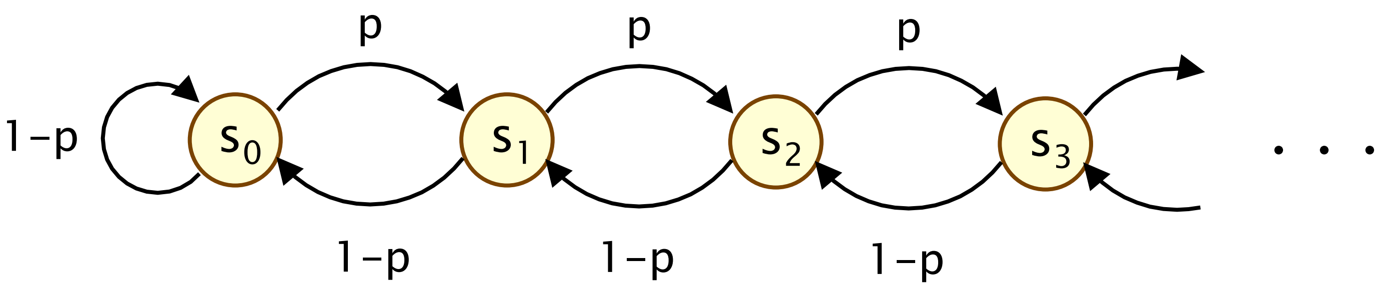
Zeroconf example

- 2 BSCCs: $\{s_6\}$, $\{s_8\}$
- Probability of trying to acquire a new address infinitely often is 0



Aside: Infinite Markov chains

- Infinite-state random walk



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

Repeated reachability

- Repeated reachability:
  - "always eventually…", "infinitely often…"
- $Pr_{s0} (s_0 s_1 s_2 \ldots \mid \forall\ i \geq 0\ \exists\ j \geq i\ s_j \in B)$
  - where $B \subseteq S$ is a set of states

- e.g. "what is the probability that the protocol successfully sends a message infinitely often?"

- Is this measurable? Yes…
  - set of satisfying paths is: $\bigcap_{n \geq 0} \bigcup_{m \geq n} C_m$
  - where C_m is the union of all cylinder sets $Cyl(s_0 s_1 \ldots s_m)$ for finite paths $s_0 s_1 \ldots s_m$ such that $s_m \in B$

Qualitative repeated reachability

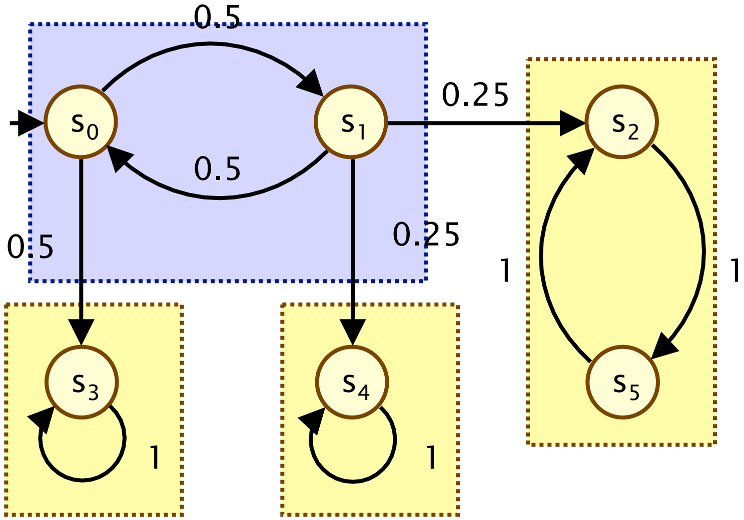
- Pr_{s0} ($s_0s_1s_2\ldots$ | $\forall$ $i \geq 0$ $\exists$ $j \geq i$ $s_j \in B$) $= 1$
  Pr_{s0} ("always eventually B") $= 1$

if and only if

- $T \cap B \neq \emptyset$ for each BSCC T that is reachable from s_0

Example:

$B = \{ s_3, s_4, s_5 \}$



Persistence

- Persistence properties:
  - “eventually forever…”
- $\mathrm{Pr}_{s0} (s_0 s_1 s_2 \ldots \mid \exists\, i \geq 0\ \forall\, j \geq i\ s_j \in B)$
  - where $B \subseteq S$ is a set of states

- e.g. “what is the probability of the leader election algorithm reaching, and staying in, a stable state?”

- e.g. “what is the probability that an irrecoverable error occurs?”

- Is this measurable? Yes…

Qualitative persistence

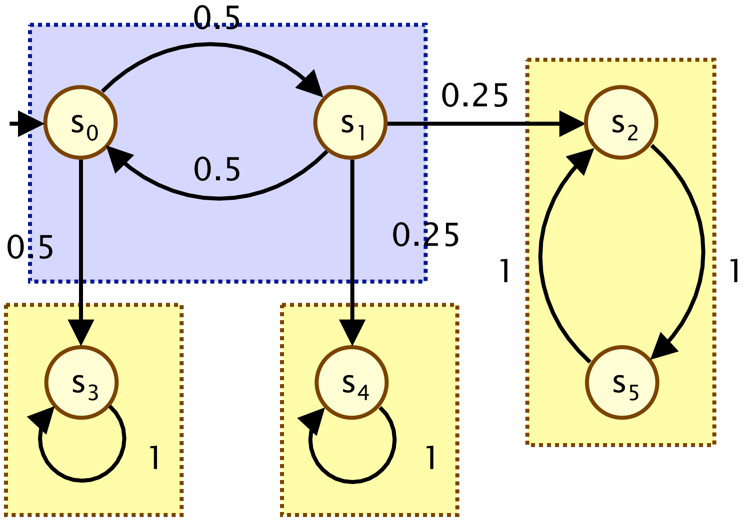
- Pr_{s0} ($s_0 s_1 s_2 \ldots \mid \exists\, i \geq 0\ \forall\, j \geq i\ s_j \in B$) $= 1$
  Pr_{s0} ("eventually forever B") $= 1$

  if and only if

- $T \subseteq B$ for each BSCC T that is reachable from s_0

Example:

$B = \{ s_2, s_3, s_4, s_5 \}$



Summing up…

- Transient state probabilities
  - successive vector-matrix multiplications

- Long-run/steady-state probabilities
  - requires graph analysis
  - irreducible case: solve linear equation system
  - reducible case: steady-state for sub-DTMCs + reachability

- Qualitative properties
  - repeated reachability
  - persistence

Probabilistic Model Checking Michaelmas Term 2011

Lecture 4
Probabilistic temporal logics

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- Temporal logic

- Non-probabilistic temporal logic
  - CTL

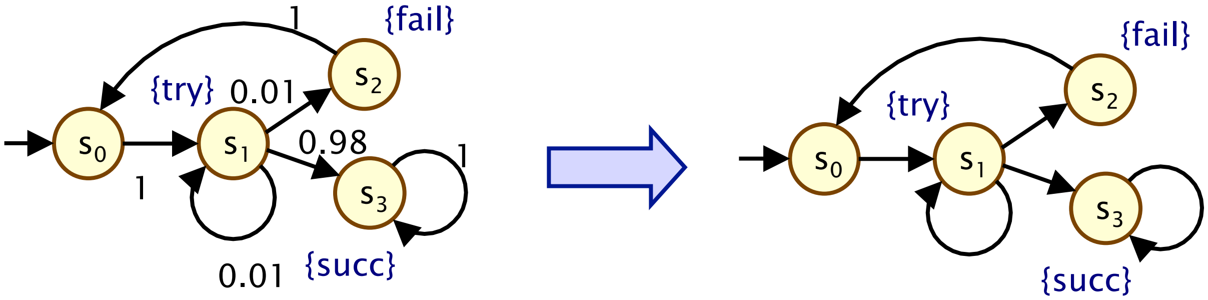
- Probabilistic temporal logic
  - PCTL = CTL + probabilities

- Qualitative vs. quantitative

- Linear-time properties
  - LTL, PCTL*

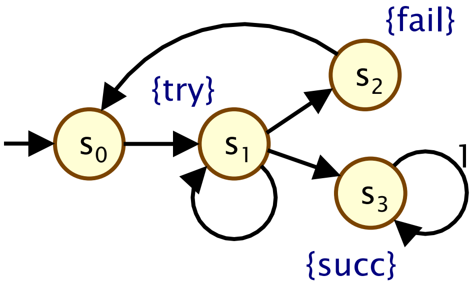
Temporal logic

- Temporal logic
  - formal language for specifying and reasoning about how the behaviour of a system changes over time
  - extends propositional logic with modal/temporal operators
  - one important use: representation of system properties to be checked by a model checker
- Logics used in this course are probabilistic extensions of temporal logics devised for non-probabilistic systems
  - So we revert briefly to (labelled) state-transition diagrams



State–transition systems

- Labelled state–transition system (LTS) (or Kripke structure)
  - is a tuple $(S,s_{init},\rightarrow,L)$ where:
  - S is a set of states (“state space”)
  - $s_{init} \in S$ is the initial state
  - $\rightarrow \subseteq S \times S$ is the **transition relation**
  - $L : S \rightarrow 2^{AP}$ is function labelling states with atomic propositions (taken from a set AP)



- DTMC $(S,s_{init},\mathbf{P},L)$ has underlying LTS $(S,s_{init},\rightarrow,L)$
  - where $\rightarrow = \{ (s,s') \text{ s.t. } \mathbf{P}(s,s') > 0 \}$

Paths – some notation

- Path $\omega = s_0 s_1 s_2 \ldots$ such that $(s_i, s_{i+1}) \in \rightarrow$ for $i \geq 0$
  - we write $s_i \rightarrow s_{i+1}$ as shorthand for $(s_i, s_{i+1}) \in \rightarrow$

- $\omega(i)$ is the (i+1)th state of ω, i.e. s_i

- $\omega[\ldots i]$ denotes the (finite) prefix ending in the (i+1)th state
  - i.e. $\omega[\ldots i] = s_0 s_{1 \ldots} s_i$

- $\omega[i \ldots]$ denotes the suffix starting from the (i+1)th state
  - i.e. $\omega[i \ldots] = s_i s_{i+1} s_{i+2} \ldots$

- As for DTMCs, Path(s) = set of all infinite paths from s

CTL

- CTL – Computation Tree Logic
- Syntax split into state and path formulae
  - specify properties of states/paths, respectively
  - a CTL formula is a state formula

Some of these operators (e.g. A, F, G) are derivable…

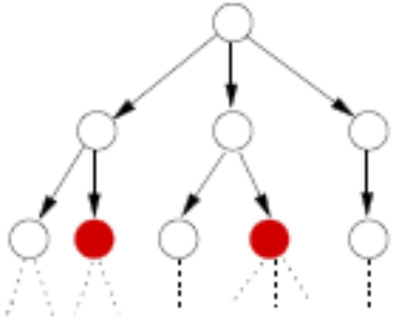
- State formulae:
  - $\phi \ ::= \ \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{A}\ \psi \mid \mathrm{E}\ \psi$
  - where $a \in AP$ and ψ is a path formula

- Path formulae
  - $\psi \ ::= \ \mathrm{X}\ \phi \mid \mathrm{F}\ \phi \mid \mathrm{G}\ \phi \mid \phi\ \mathrm{U}\ \phi$
  - where ϕ is a state formula

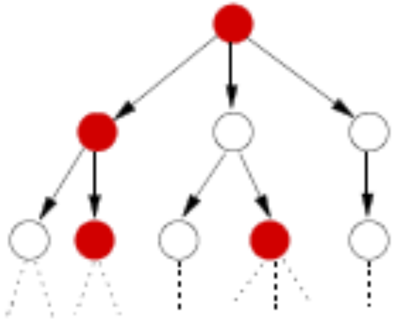
X = “next”
F = “future”
G = “globally”
U = “until”

CTL semantics

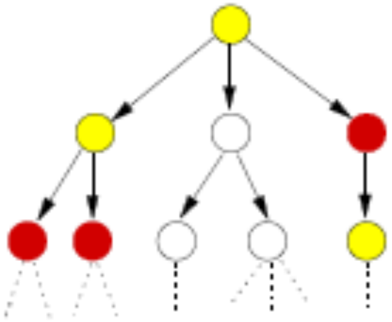
- Intuitive semantics:
  - of quantifiers (A/E) and temporal operators (F/G/U)



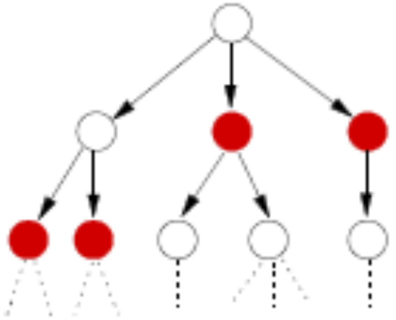
EF red



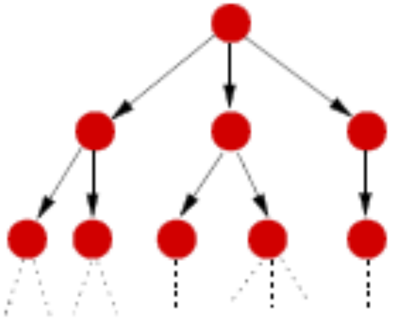
EG red



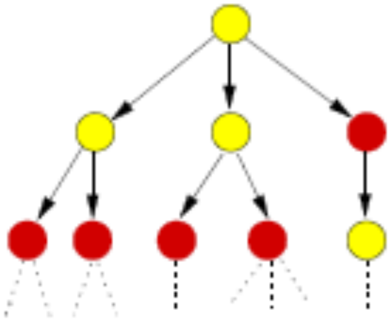
E [yellow U red]



AF red



AG red



A [yellow U red]

CTL semantics

- Semantics of state formulae:
  - $s \models \phi$ denotes "s satisfies ϕ" or "ϕ is true in s"

- For a state s of an LTS $(S, s_{init}, \rightarrow, L)$:

  - $s \models \text{true}$ always
  - $s \models a \Leftrightarrow a \in L(s)$
  - $s \models \phi_1 \wedge \phi_2 \Leftrightarrow s \models \phi_1$ and $s \models \phi_2$
  - $s \models \neg\phi \Leftrightarrow s \not\models \phi$
  - $s \models A\,\psi \Leftrightarrow \omega \models \psi$ for all $\omega \in Path(s)$
  - $s \models E\,\psi \Leftrightarrow \omega \models \psi$ for some $\omega \in Path(s)$

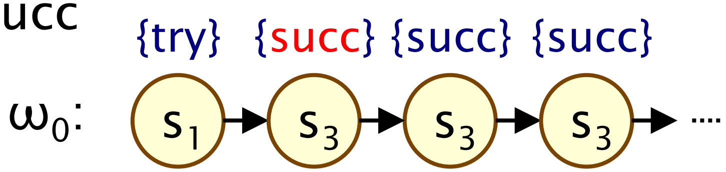
CTL semantics

- Semantics of path formulae:
  - $\omega \vDash \psi$ denotes "ω satisfies ψ" or "ψ is true along ω"

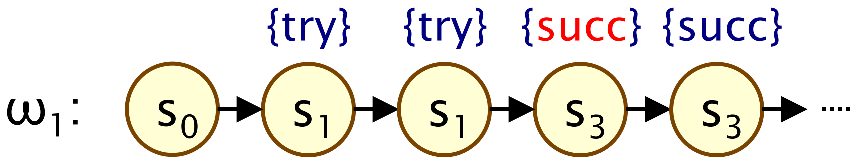
- For a path ω of an LTS $(S, s_{init}, \rightarrow, L)$:
  - $\omega \vDash X\ \phi \quad \Leftrightarrow \quad \omega(1) \vDash \phi$
  - $\omega \vDash F\ \phi \quad \Leftrightarrow \quad \exists k \geq 0$ s.t. $\omega(k) \vDash \phi$
  - $\omega \vDash G\ \phi \quad \Leftrightarrow \quad \forall i \geq 0\ \omega(i) \vDash \phi$
  - $\omega \vDash \phi_1\ U\ \phi_2 \quad \Leftrightarrow \quad \exists k \geq 0$ s.t. $\omega(k) \vDash \phi_2$ and $\forall i < k\ \omega(i) \vDash \phi_1$

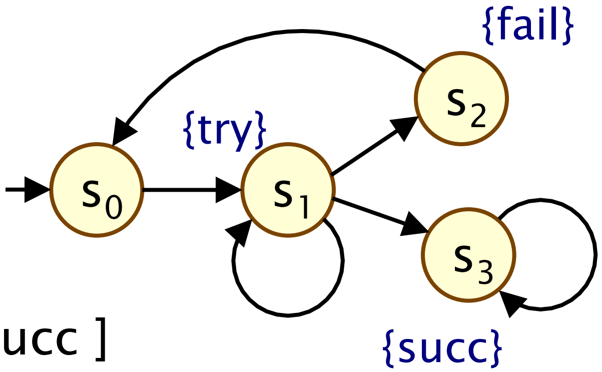
CTL examples

- Some examples of satisfying paths:
  - $\omega_0 \models$ X succ



  - $\omega_1 \models \neg$fail U succ





- Example CTL formulas:
  - $s_1 \models$ try $\wedge$ $\neg$fail
  - $s_1 \models$ E [X succ] and s_1, $s_3 \models$ A [X succ]
  - $s_0 \models$ E [$\neg$fail U succ] but $s_0 \not\models$ A [$\neg$fail U succ]

CTL examples

- $\mathrm{AG}\ (\neg(\mathrm{crit}_1 \wedge \mathrm{crit}_2))$
  - mutual exclusion
- AG EF initial
  - for every computation, it is always possible to return to the initial state
- $\mathrm{AG}\ (\mathrm{request} \rightarrow \mathrm{AF}\ \mathrm{response})$
  - every request will eventually be granted
- $\mathrm{AG}\ \mathrm{AF}\ \mathrm{crit}_1 \wedge \mathrm{AG}\ \mathrm{AF}\ \mathrm{crit}_2$
  - each process has access to the critical section infinitely often

CTL equivalences

- Basic logical equivalences:
  - false $\equiv \neg$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)

- Path quantifiers:
  - $A\ \psi \equiv \neg E(\neg\psi)$
  - $E\ \psi \equiv \neg A(\neg\psi)$

- Temporal operators:
  - $F\ \phi \equiv \text{true}\ U\ \phi$
  - $G\ \phi \equiv \neg F(\neg\phi)$

For example:

$AG\ \phi \equiv \neg EF(\neg\ \phi)$

CTL – Alternative notation

- Some commonly used notation…

- Temporal operators:
  - $F\ \phi \equiv \Diamond\ \phi$ (“diamond”)
  - $G\ \phi \equiv \Box\ \phi$ (“box”)
  - $X\ \phi \equiv \bigcirc\ \phi$
- Path quantifiers:
  - $A\ \psi \equiv \forall\ \psi$
  - $E\ \psi \equiv \exists\ \psi$
- Brackets: none/round/square
  - $AF\ \psi$
  - $A\ (\ \psi_1\ U\ \psi_2\ )$
  - $A\ [\ \psi_1\ U\ \psi_2\ ]$

PCTL

- Temporal logic for describing properties of DTMCs
  - PCTL = Probabilistic Computation Tree Logic [HJ94]
  - essentially the same as the logic pCTL of [ASB+95]

- Extension of (non-probabilistic) temporal logic CTL
  - key addition is probabilistic operator P
  - quantitative extension of CTL's A and E operators

- Example
  - send → $P_{\geq 0.95}$ [$F^{\leq 10}$ deliver]
  - "if a message is sent, then the probability of it being delivered within 10 steps is at least 0.95"

PCTL syntax

- PCTL syntax:

ψ is true with probability ~p

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

“next” “bounded until” “until”

  - where a is an atomic proposition, $p \in [0,1]$ is a probability bound, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

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

PCTL semantics for DTMCs

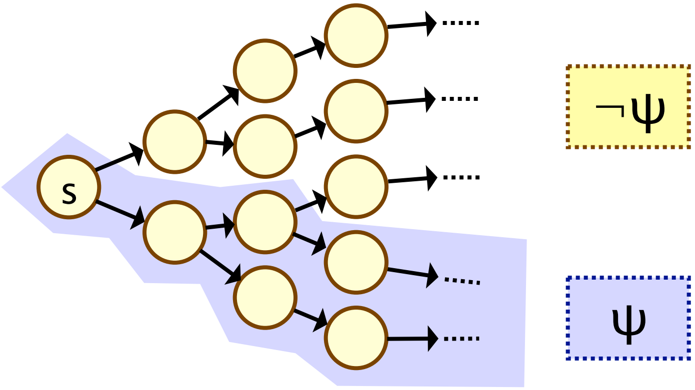
- Semantics for non-probabilistic operators same as for CTL:
  - $s \vDash \phi$ denotes "s satisfies ϕ" or "ϕ is true in s"
  - $\omega \vDash \psi$ denotes "ω satisfies ψ" or "ψ is true along ω"

- For a state s of a DTMC $(S,s_{init},\mathbf{P},L)$:
  - $s \vDash \text{true}$ always
  - $s \vDash a \;\Leftrightarrow\; a \in L(s)$
  - $s \vDash \phi_1 \wedge \phi_2 \;\Leftrightarrow\; s \vDash \phi_1$ and $s \vDash \phi_2$
  - $s \vDash \neg\phi \;\Leftrightarrow\; s \nvDash \phi$
- For a path ω of a DTMC $(S,s_{init},\mathbf{P},L)$:
  - $\omega \vDash X\,\phi \;\Leftrightarrow\; \omega(1) \vDash \phi$
  - $\omega \vDash \phi_1\, U^{\leq k}\, \phi_2 \;\Leftrightarrow\; \exists i \leq k$ such that $\omega(i) \vDash \phi_2$ and $\forall j<i,\ \omega(j) \vDash \phi_1$
  - $\omega \vDash \phi_1\, U\, \phi_2 \;\Leftrightarrow\; \exists k \geq 0$ s.t. $\omega(k) \vDash \phi_2$ and $\forall i<k\ \omega(i) \vDash \phi_1$

$U^{\leq k}$ not in CTL (but could easily be added)

PCTL semantics for DTMCs

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



$\text{Prob}(s, \psi) \sim p$?

PCTL equivalences for DTMCs

- Basic logical equivalences:
  - false $\equiv \neg$true (false)
  - $\phi_1 \lor \phi_2 \equiv \neg(\neg\phi_1 \land \neg\phi_2)$ (disjunction)
  - $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \lor \phi_2$ (implication)

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

Reachability and invariance

- Derived temporal operators, like CTL…

- Probabilistic reachability: $P_{\sim p}$ [F ϕ]
  - the probability of reaching a state satisfying ϕ
  - $F\ \phi \equiv \text{true}\ U\ \phi$
  - "ϕ is eventually true"
  - bounded version: $F^{\leq k}\ \phi \equiv \text{true}\ U^{\leq k}\ \phi$

- Probabilistic invariance: $P_{\sim p}$ [G ϕ]
  - the probability of ϕ always remaining true
  - $G\ \phi \equiv \neg(F\ \neg\phi) \equiv \neg(\text{true}\ U\ \neg\phi)$
  - "ϕ is always true"
  - bounded version: $G^{\leq k}\ \phi \equiv \neg(F^{\leq k}\ \neg\phi)$

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

Derivation of $P_{\sim p}$ [G ϕ]

- In fact, we can derive $P_{\sim p}$ [G ϕ] directly in PCTL…

PCTL examples

- $P_{<0.05}$ [F err/total>0.1]
  - "with probability at most 0.05, more than 10% of the NAND gate outputs are erroneous?"
- $P_{\geq 0.8}$ [$F^{\leq k}$ reply_count=n]
  - "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8"
- $P_{<0.4}$ [$\neg \mathrm{fail}_A$ U fail_B]
  - "the probability that component B fails before component A is less than 0.4"
- $\neg$oper → $P_{\geq 1}$ [F ($P_{>0.99}$ [$G^{\leq 100}$ oper])]
  - "if the system is not operational, it almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units"

PCTL and measurability

- All the sets of paths expressed by PCTL are measurable
  - i.e. are elements of the σ–algebra $\Sigma_{Path(s)}$
  - see for example [Var85] (for a stronger result in fact)

- Recall: probability space $(Path(s), \Sigma_{Path(s)}, Pr_s)$
  - $\Sigma_{Path(s)}$ contains cylinder sets C(ω) for all finite paths ω starting in s and is closed under complementation, countable union

- Next $(X\ \phi)$
  - cylinder sets constructed from paths of length one
- Bounded until $(\phi_1\ U^{\leq k}\ \phi_2)$
  - (finite number of) cylinder sets from paths of length at most k
- Until $(\phi_1\ U\ \phi_2)$
  - countable union of paths satisfying $\phi_1\ U^{\leq k}\ \phi_2$ for all $k \geq 0$

Qualitative vs. quantitative properties

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

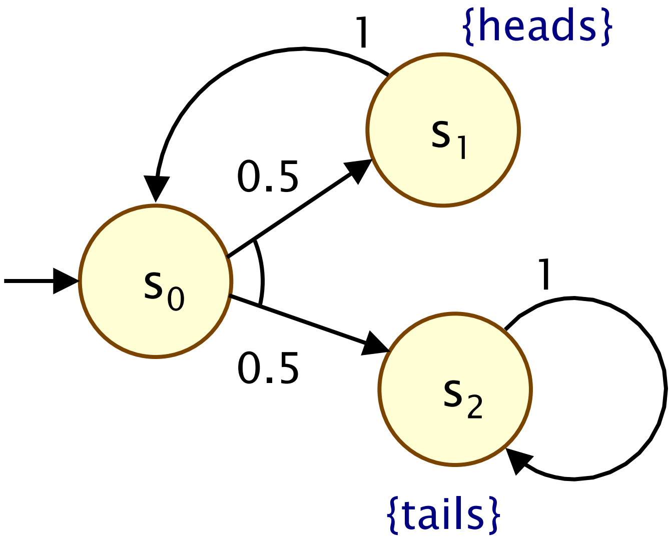
- Qualitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is either 0 or 1
- Quantitative PCTL properties
  - $P_{\sim p}$ [ψ] where p is in the range (0,1)

- $P_{>0}$ [F ϕ] is identical to EF ϕ
  - there exists a finite path to a ϕ-state
- $P_{\geq 1}$ [F ϕ] is (similar to but) weaker than AF ϕ
  - a ϕ-state is reached "almost surely"
  - see next slide…

Example: Qualitative/quantitative

- Toss a coin repeatedly until "tails" is thrown

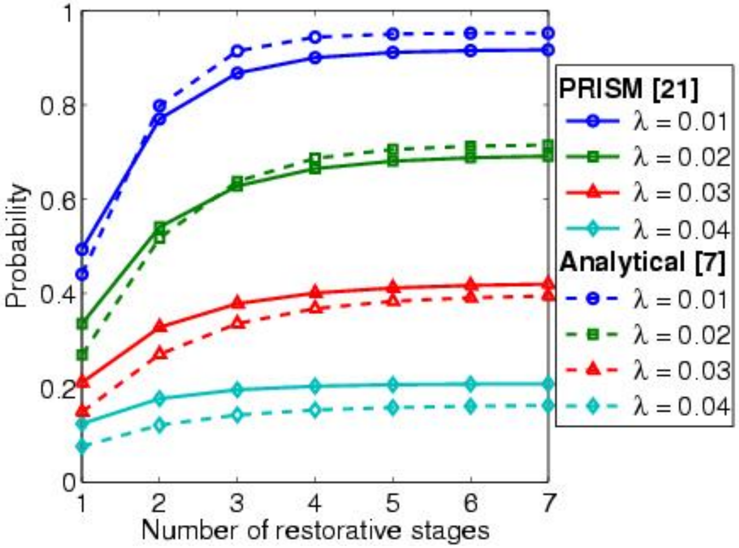
- Is "tails" always eventually thrown?
  - CTL: AF "tails"
  - Result: false
  - Counterexample: $s_0s_1s_0s_1s_0s_1\ldots$



- Does the probability of eventually throwing "tails" equal one?
  - PCTL: $P_{\geq 1}$ [F "tails"]
  - Result: true
  - Infinite path $s_0s_1s_0s_1s_0s_1\ldots$ has zero probability

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
  - PRISM allows formulae of 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?"



Limitations of PCTL

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

- More expressive logics can be used, for example:
  - LTL [Pnu77], the non-probabilistic linear-time temporal logic
  - PCTL* [ASB+95,BdA95] which subsumes both PCTL and LTL

- To introduce these logics, we return briefly again to non-probabilistic logics and models…

Branching vs. Linear time

- In CTL, temporal operators always appear inside A or E
  - in LTL, temporal operators can be combined

- LTL but not CTL:
  - F [req $\wedge$ X ack]
  - "eventually a request occurs, followed immediately by an acknowledgement"

- CTL but not LTL:
  - AG EF initial
  - "for every computation, it is always possible to return to the initial state"

LTL

- 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

- LTL semantics (for a path ω)
  - $\omega \models \text{true}$ always
  - $\omega \models a \quad \Leftrightarrow \quad a \in L(\omega(0))$
  - $\omega \models \psi_1 \wedge \psi_2 \quad \Leftrightarrow \quad \omega \models \psi_1$ and $\omega \models \psi_2$
  - $\omega \models \neg\psi \quad \Leftrightarrow \quad \omega \not\models \psi$
  - $\omega \models X\,\psi \quad \Leftrightarrow \quad \omega[1\ldots] \models \psi$
  - $\omega \models \psi_1\ U\ \psi_2 \quad \Leftrightarrow \quad \exists k \geq 0$ s.t. $\omega[k\ldots] \models \psi_2$ and $\forall i{<}k\ \omega[i\ldots] \models \psi_1$

LTL

- LTL semantics
  - implicit universal quantification over paths
  - i.e. for an LTS $M = (S, s_{init}, \rightarrow, L)$ and LTL formula ψ
  - $s \vDash \psi$ iff $\omega \vDash \psi$ for all paths $\omega \in Path(s)$
  - $M \vDash \psi$ iff $s_{init} \vDash \psi$
- e.g:
  - $A\ F\ [\ req \wedge X\ ack\ ]$
  - “it is always the case that, eventually, a request occurs, followed immediately by an acknowledgement”

- Derived operators like CTL, for example:
  - $F\ \psi \equiv true\ U\ \psi$
  - $G\ \psi \equiv \neg F(\neg \psi)$

LTL + probabilities

- Same idea as PCTL: probabilities of sets of path formulae
  - for a state s of a DTMC and an LTL formula ψ:
  - Prob(s, ψ) = $\Pr_s \{ \omega \in \mathrm{Path}(s) \mid \omega \vDash \psi \}$
  - all such path sets are measurable (see later)
- Examples (from DTMC lectures)…
- Repeated reachability: "always eventually…"
  - Prob(s, GF send)
  - e.g. "what is the probability that the protocol successfully sends a message infinitely often?"
- Persistence properties: "eventually forever…"
  - Prob(s, FG stable)
  - e.g. "what is the probability of the leader election algorithm reaching, and staying in, a stable state?"

PCTL*

- PCTL* subsumes both (probabilistic) LTL and PCTL

- State formulae:
  - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p} [\psi]$
  - where $a \in AP$ and ψ is a path formula

- Path formulae:
  - $\psi ::= \phi \mid \psi \wedge \psi \mid \neg\psi \mid X\, \psi \mid \psi\, U\, \psi$
  - where ϕ is a state formula

- A PCTL* formula is a state formula ϕ
  - e.g. $P_{>0.1} [\text{GF crit}_1] \wedge P_{>0.1} [\text{GF crit}_2]$

Summing up…

- Temporal logic:
  - formal language for specifying and reasoning about how the behaviour of a system changes over time

CTL	Φ	non-probabilistic (e.g. LTSs)
LTL	ψ	
PCTL	Φ	probabilistic (e.g. DTMCs)
LTL + prob.	Prob(s, ψ)	
PCTL*	Φ	

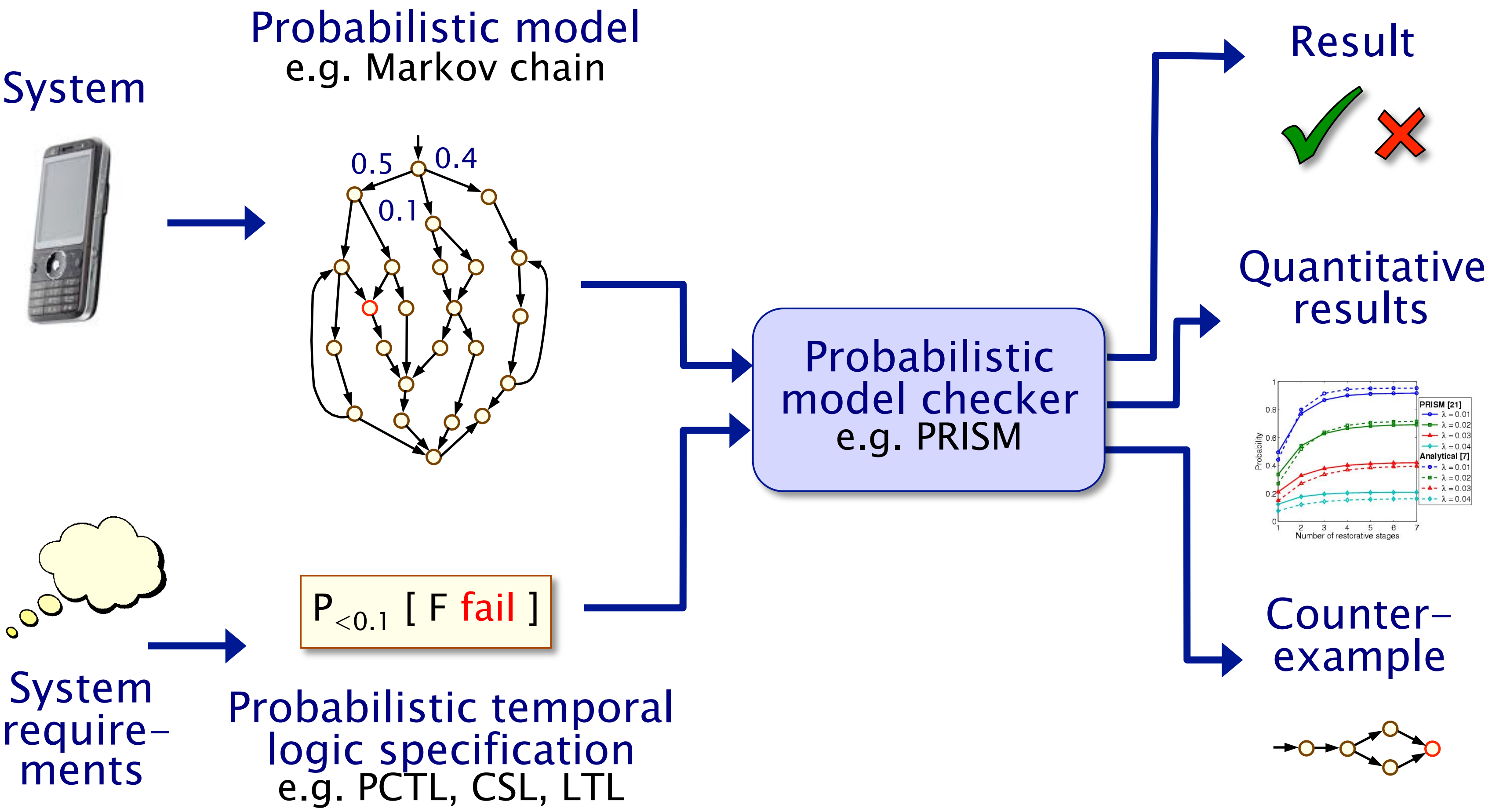
Lecture 5

PCTL Model Checking for DTMCs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Probabilistic model checking



System

Probabilistic model
e.g. Markov chain

System require-ments

$P_{<0.1}$ [F fail]

Probabilistic temporal logic specification
e.g. PCTL, CSL, LTL

Probabilistic model checker
e.g. PRISM

Result

✓ ✗

Quantitative results

Counter-example

Overview

- PCTL model checking for DTMCs
- Computation of probabilities for PCTL formulae
  - next
  - bounded until
  - (unbounded) until
- Solving large linear equation systems
  - direct vs. iterative methods
  - iterative solution methods

PCTL

- PCTL syntax:

ψ is true with probability ~p

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

“next” “bounded until” “until”

  - where a is an atomic proposition, $p \in [0,1]$ is a probability bound, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

- Remaining operators can be derived (false, ∨, →, F, G, …)
  - hence will not be discussed here

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

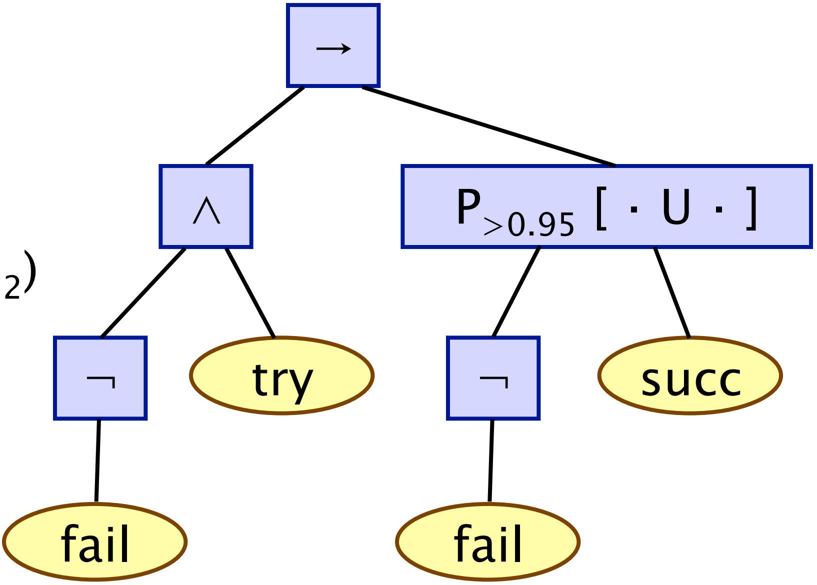
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - often, just want to know if $s_{init} \vDash \phi$, i.e. if $s_{init} \in Sat(\phi)$
  - sometimes, want to check that $s \vDash \phi$ $\forall$ $s \in S$, i.e. $Sat(\phi) = S$

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



- For the $P_{\sim p}$ [ψ] operator:
  - need to compute the probabilities $\text{Prob}(s, \psi)$ for all states $s \in S$
  - $\text{Sat}(P_{\sim p} [\ \psi\ ]) = \{ s \in S \mid \text{Prob}(s, \psi) \sim p \}$

Probability computation

- Three temporal operators to consider:

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

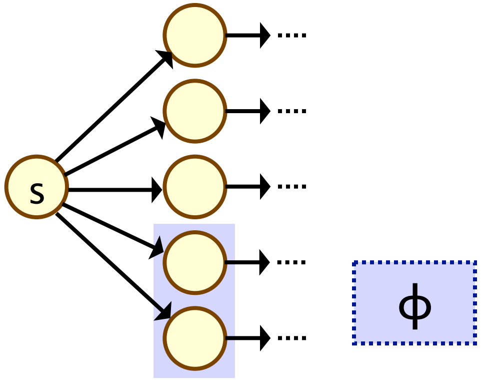
- Bounded until: $P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]$
  - adaptation of bounded reachability for DTMCs

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

PCTL next for DTMCs

- Computation of probabilities for PCTL next operator
  - $\mathrm{Sat}(P_{\sim p}[\ X\ \phi\ ]) = \{\ s \in S \mid \mathrm{Prob}(s, X\ \phi) \sim p\ \}$
  - need to compute $\mathrm{Prob}(s, X\ \phi)$ for all $s \in S$

- Sum outgoing probabilities for transitions to ϕ-states
  - $\mathrm{Prob}(s, X\ \phi) = \Sigma_{s' \in \mathrm{Sat}(\phi)}\ \mathbf{P}(s,s')$



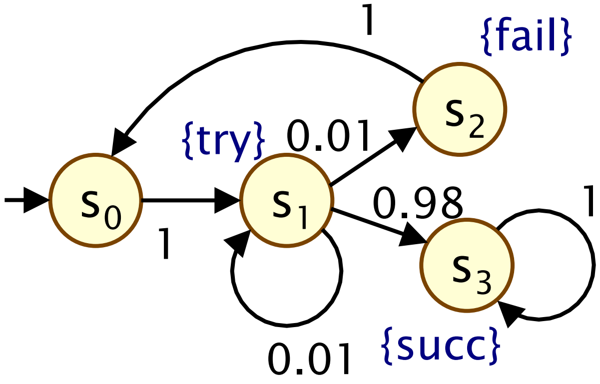
- Compute vector $\underline{\mathrm{Prob}}(X\ \phi)$ of probabilities for all states s
  - $\underline{\mathrm{Prob}}(X\ \phi) = \mathbf{P} \cdot \underline{\phi}$
  - where $\underline{\phi}$ is a 0-1 vector over S with $\underline{\phi}(s) = 1$ iff $s \vDash \phi$
  - computation requires a single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

PCTL next – Example

- Model check: $P_{\geq 0.9}$ [X ($\neg$try $\vee$ succ)]
  - Sat ($\neg$try $\vee$ succ) = (S \ Sat(try)) $\cup$ Sat(succ)
    = $(\{s_0,s_1,s_2,s_3\} \setminus \{s_1\}) \cup \{s_3\} = \{s_0,s_2,s_3\}$

  - $\underline{\text{Prob}}$(X ($\neg$try $\vee$ succ)) = **P** · $\underline{(\neg\text{try} \vee \text{succ})}$ = …

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

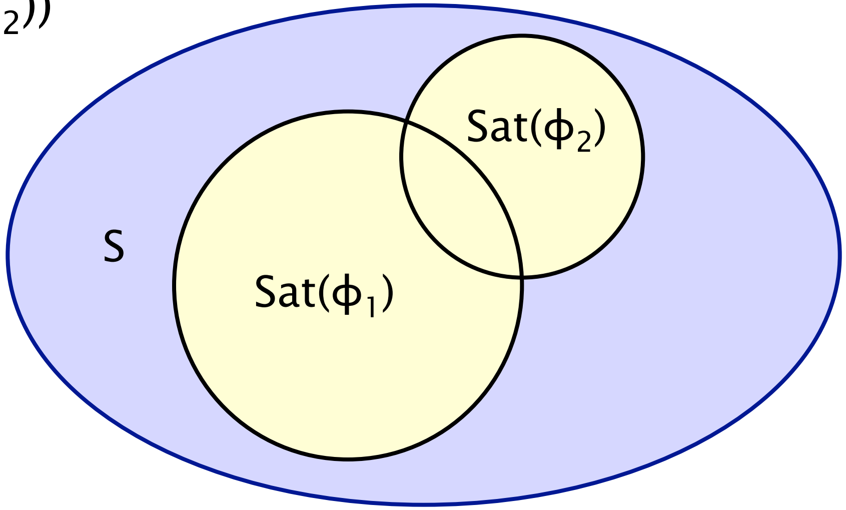


- Results:
  - $\underline{\text{Prob}}$(X ($\neg$try $\vee$ succ)) = [0, 0.99, 1, 1]
  - Sat($P_{\geq 0.9}$ [X ($\neg$try $\vee$ succ)]) = $\{s_1, s_2, s_3\}$

PCTL bounded until for DTMCs

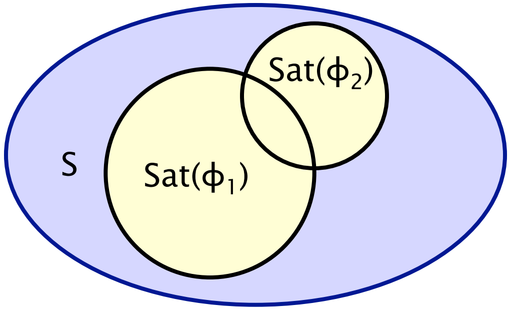
- Computation of probabilities for PCTL $U^{\leq k}$ operator
  - $\text{Sat}(P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]) = \{\ s \in S \mid \text{Prob}(s,\ \phi_1\ U^{\leq k}\ \phi_2) \sim p\ \}$
  - need to compute $\text{Prob}(s,\ \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$

- First identify (some) states where probability is trivially 1/0
  - $S^{yes} = \text{Sat}(\phi_2)$
  - $S^{no} = S \setminus (\text{Sat}(\phi_1) \cup \text{Sat}(\phi_2))$



PCTL bounded until for DTMCs

- Let:
  - $S^{yes} = Sat(\phi_2)$
  - $S^{no} = S \setminus (Sat(\phi_1) \cup Sat(\phi_2))$
- And let:
  - $S^? = S \setminus (S^{yes} \cup S^{no})$



- Compute solution of recursive equations:

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

PCTL bounded until for DTMCs

- Simultaneous computation of vector $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2)$
  - i.e. probabilities $\text{Prob}(s, \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$

- Iteratively define in terms of matrices and vectors
  - define matrix **P'** as follows: $\mathbf{P'}(s,s') = \mathbf{P}(s,s')$ if $s \in S^?$, $\mathbf{P'}(s,s') = 1$ if $s \in S^{yes}$ and $s=s'$, $\mathbf{P'}(s,s') = 0$ otherwise
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq 0}\ \phi_2) = \underline{\phi}_2$
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2) = \mathbf{P'} \cdot \underline{\text{Prob}}(\phi_1\ U^{\leq k-1}\ \phi_2)$
  - requires k matrix-vector multiplications

- Note that we could express this in terms of matrix powers
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2) = (\mathbf{P'})^k \cdot \underline{\phi}_2$ and compute $(\mathbf{P'})^k$ in $\log_2 k$ steps
  - but this is actually inefficient: $(\mathbf{P'})^k$ is much less sparse than **P'**

PCTL bounded until – Example

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

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

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

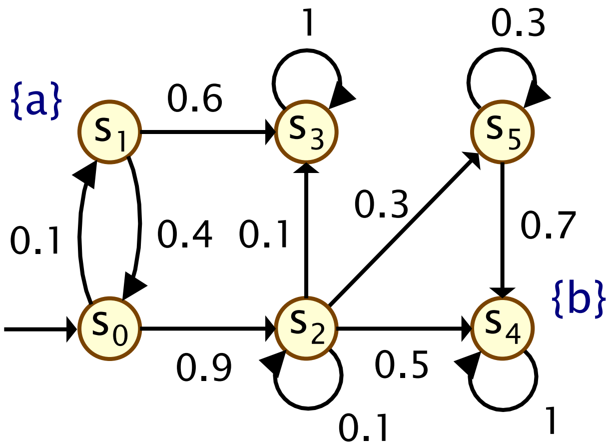
  - Sat($P_{>0.98}$ [$F^{\leq 2}$ succ]) = $\{s_1, s_3\}$

PCTL until for DTMCs

- Computation of probabilities Prob(s, ϕ_1 U ϕ_2) for all s ∈ 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

- Running example:

$P_{>0.8}$ [¬a U b]



Precomputation

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

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

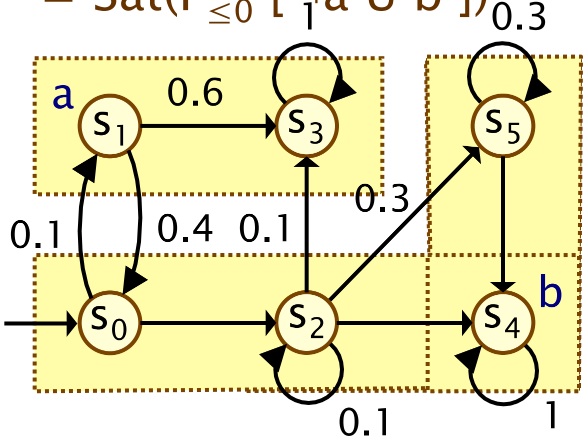
Precomputation – Prob0

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

S^{no} = Sat($P_{\leq 0}$ [¬a U b])

Example:

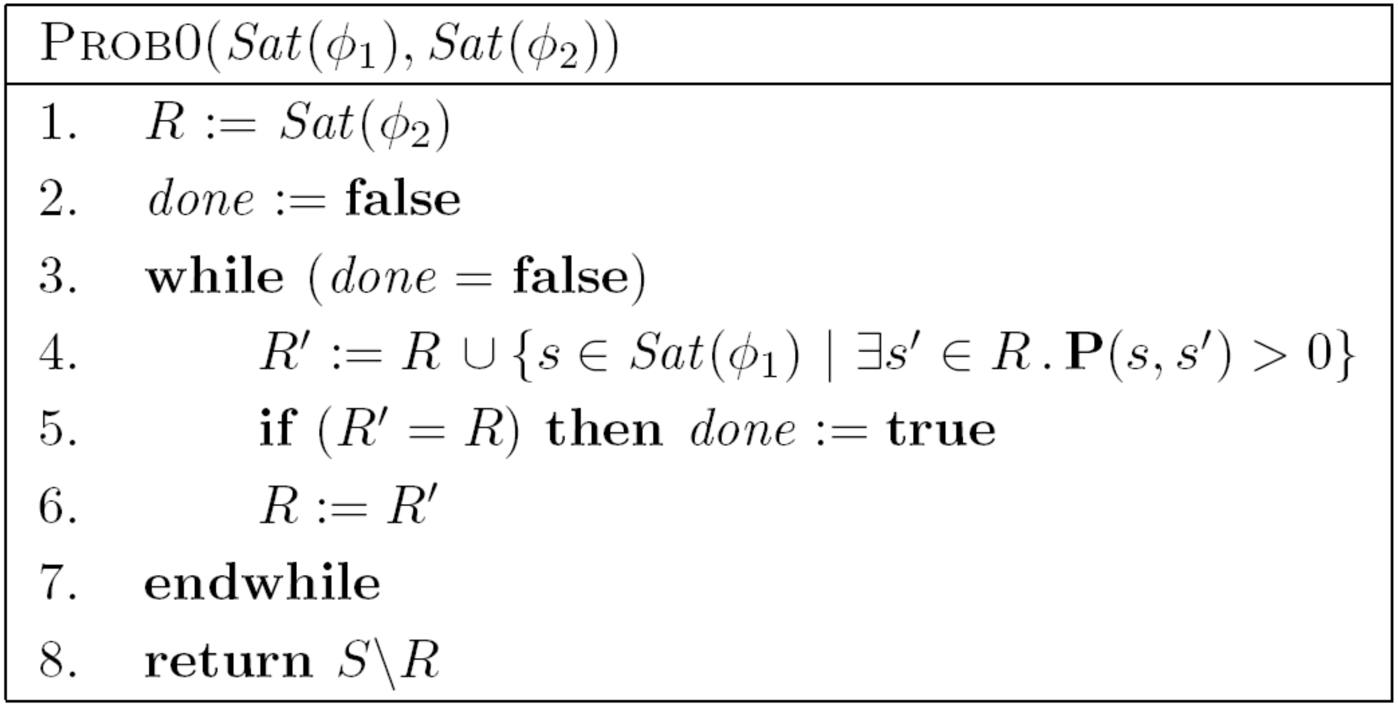
$P_{>0.8}$ [¬a U b]



Sat(b)

Sat($P_{>0}$ [¬a U b])

Prob0 algorithm



```
PROB0($Sat(\phi_1)$, $Sat(\phi_2)$)
R := $Sat(\phi_2)$
done := false
while (done = false)
    $R' := R \cup \{s \in Sat(\phi_1) \mid \exists s' \in R \,.\, \mathbf{P}(s,s') > 0\}$
    if ($R' = R$) then done := true
    $R := R'$
endwhile
return $S \backslash R$
```

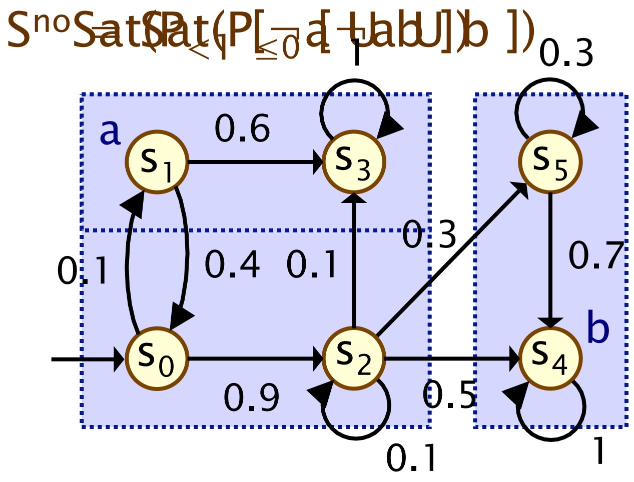
- Note: can be formulated as a least fixed point computation
    - also well suited to computation with binary decision diagrams

Precomputation – Prob1

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

Example:

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



$S^{yes} =$

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

Prob1 algorithm

```
PROB1($Sat(\phi_1), Sat(\phi_2), S^{no}$)
$R := S^{no}$
$done :=$ false
while ($done =$ false)
    $R' := R \cup \{s \in (Sat(\phi_1) \backslash Sat(\phi_2)) \mid \exists s' \in R \,.\, \mathbf{P}(s, s') > 0\}$
    if ($R' = R$) then $done :=$ true
    $R := R'$
endwhile
return $S \backslash R$
```

Prob 1 explanation

PCTL until – linear equations

- Probabilities Prob(s, ϕ_1 U ϕ_2) can now be obtained as the unique solution of the following set of linear equations
  - essentially the same as for probabilistic reachability

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

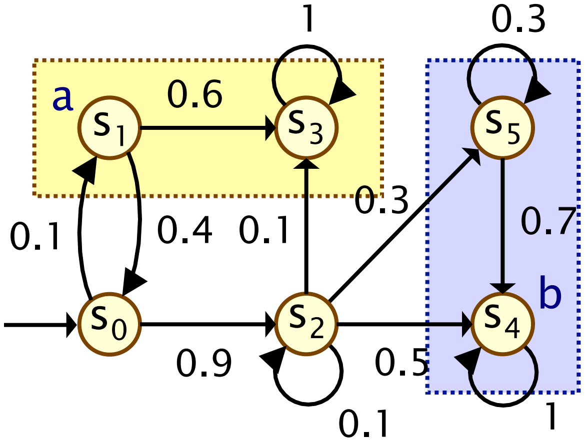
- Can also be reduced to a system in $|S^?|$ unknowns instead of $|S|$ where $S^? = S \setminus (S^{yes} \cup S^{no})$

PCTL until – linear equations

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

S^{no} =

Sat($P_{\leq 0}$ [¬a U b])



S^{yes} =

Sat($P_{\geq 1}$ [¬a U b])

$x_1 = x_3 = 0$

$x_4 = x_5 = 1$

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

$x_0 = 0.1x_1+0.9x_2 = 0.8$

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

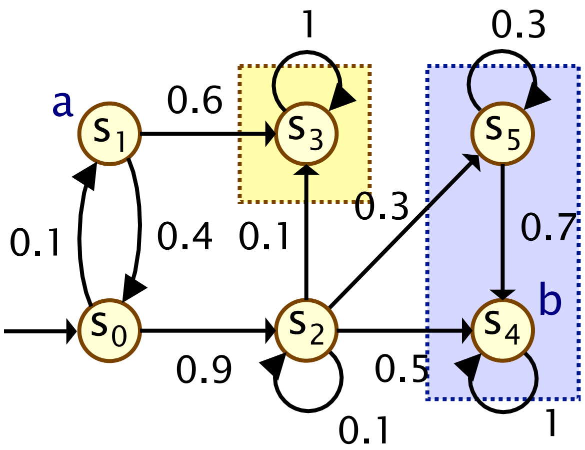
Sat($P_{>0.8}$ [¬a U b]) = { s_2,s_4,s_5 }

PCTL Until – Example 2

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

- Let x_i = Prob(s_i, F b)

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



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

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

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

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

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

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

Sat($P_{>0.5}$ [G¬b]) = { s_1,s_3 }

Linear equation systems

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

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

Iterative solution methods

- Start with an initial estimate for the vector $\underline{x}$, say $\underline{x}^{(0)}$
- Compute successive (increasingly accurate) approximations
  - approximation (solution vector) at kth iteration denoted $\underline{x}^{(k)}$
  - computation of $x^{(k)}$ uses values of $x^{(k-1)}$
- Terminate when solution vector has converged sufficiently
- Several possibilities for convergence criteria, e.g.:
  - maximum absolute difference

$$\max_i \left| \underline{x}^{(k)}(i) - \underline{x}^{(k-1)}(i) \right| \quad < \quad \varepsilon$$

  - maximum relative difference

$$\max_i \left(\frac{\left|\underline{x}^{(k)}(i) - \underline{x}^{(k-1)}(i)\right|}{\left|\underline{x}^{(k)}(i)\right|} \right) \quad < \quad \varepsilon$$

Jacobi method

- Based on fact that:

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

For probabilistic model checking, A(i,i) is always non-zero

- can be rearranged as:

$$\underline{x}(i) \quad = \quad \left(\underline{b}(i) - \sum_{j \neq i} \mathbf{A}(i,j) \cdot \underline{x}(j) \right) / \mathbf{A}(i,i)$$

- yielding this update scheme:

$$\underline{x}^{(k)}(i) \quad := \quad \left(\underline{b}(i) - \sum_{j \neq i} \mathbf{A}(i,j) \cdot \underline{x}^{(k-1)}(j) \right) / \mathbf{A}(i,i)$$

Gauss-Seidel

- The update scheme for Jacobi:

$$\underline{x}^{(k)}(i) \quad := \quad \left(\underline{b}(i) - \sum_{j \neq i} \mathbf{A}(i,j) \cdot \underline{x}^{(k-1)}(j) \right) / \mathbf{A}(i,i)$$

- can be improved by using the most up-to-date values of $\underline{x}^{(j)}$ that are available
- This is the Gauss-Seidel method:

$$\underline{x}^{(k)}(i) \quad := \quad \left(\underline{b}(i) - \sum_{j<i} \mathbf{A}(i,j) \cdot \underline{x}^{(k)}(j) - \sum_{j>i} \mathbf{A}(i,j) \cdot \underline{x}^{(k-1)}(j) \right) / \mathbf{A}(i,i)$$

Over-relaxation

- Over-relaxation:
  - compute new values with existing schemes (e.g. Jacobi)
  - but use weighted average with previous vector

- Example: Jacobi + over-relaxation

$$\begin{aligned}\underline{x}^{(k)}(i) \quad := \quad & (1-\omega)\cdot\underline{x}^{(k-1)}(i) \\ & + \;\; \omega\cdot\left(\underline{b}(i)-\sum_{j\neq i}\mathbf{A}(i,j)\cdot\underline{x}^{(k-1)}(j)\right)/\mathbf{A}(i,i)\end{aligned}$$

- where ω ∈ (0,2) is a parameter to the algorithm

Comparison

- Gauss-Seidel typically outperforms Jacobi
  - i.e. faster convergence
  - also: only need to store a single solution vector
- Both Gauss-Seidel and Jacobi usually outperform the Power method (see least fixed point method from Lecture 2)
- However Power method has guaranteed convergence
  - Jacobi and Gauss-Seidel do not
- Over-relaxation methods may converge faster
  - for well chosen values of ω
  - need to rely on heuristics for this selection

Model checking complexity

- Model checking of DTMC $(S, s_{init}, \mathbf{P}, L)$ against PCTL formula Φ complexity is linear in $|\Phi|$ and polynomial in $|S|$

- Size $|\Phi|$ of Φ is defined as number of logical connectives and temporal operators plus sizes of temporal operators
  - model checking is performed for each operator

- Worst-case operator is $P_{\sim p}$ [Φ_1 U Φ_2]
  - main task: solution of linear equation system of size $|S|$
  - can be solved with Gaussian elimination: cubic in $|S|$
  - and also precomputation algorithms (max $|S|$ steps)
- Strictly speaking, $U^{\leq k}$ could be worse than U for large k
  - but in practice k is usually small

Summing up…

- Model checking a PCTL formula ϕ on a DTMC
  - i.e. determine set Sat(ϕ)
  - recursive: bottom-up traversal of parse tree of ϕ

- Atomic propositions and logical connectives: trivial

- Key part: computing probabilities for $P_{\sim p}$ […] formulae
  - X Φ : one matrix-vector multiplications
  - $Φ_1$ $U^{\leq k}$ $Φ_2$: k matrix-vector multiplications
  - $Φ_1$ U $Φ_2$: graph-based precomputation algorithms + solution of linear equation system in at most |S| variables

- Iterative methods for solving large linear equation systems

Lecture 6
PRISM

Dr. Dave Parker

Department of Computer Science
University of Oxford

Practicals

- 4 practical exercises
- 4 scheduled 2 hour practical sessions:
  - Tuesday 4-6pm, room 379, weeks 3, 4, 6 and 7
  - demonstrator: Aistis Simaitis

- Note:
  - you will also be expected to complete some of the practical work outside these hours
  - final assignment will include practical (PRISM) exercises

http://www.prismmodelchecker.org/courses/pmc1112/

Overview

- Tool support for probabilistic model checking
  - motivation, existing tools

- The PRISM model checker
  - functionality, features
  - modelling language
  - property specification

- Running example
  - leader election protocol

- PRISM tool demo

Motivation

- Complexity of PCTL model checking
  - generally polynomial in model size (number of states)

- State space explosion problem
  - models for realistic case studies are typically huge

- Clearly (efficient) tool support is required

- Benefits:
  - fully automated process
  - high-level languages/formalisms for building models
  - visualisation of quantitative results

Probabilistic model checkers

- PRISM (this lecture): DTMCs, MDPs, CTMCs, PTAs + rewards
- Markov chain model checkers
  - MRMC: DTMCs, CTMCs + reward extensions
  - PEPA toolset: CTMCs + CSL
- Markov decision process (MDP) tools
  - LiQuor: LTL verification for MDPs (Probmela language)
  - RAPTURE: prototype for abstraction/refinement of MDPs
  - ProbDiVinE: parallel/distributed LTL model checking of MDPs
- Simulation-based probabilistic model checking:
  - APMC, Ymer (both based on PRISM language), VESTA
- And more
  - APNN-Toolbox, SMART, CADP, Möbius, PASS, PARAM, …
  - see: http://www.prismmodelchecker.org/other-tools.php

The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL)
  - versions for Linux, Unix, Mac OS X, Windows, 64-bit OSs

- Modelling of:
  - DTMCs, CTMCs, MDPs + costs/rewards
  - probabilistic timed automata (PTAs) (not covered here)

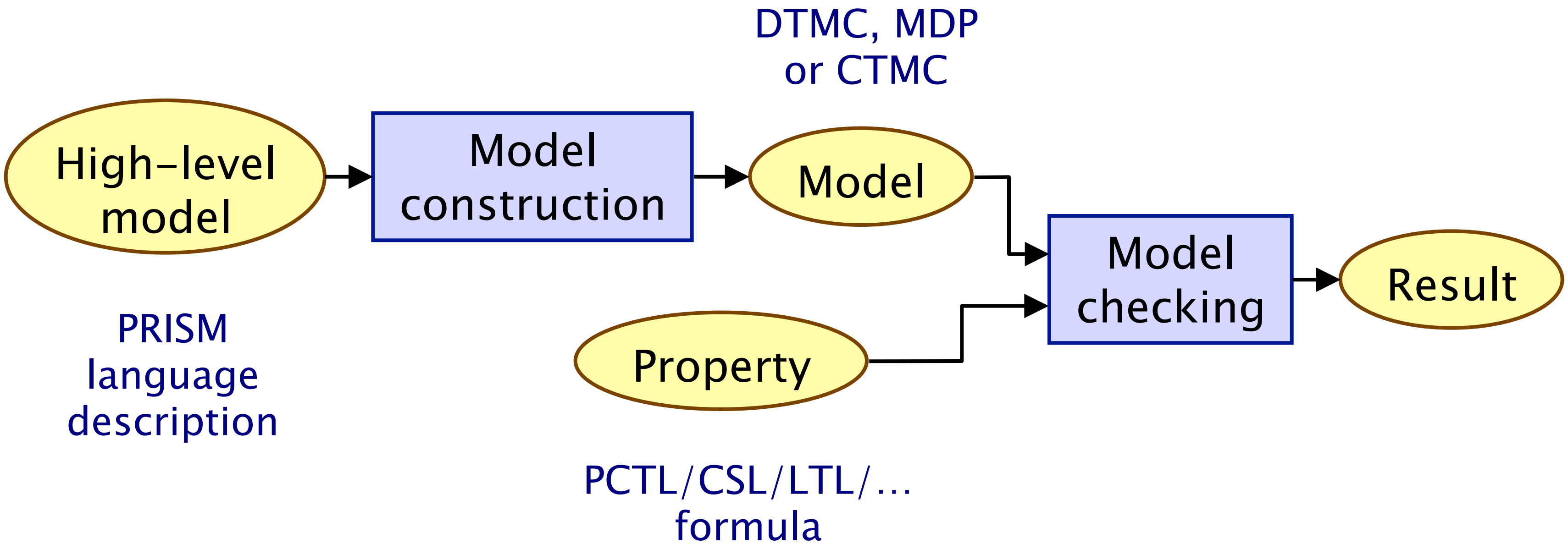
- Model checking of:
  - PCTL, CSL, LTL, PCTL* + extensions + costs/rewards

PRISM functionality

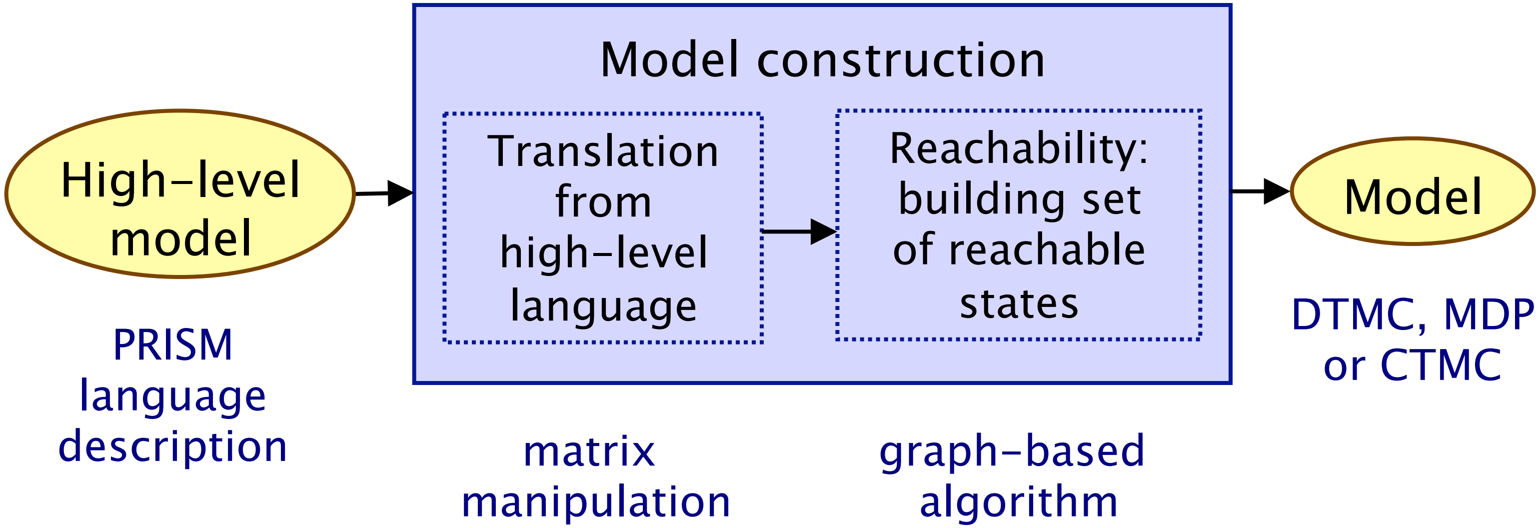
- High-level modelling language
- Wide range of model analysis methods
  - efficient symbolic implementation techniques
  - also: approximate verification using simulation + sampling
- Graphical user interface
  - model/property editor
  - discrete-event simulator - model traces for debugging, etc.
  - easy automation of verification experiments
  - graphical visualisation of results
- Command-line version
  - same underlying verification engines
  - useful for scripting, batch jobs

Probabilistic model checking

- Overview of the probabilistic model checking process
  - two distinct phases: model construction, model checking



Model construction



High-level model

PRISM language description

Model construction

Translation from high-level language

matrix manipulation

Reachability: building set of reachable states

graph-based algorithm

Model

DTMC, MDP or CTMC

Modelling languages/formalisms

- Many high-level modelling languages, formalisms available

- For example:
  - probabilistic/stochastic process algebras
  - stochastic Petri nets
  - stochastic activity networks

- Custom languages for tools, e.g.:
  - PRISM modelling language
  - Probmela (probabilistic variant of Promela, the input language for the model checker SPIN) – used in LiQuor

PRISM modelling language

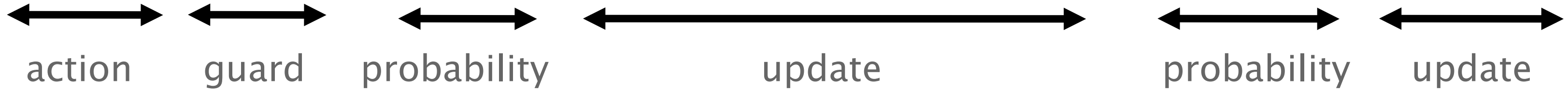
- Simple, textual, state-based language
  - modelling of DTMCs, CTMCs, MDPs, …
  - based on Reactive Modules [AH99]

- Basic components…
- Modules:
  - components of system being modelled
  - composed in parallel
- Variables
  - finite (integer ranges or Booleans)
  - local or global
  - all variables public: anyone can read, only owner can modify

PRISM modelling language

- Guarded commands
  - describe behaviour of each module
  - i.e. the changes in state that can occur
  - labelled with probabilities (or, for CTMCs, rates)
  - (optional) action labels

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



PRISM modelling language

- Parallel composition
  - model multiple components that can execute independently
  - for DTMC models, mostly assume components operate synchronously, i.e. move in lock-step
- Synchronisation
  - simultaneous transitions in more than one module
  - guarded commands with matching action-labels
  - probability of combined transition is product of individual probabilities for each component
- More complex parallel compositions can be defined
  - using process-algebraic operators
  - other types of parallel composition, action hiding/renaming

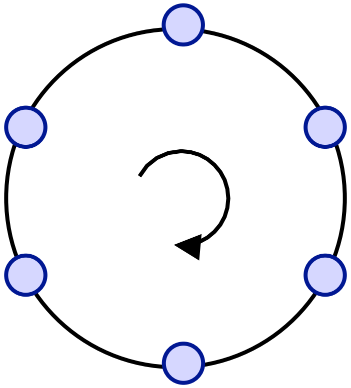
Simple example

```
module M1
    x : [0..3] init 0;
    [a] x=0 -> (x'=1);
    [b] x=1 -> 0.5:(x'=2) + 0.5:(x'=3);
endmodule
```

```
module M2
    y : [0..3] init 0;
    [a] y=0 -> (y'=1);
    [b] y=1 -> 0.4:(y'=2) + 0.6:(y'=3);
endmodule
```

Example: Leader election

- Randomised leader election protocol
  - due to Itai & Rodeh (1990)
- Set-up: N nodes, connected in a ring
  - communication is synchronous (lock-step)
- Aim: elect a leader
  - i.e. one uniquely designated node
  - by passing messages around the ring



- Protocol operates in rounds. In each round:
  - each node choose a (uniformly) random id ∈ {0,…,k-1}
  - (k is a parameter of the protocol)
  - all nodes pass their id around the ring
  - if there is (maximum) unique id, node with this id is the leader
  - if not, try again with a new round

PRISM code

PRISM property specifications

- Based on (probabilistic extensions of) temporal logic
  - incorporates PCTL, CSL, LTL, PCTL*
  - also includes: quantitative extensions, costs/rewards

- Leader election properties
  - $P_{\geq 1}$ [F elected]
    - with probability 1, a leader is eventually elected
  - $P_{>0.8}$ [$F^{\leq k}$ elected]
    - with probability greater than 0.8, a leader is elected within k steps

- Usually focus on quantitative properties:
  - $P_{=?}$ [$F^{\leq k}$ elected]
    - what is the probability that a leader is elected within k steps?

PRISM property specifications

- Best/worst-case scenarios
  - combining “quantitative” and “exhaustive” aspects

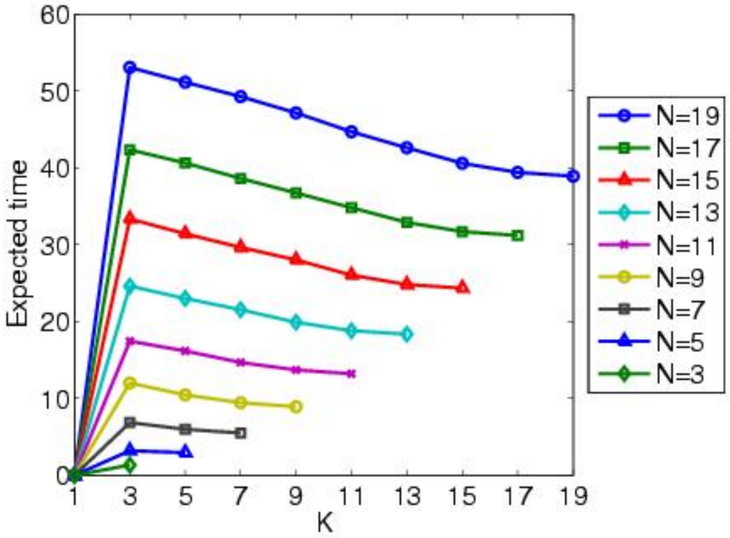
- e.g. computing values for a range of states…

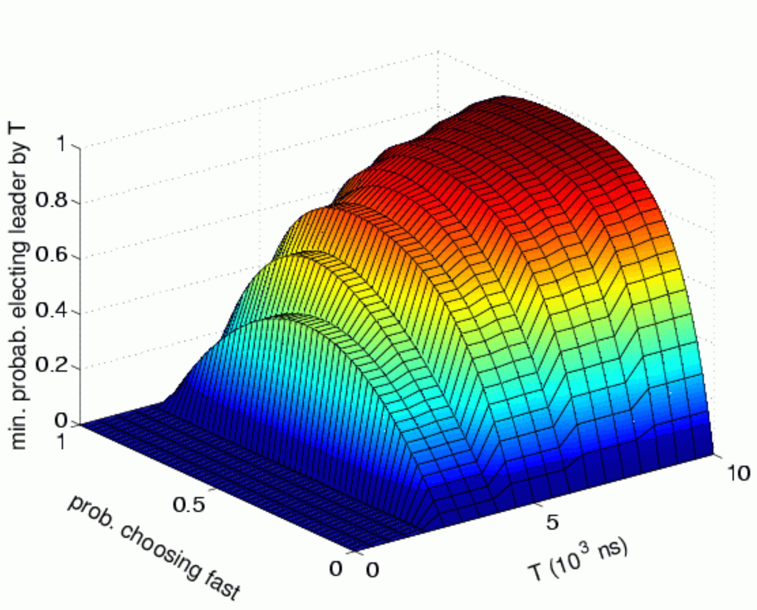
- $P_{=?}$ [$F^{\leq t}$ elected {tokens$\leq$k}{min}] –
  - “minimum probability of the leader election algorithm completing within t steps from any state where there are at most k tokens”

- $R_{=?}$ [F end {“init”}{max}] –
  - “maximum expected run-time over all possible initial configurations”

PRISM property specifications

- Experiments:
  - ranges of model/property parameters
  - e.g. $P_{=?}$ [$F^{\leq T}$ error] for N=1..5, T=1..100
    where N is some model parameter and T a time bound
  - identify patterns, trends, anomalies in quantitative results





PRISM…

More info on PRISM

- PRISM website: http://www.prismmodelchecker.org/
  - tool download: binaries, source code (GPL)
  - on-line example repository (50+ case studies)
  - on-line documentation:
    - PRISM manual
    - PRISM tutorial
  - support: help forum, bug tracking, feature requests
  - related publications, talks, tutorials, links
- Course practicals info at:
  - http://www.prismmodelchecker.org/courses/pmc1112/

Lecture 7
Costs & Rewards

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- Specifying costs and rewards
    - DTMCs
    - PRISM language
- Properties: expected reward values
    - instantaneous
    - cumulative
    - reachability
    - temporal logic extensions
- Model checking
    - computing reward values
- Case study
    - randomised contract signing

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 used here: 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
  - e.g. the expected value of the reward at some time point

- Cumulative properties
  - e.g. the expected cumulated reward over some period

DTMC reward structures

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

- Example (for use with instantaneous properties)
  - "size of message queue": $\underline{\rho}$ maps each state to the number of jobs in the queue in that state, ι is not used
- Examples (for use with cumulative properties)
  - "time-steps": $\underline{\rho}$ returns 1 for all states and ι is zero (equivalently, $\underline{\rho}$ is zero and ι returns 1 for all transitions)
  - "number of messages lost": $\underline{\rho}$ is zero and ι maps transitions corresponding to a message loss to 1
  - "power consumption": $\underline{\rho}$ is defined as the per-time-step energy consumption in each state and ι as the energy cost of each transition

Rewards in the PRISM language

```
rewards “total_queue_size”
   true : queue1+queue2;
endrewards
```

(instantaneous, state rewards)

```
rewards “time”
   true : 1;
endrewards
```

(cumulative, state rewards)

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

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

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

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

Expected reward properties

- Expected (“average”) values of rewards…

- Instantaneous
  - “the expected value of the state reward at time-step k”
  - e.g. “the expected queue size after exactly 90 seconds”

- Cumulative (time-bounded)
  - “the expected reward cumulated up to time-step k”
  - e.g. “the expected power consumption over one hour”
- Reachability (also cumulative)
  - “the expected reward cumulated before reaching states T⊆S”
  - e.g. “the expected time for the algorithm to terminate”

Expectation

- Probability space $(\Omega, \Sigma, \Pr)$
  - probability measure $\Pr : \Sigma \rightarrow [0,1]$
- Random variable X
  - a measurable function $X : \Omega \rightarrow \Delta$
  - usually real-valued, i.e.: $X : \Omega \rightarrow \mathbb{R}$
- Expected (“average”) value of the random variable: Exp(X)

$$\mathrm{Exp}(X) = \sum_{\omega\in\Omega} X(\omega)\cdot \Pr(\omega)$$

discrete case

$$\mathrm{Exp}(X) = \int_{\omega\in\Omega} X(\omega)\, d\Pr$$

Reachability + rewards

- Expected reward cumulated before reaching states $T \subseteq S$
- Define a random variable:
  - $X_{Reach(T)} : Path(s) \rightarrow \mathbb{R}_{\geq 0}$
  - where for an infinite path $\omega = s_0 s_1 s_2 \ldots$

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

  - where $k_T = \min\{ j \mid s_j \in T \}$
- Then define:
  - $ExpReach(s, T) = Exp(s, X_{Reach(T)})$
  - denoting: expectation of the random variable $X_{Reach(T)}$ with respect to the probability measure Pr_s, i.e.:

$$\int_{\omega \in Path(s)} X_{Reach(T)}(\omega) \, dPr_s$$

Computing the rewards

- Determine states for which ProbReach(s, T) = 1

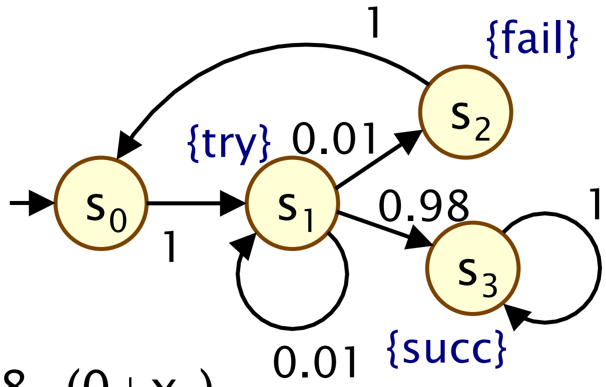
- Solve linear equation system:

  - ExpReach(s, T) =

$$\begin{cases} \infty & \text{if ProbReach(s, T)} < 1 \\ 0 & \text{if } s \in T \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s,s') \cdot \left(\iota(s,s') + \text{ExpReach}(s', T) \right) & \text{otherwise} \end{cases}$$

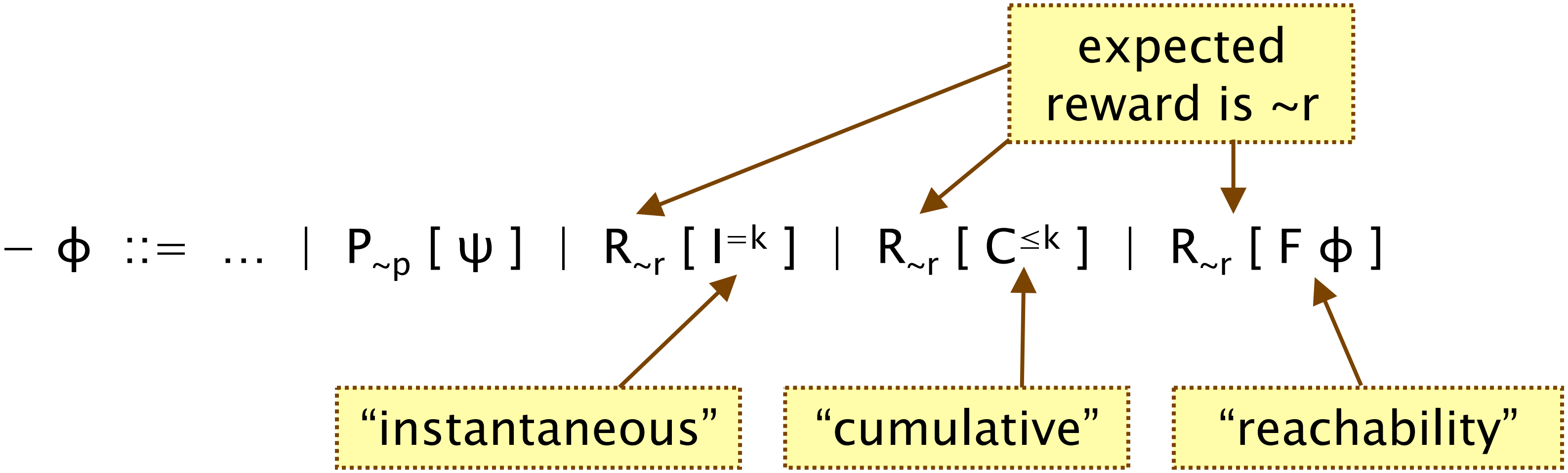
Example

- Let $\underline{\rho}$ = [0, 1, 0, 0] and $\iota(s,s') = 0$ for all $s,s' \in S$
- Compute ExpReach(s_0, {s_3})
  - (“expected number of times pass through s_1 to get to s_3”)
- First check:
  - $\underline{\text{ProbReach}}(\{s_3\}) = \{ 1, 1, 1, 1 \}$
- Then solve linear equation system:
  - (letting x_i = ExpReach(s_i, {s_3})):
  - $x_0 = 0 + 1 \cdot (0 + x_1)$
  - $x_1 = 1 + 0.01 \cdot (0+x_2)+0.01 \cdot (0+x_1)+0.98 \cdot (0+x_3)$
  - $x_2 = 0 + 1 \cdot (0 + x_0)$
  - $x_3 = 0$
  - Solution: $\underline{\text{ExpReach}}(\{s_3\}) = [\ 100/98, 100/98, 100/98, 0\ ]$
- So: ExpReach(s_0, {s_3}) = 100/98 ≈ 1.020408



Specifying reward properties

- PRISM extends PCTL to include expected reward 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”

Random variables for reward formulae

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

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

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

$X_{F\phi}$ same as $X_{Reach(Sat(\phi))}$ from earlier

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

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

Reward formula semantics

- Formal semantics of the three reward operators:

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

$Exp(s, X_{F\Phi})$
same as
$ExpReach(s, Sat(\Phi))$
from earlier

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

- We can also define $R_{=?}$ [...] properties, as for the P operator
  - e.g. $R_{=?} [\ F\ \Phi\ ]$ returns the value $Exp(s, X_{F\Phi})$

Model checking reward operators

- Like for model checking $P_{\sim p}$ [...], to check $R_{\sim r}$ [...]
  - compute reward values for all states, compare with bound r

- Instantaneous: $R_{\sim r}$ [$I^{=k}$] – compute Exp($X_{I=k}$)
  - solution of recursive equations
  - essentially: k matrix-vector multiplications
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$] – compute Exp($X_{C\leq k}$)
  - solution of recursive equations
  - essentially: k matrix-vector multiplications
- Reachability: $R_{\sim r}$ [F ϕ] – compute Exp($X_{F\phi}$)
  - graph analysis + linear equation system
  - (see computation of ExpReach(s, T) earlier)

Model checking
R operator
same complexity
as for P operator

Model checking $R_{\sim r}$ [$I^{=k}$]

- Expected instantaneous reward at step k
  - can be defined in terms of transient probabilities for step k

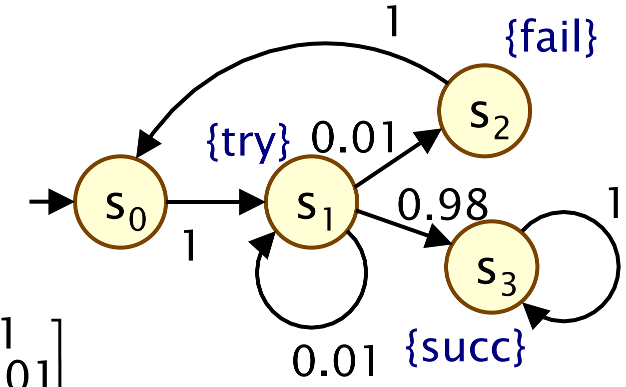
- $\text{Exp}(s, X_{I=k}) = \Sigma_{s' \in S}\ \pi_{s,k}(s') \cdot \underline{\rho}(s')$

- $\underline{\text{Exp}}(X_{I=k}) = \mathbf{P}^k \cdot \underline{\rho}$

- Yielding recursive definition:
  - $\underline{\text{Exp}}(X_{I=0}) = \underline{\rho}$
  - $\underline{\text{Exp}}(X_{I=k}) = \mathbf{P} \cdot \underline{\text{Exp}}(X_{I=(k-1)})$
  - i.e. k matrix-vector multiplications
  - note: “backwards” computation (like bounded until prob.s) rather than “forwards” computation (like transient prob.s)

Example

- Let $\underline{\rho} = [\ 0, 1, 0, 0\ ]$ and $\iota(s,s') = 0$ for all $s,s' \in S$
- Compute $\mathrm{Exp}(s_0, X_{I=2})$
  - (“probability of being in state s_1”)
  - $\mathrm{Exp}(X_{I=0}) = [\ 0, 1, 0, 0\ ]$
  - $\mathrm{Exp}(X_{I=1}) = \mathbf{P} \cdot \mathrm{Exp}(X_{I=0})$



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

  - $\mathrm{Exp}(X_{I=2}) = \mathbf{P} \cdot \mathrm{Exp}(X_{I=1})$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0.01 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.0001 \\ 1 \\ 0 \end{bmatrix}$$

- Result: $\mathrm{Exp}(s_0, X_{I=2}) = 0.01$

Model checking $R_{\sim r}$ [$C^{\leq k}$]

- Expected reward cumulated up to time-step k

- Again, a recursive definition:

$$\mathrm{Exp}(s,\ X_{C\leq k}) \ = \begin{cases} 0 & \text{if } k = 0 \\ \underline{\rho}(s) + \sum_{s' \in S} P(s,s') \cdot \big(\iota(s,s') + \mathrm{Exp}(s',\ X_{C\leq k-1})\big) & \text{if } k > 0 \end{cases}$$

- And in matrix/vector notation:

$$\underline{\mathrm{Exp}}(X_{C\leq k}) \ = \begin{cases} 0 & \text{if } k = 0 \\ \underline{\rho} + (P \bullet \iota) \cdot \underline{1} + P \cdot \underline{\mathrm{Exp}}(X_{C\leq k-1}) & \text{if } k > 0 \end{cases}$$

  - where • denotes Schur (pointwise) matrix multiplication
  - and $\underline{1}$ is a vector of all 1s

Case study: Contract signing

- Two parties want to agree on a contract
  - each will sign if the other will sign, but do not trust each other
  - there may be a trusted third party (judge)
  - but it should only be used if something goes wrong

- In real life: contract signing with pen and paper
  - sit down and write signatures simultaneously

- On the Internet…
  - how to exchange commitments on an asynchronous network?
  - "partial secret exchange protocol" [EGL85]

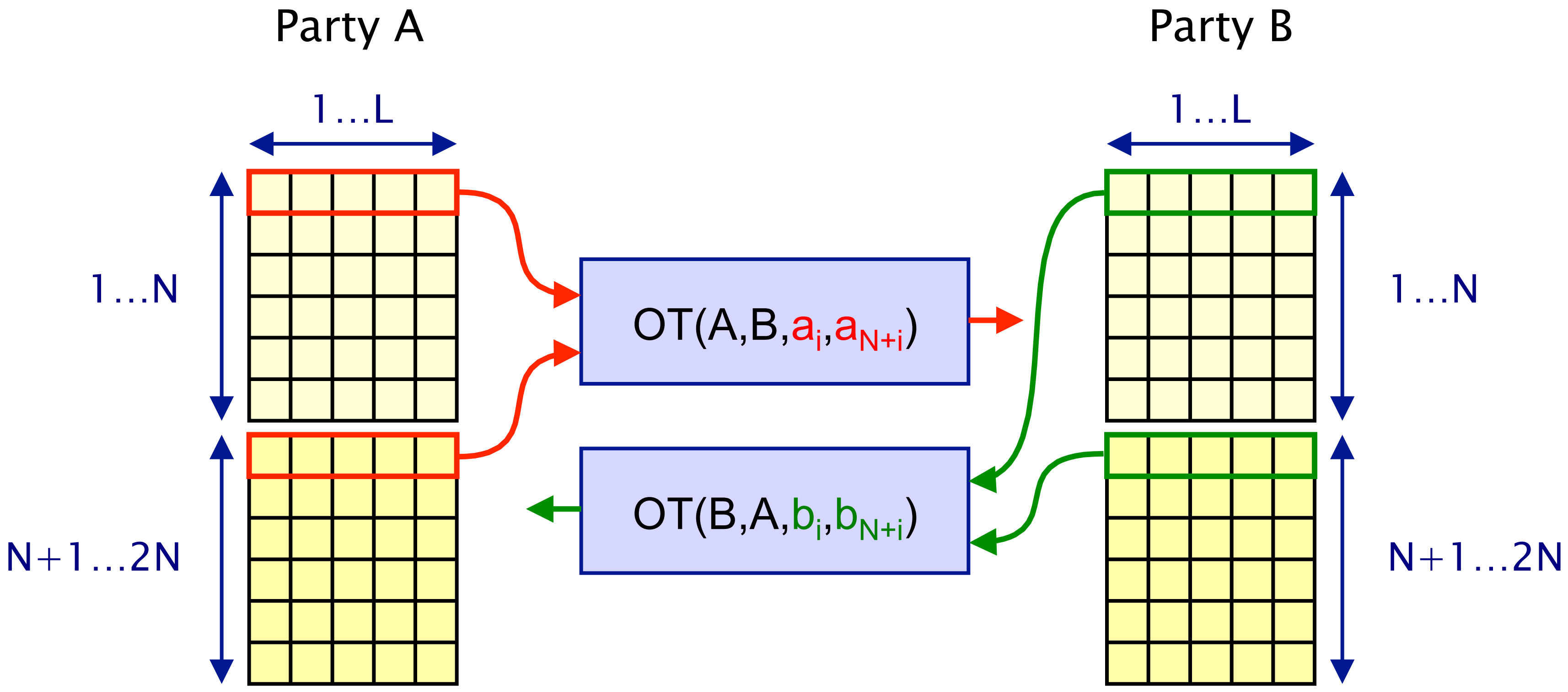
Contract signing – EGL protocol

- Partial secret exchange protocol for 2 parties (A and B)

- A (B) holds 2N secrets $a_1,\ldots,a_{2N}$ $(b_1,\ldots,b_{2N})$
  - a secret is a binary string of length L
  - secrets partitioned into pairs: e.g. $\{ (a_i, a_{N+i}) \mid i=1,\ldots,N \}$
  - A (B) committed if B (A) knows one of A's (B's) pairs

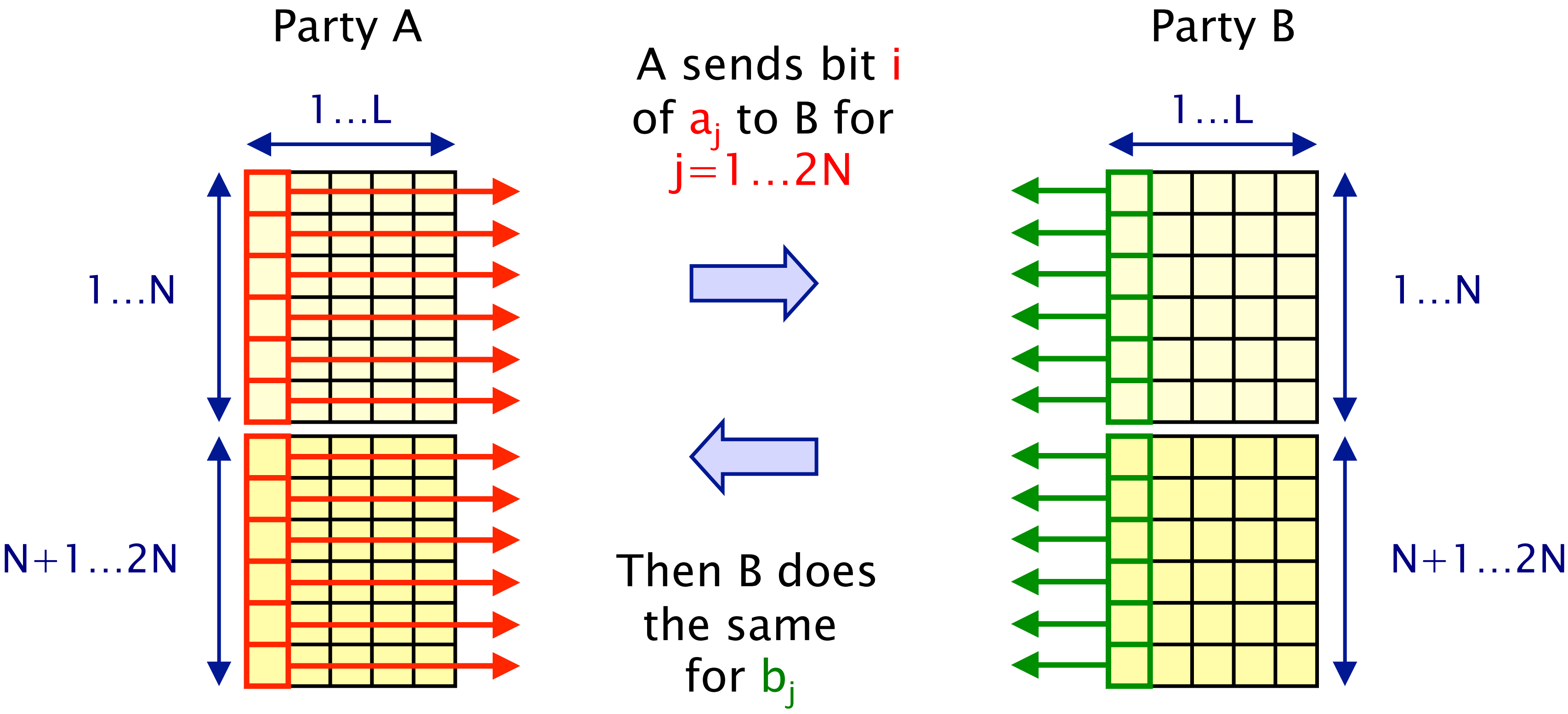
- Uses "1-out-of-2 oblivious transfer protocol" OT(S,R,x,y)
  - Sender S sends x and y to receiver R
  - R receives x with probability ½ otherwise receives y
  - S does not know which one R receives
  - if S cheats then R can detect this with probability ½

EGL protocol – Step 1



(repeat for i=1…N)

EGL protocol – Step 2



A sends bit i of a_j to B for j=1…2N

Then B does the same for b_j

(repeat for i=1…L)

Contract signing – Results

- Modelled in PRISM as a DTMC (no concurrency) [NS06]

- Highlights a weakness in the protocol
  - party B can act maliciously by quitting the protocol early
  - this behaviour not considered in the original analysis

- PRISM analysis shows
  - if B stops participating in the protocol as soon as he/she has obtained one of A pairs, then, with probability 1, at this point:
    - B possesses a pair of A's secrets
    - A does not have complete knowledge of any pair of B's secrets
  - protocol is not fair under this attack:
  - B has a distinct advantage over A

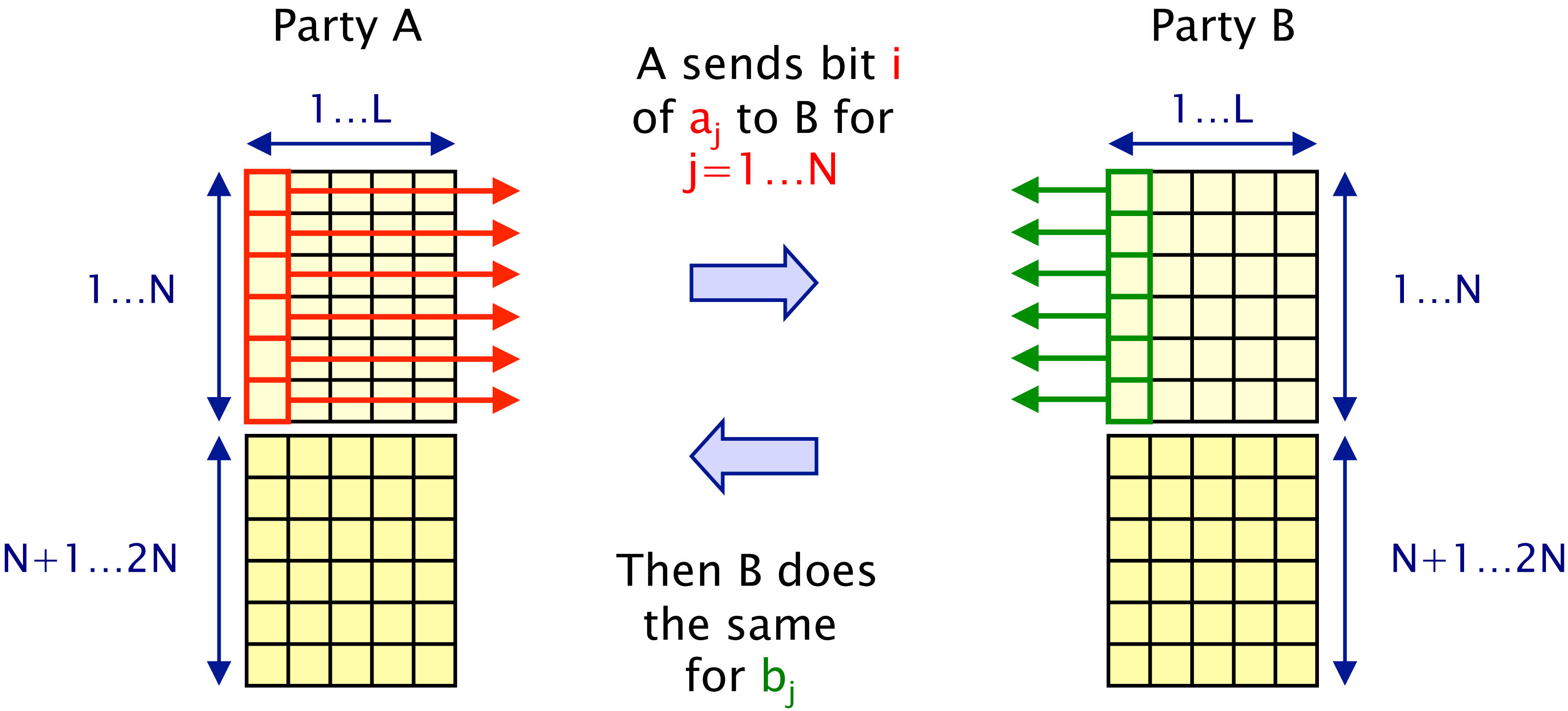
Contract signing – Results

- The protocol is unfair because in step 2:
  - A sends a bit for each of its secret before B does
- Can we make this protocol fair by changing the message sequence scheme?
- Since the protocol is asynchronous the best we can hope for is:
  - B (or A) has this advantage with probability ½
- We consider 3 possible alternative message sequence schemes (EGL2, EGL3, EGL4)

Contract signing – EGL2

```
(step 1)
…
(step 2)
for ( i=1,…,L )
    for ( j=1,…,N )  A transmits bit i of secret $a_j$ to B
    for ( j=1,…,N )  B transmits bit i of secret $b_j$ to A
    for ( j=N+1,…,2N )  A transmits bit i of secret $a_j$ to B
    for ( j=N+1,…,2N )  B transmits bit i of secret $b_j$ to A
```

Modified step 2 for EGL2



A sends bit i of a_j to B for j=1…N

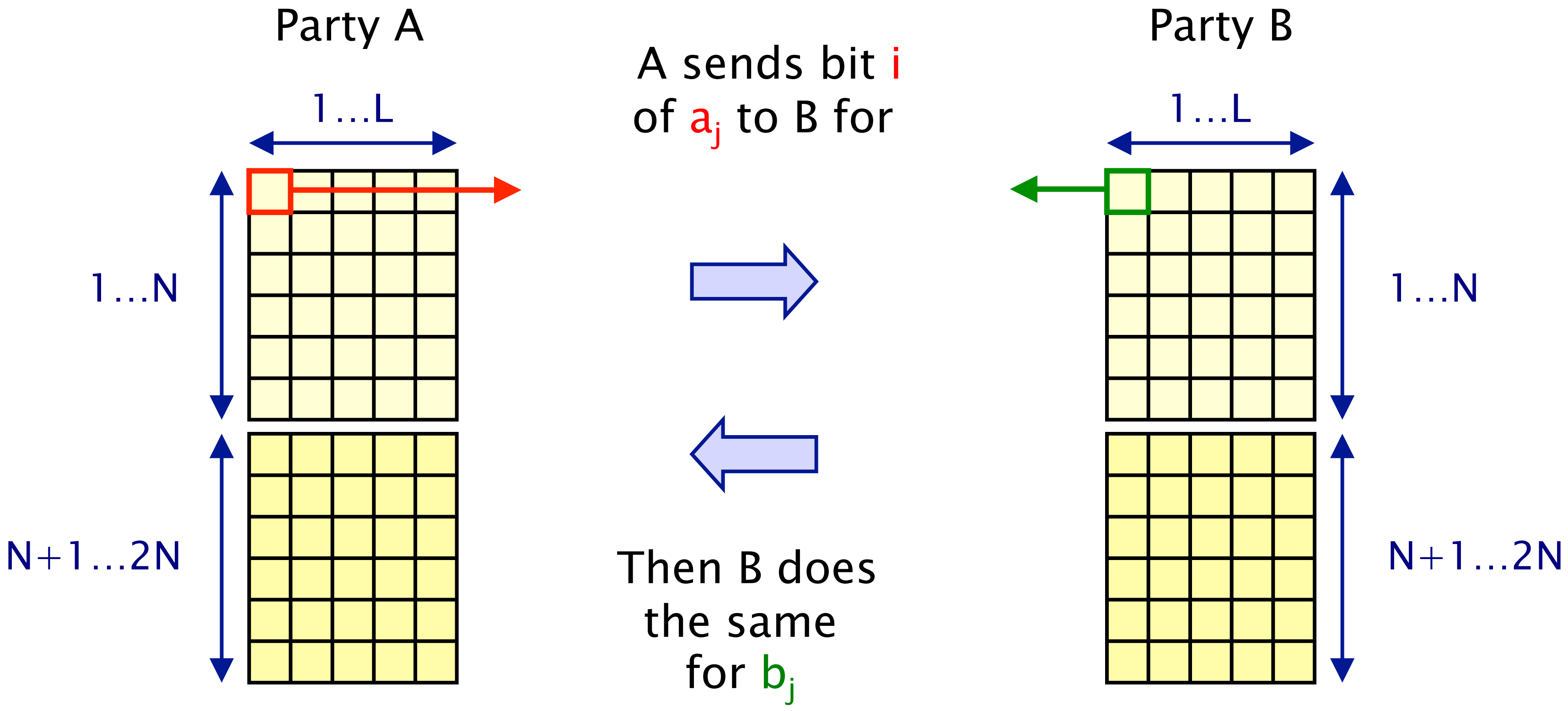
Then B does the same for b_j

(after j=1…N, send j=N+1…2N)
(then repeat for i=1…L)

Contract signing – EGL3

```
(step 1)
…
(step 2)
for ( i=1,…,L )  for ( j=1,…,N )
    A transmits bit i of secret $a_j$ to B
    B transmits bit i of secret $b_j$ to A
for ( i=1,…,L )  for ( j=N+1,…,2N )
    A transmits bit i of secret $a_j$ to B
    B transmits bit i of secret $b_j$ to A
```

Modified step 2 for EGL3



Party A

Party B

A sends bit i of a_j to B for

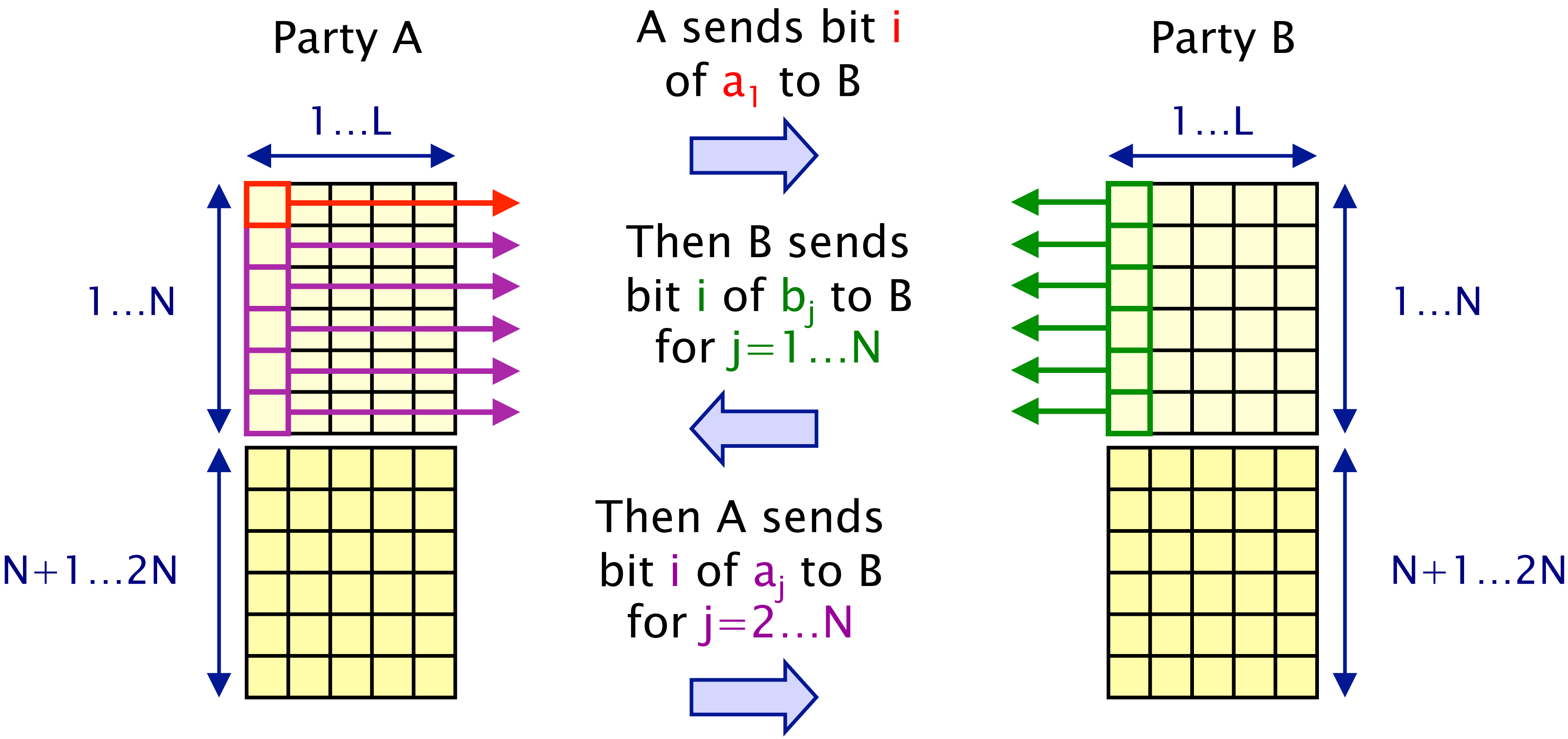
Then B does the same for b_j

(repeat for j=1…N and for i=1…L)
(then send j=N+1…2N for i=1…L)

Contract signing – EGL4

```
(step 1)
…
(step 2)
for ( i=1,…,L )
    A transmits bit i of secret $a_1$ to B
    for ( j=1,…,N )  B transmits bit i of secret $b_j$ to A
    for ( j=2,…,N )  A transmits bit i of secret $a_j$ to B
for ( i=1,…,L )
    A transmits bit i of secret $a_{N+1}$ to B
    for ( j=N+1,…,2N )  B transmits bit i of secret $b_j$ to A
    for ( j=N+2,…,2N )  A transmits bit i of secret $a_j$ to B
```

Modified step 2 for EGL4



A sends bit i of a_1 to B

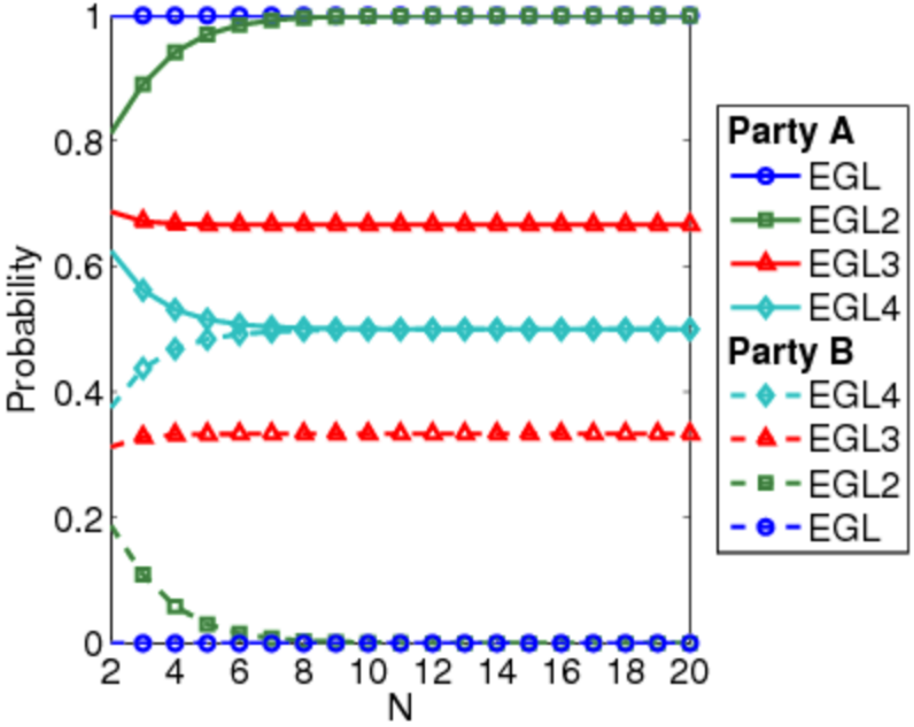
Then B sends bit i of b_j to B for j=1…N

Then A sends bit i of a_j to B for j=2…N

(repeat for i=1…L)
(then send j=N+1…2N in same fashion)

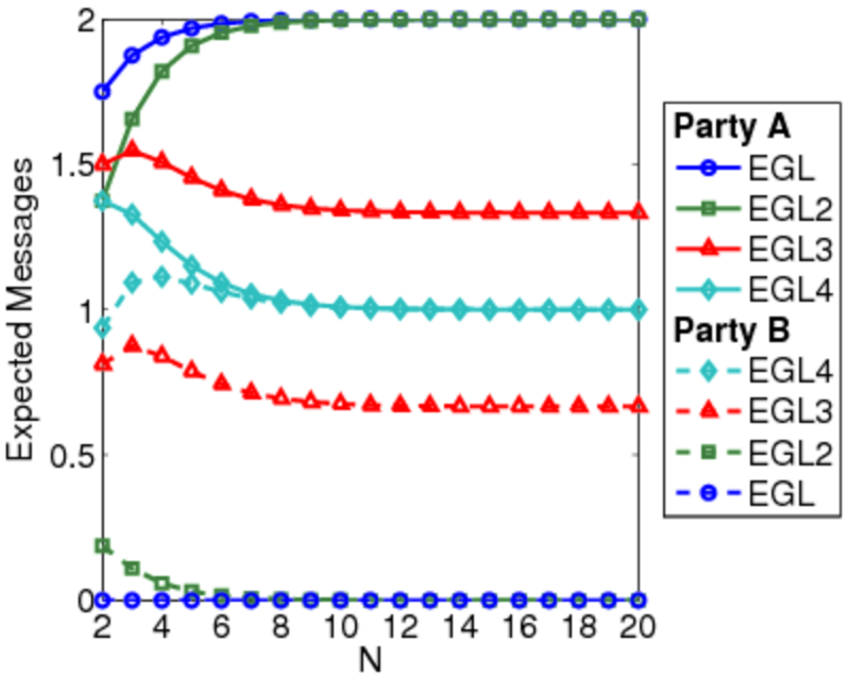
Contract signing – Results

- The chance that the protocol is unfair
  - probability that one party gains knowledge first
  - $P_{=?}$ [F $\text{know}_B \wedge \neg\text{know}_A$] and $P_{=?}$ [F $\text{know}_A \wedge \neg\text{know}_B$]



Contract signing – Results

- The influence that each party has on the fairness
  - once a party knows a pair, the expected number of messages from this party required before the other party knows a pair



R=? [F know$_A$]

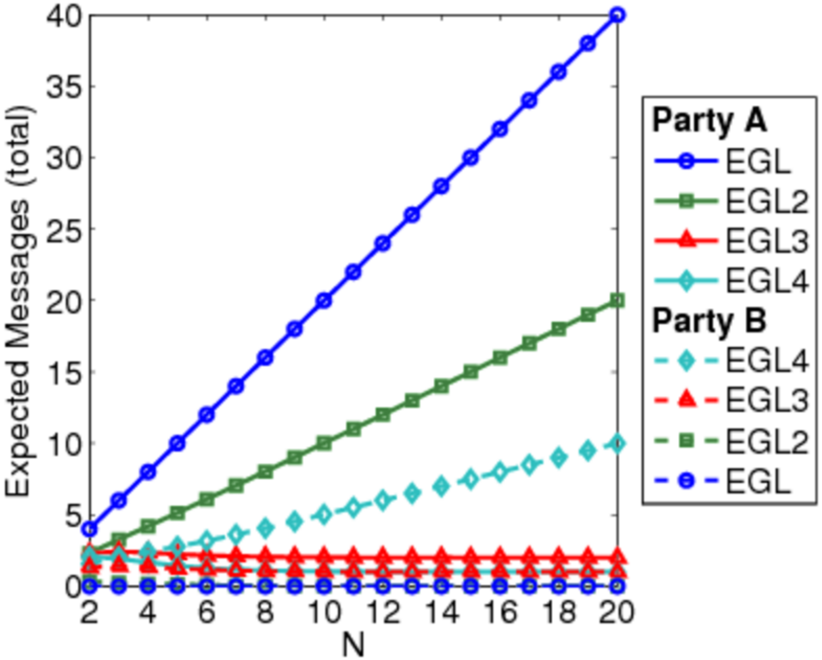
Reward structure:

Assign 1 to transitions corresponding to messages being sent from B to A after B knows a pair

(and 0 to all other transitions)

Contract signing – Results

- The duration of unfairness of the protocol
  - once a party knows a pair, the expected total number of messages that need to be sent before the other knows a pair



R=? [F know_A]

Reward structure:

Assign 1 to transitions corresponding to any message being sent between A and B after B knows a pair

(and 0 to all other transitions)

Contract signing – Results

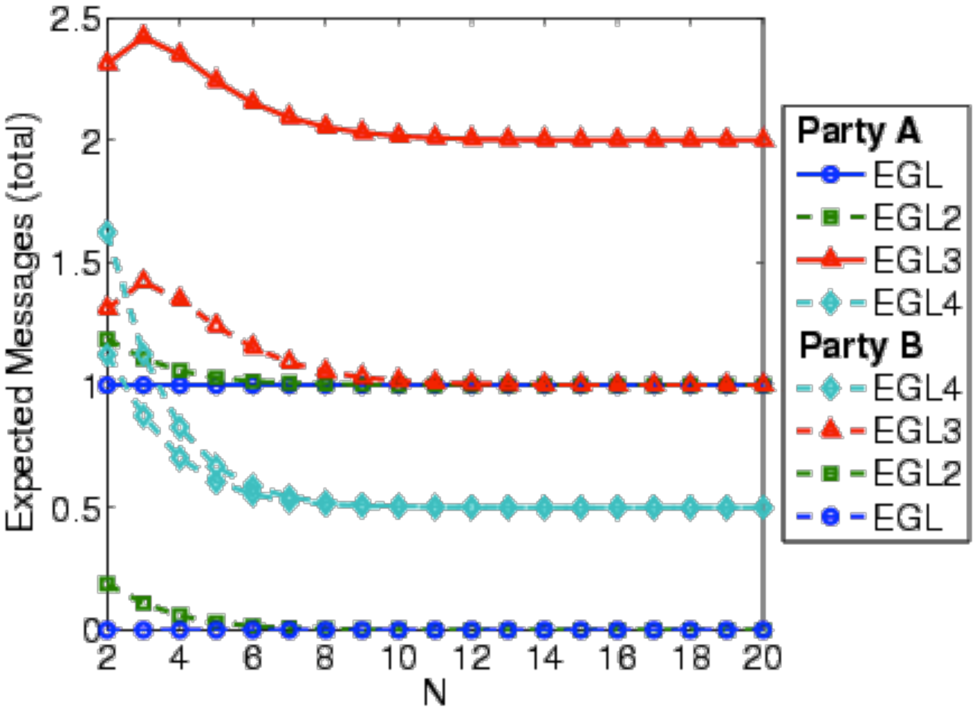
- Results show EGL4 is the ‘fairest’ protocol

- Except for “duration of fairness” measure
  - expected messages that need to be sent for a party to know a pair once the other party knows a pair
  - this value is larger for B than for A
  - and, in fact, as n increases, this measure:
    - increases for B
    - decreases for A

- Solution:
  - if a party sends a sequence of bits in a row (without the other party sending messages in between), require that the party send these bits as as a single message

Contract signing – Results

- The duration of unfairness of the protocol
  - (with the solution on the previous slide applied to all variants)



Summing up…

- Costs and rewards
  - real-valued assigned to states/transitions of a DTMC
- Properties
  - expected instantaneous/cumulative reward values
  - PRISM property specifications: adds R operator to PCTL
- Model checking
  - instantaneous: matrix-vector multiplications
  - cumulative: matrix-vector multiplications
  - reachability: graph analysis + linear equation systems
- Case study
  - randomised contract signing

Lecture 8

Continuous-time Markov chains

Dr. Dave Parker

Department of Computer Science
University of Oxford

Time in DTMCs

- Time in a DTMC proceeds in discrete steps

- Two possible interpretations:
  - accurate model of (discrete) time units
    - e.g. clock ticks in model of an embedded device
  - time-abstract
    - no information assumed about the time transitions take

- Continuous-time Markov chains (CTMCs)
  - dense model of time
  - transitions can occur at any (real-valued) time instant
  - modelled using exponential distributions

Overview

- Exponential distribution and its properties

- Continuous-time Markov chains (CTMCs)
    - definition, examples
    - race condition
    - embedded DTMC
    - generator matrix

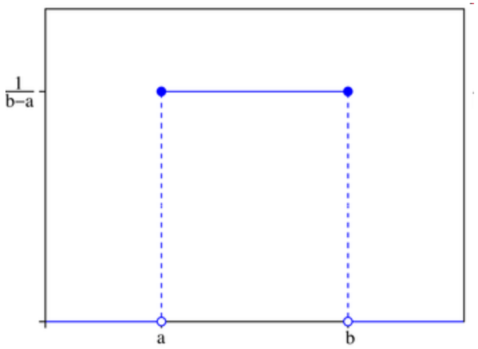
- Paths and probabilities
    - probabilistic reachability

Continuous probability distributions

- Defined by:
  - cumulative distribution function

$$F(t) = \Pr(X \le t) = \int_{-\infty}^{t} f(x)\,dx$$

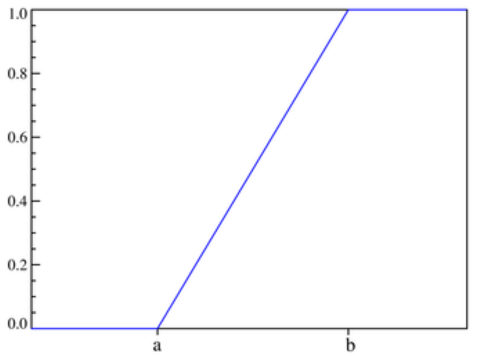
  - where f is the probability density function
  - $\Pr(X=t) = 0$ for all t



- Example: uniform distribution: U(a,b)

$$f(t) = \begin{cases} \frac{1}{b-a} & \text{if } a \le t \le b \\ 0 & \text{otherwise} \end{cases}$$

$$F(t) = \begin{cases} 0 & \text{if } t < a \\ \frac{t-a}{b-a} & \text{if } a \le t < b \\ 1 & \text{if } t \ge b \end{cases}$$



Exponential distribution

- A continuous random variable X is exponential with parameter $\lambda > 0$ if the density function is given by:

$$f(t) = \begin{cases} \lambda \cdot e^{-\lambda \cdot t} & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

λ = "rate"

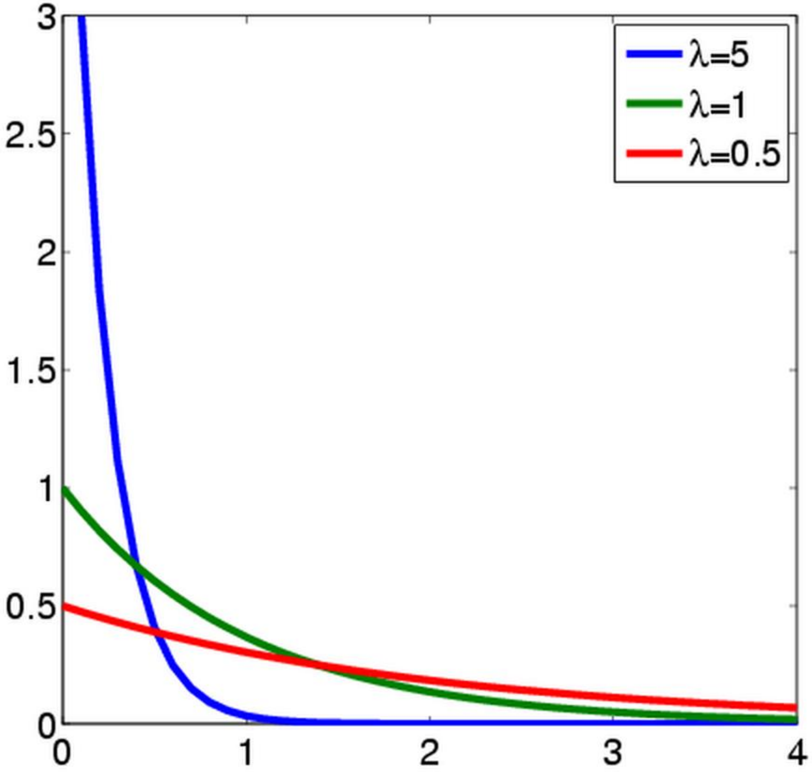
  - we write: $X \sim \text{Exponential}(\lambda)$
- Cumulative distribution function (for $t \geq 0$):

$$F(t) = \Pr(X \leq t) = \int_0^t \lambda \cdot e^{-\lambda \cdot x} dx = [-e^{-\lambda \cdot x}]_0^t = 1 - e^{-\lambda \cdot t}$$

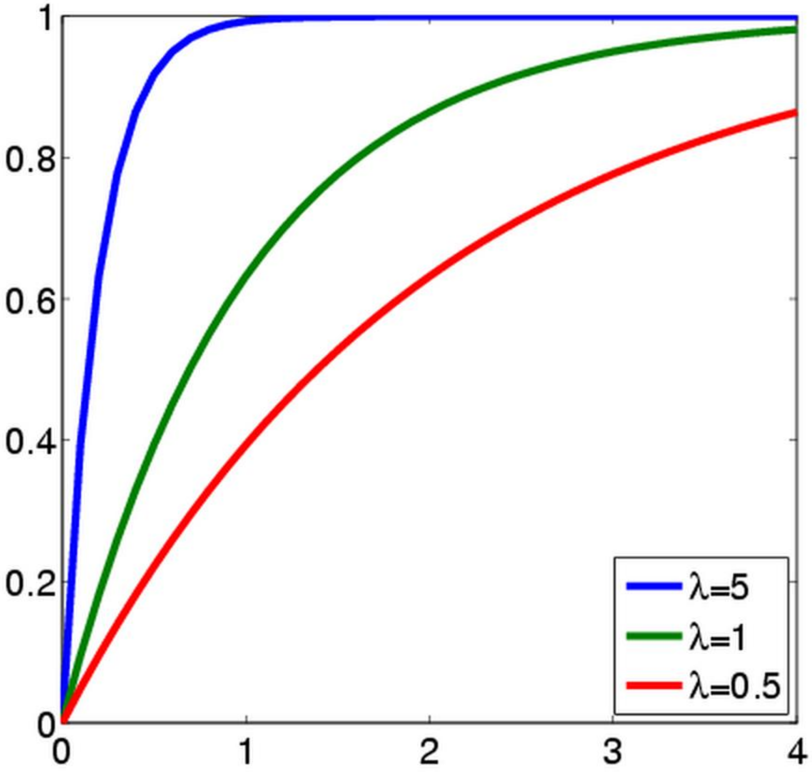
- Other properties:
  - negation: $\Pr(X > t) = e^{-\lambda \cdot t}$
  - mean (expectation): $E[X] = \int_0^\infty x \cdot \lambda \cdot e^{-\lambda \cdot x} dx = \frac{1}{\lambda}$
  - variance: $Var(X) = 1/\lambda^2$

Exponential distribution – Examples

Probability density function



Cumulative distribution function



- The more λ increases, the faster the c.d.f. approaches 1

Exponential distribution

- Adequate for modelling many real-life phenomena
  - failures
    - e.g. time before machine component fails
  - inter-arrival times
    - e.g. time before next call arrives to a call centre
  - biological systems
    - e.g. times for reactions between proteins to occur

- Maximal entropy ("uncertainty") if just the mean is known
  - i.e. best approximation when only mean is known

- Can approximate general distributions arbitrarily closely
  - phase-type distributions

Exponential distribution – Property 1

- The exponential distribution has the memoryless property:
  - $Pr(X>t_1+t_2 \mid X>t_1) = Pr(X>t_2)$

- The exponential distribution is the only continuous distribution which is memoryless
  - discrete-time equivalent is the geometric distribution

Exponential distribution – Property 2

- The minimum of two independent exponential distributions is an exponential distribution (parameter is sum)
  - $X_1 \sim \text{Exponential}(\lambda_1)$, $X_2 \sim \text{Exponential}(\lambda_2)$
  - $Y = \min(X_1,X_2)$
  - $Y \sim \text{Exponential}(\lambda_1+\lambda_2)$
- Generalises to minimum of n distributions

Exponential distribution – Property 3

- Consider two independent exponential distributions
  - $X_1 \sim \text{Exponential}(\lambda_1)$, $X_2 \sim \text{Exponential}(\lambda_2)$
  - what is the probability that $X_1 < X_2$?
  - probability that $X_1 < X_2$ is $\lambda_1/(\lambda_1+\lambda_2)$
- Generalises to n distributions

Continuous-time Markov chains

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

- Suited to modelling:
  - reliability models
  - control systems
  - queueing networks
  - biological pathways
  - chemical reactions
  - ...

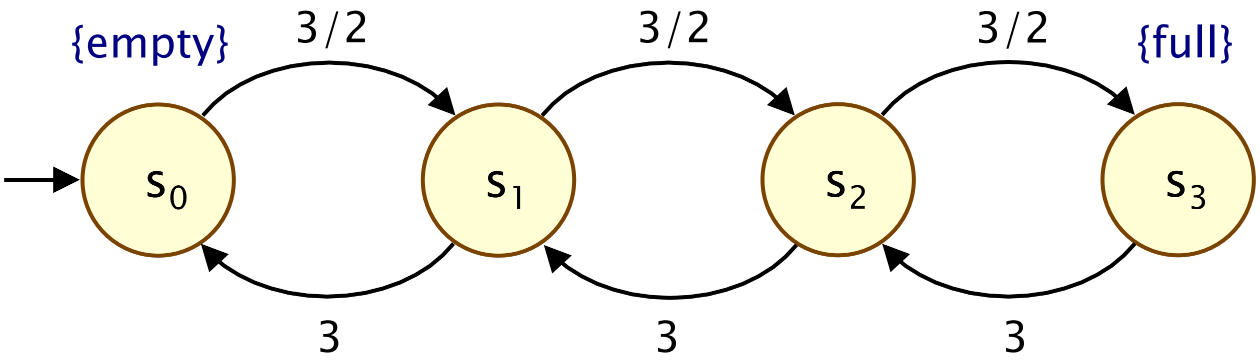
Continuous-time Markov chains

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

- Transition rate matrix assigns rates to each pair of states
    - used as a parameter to the exponential distribution
    - transition between s and s' when $\mathbf{R}(s,s')>0$
    - probability triggered before t time units: $1 - e^{-\mathbf{R}(s,s') \cdot t}$

Simple CTMC example

- Modelling a queue of jobs
  - initially the queue is empty
  - jobs arrive with rate 3/2 (i.e. mean inter-arrival time is 2/3)
  - jobs are served with rate 3 (i.e. mean service time is 1/3)
  - maximum size of the queue is 3
  - state space: $S = \{s_i\}_{i=0..3}$ where s_i indicates i jobs in queue



Race conditions

- What happens when there exists multiple s' with **R**(s,s')>0?
  - race condition: first transition triggered determines next state
  - two questions:
  - 1. How long is spent in s before a transition occurs?
  - 2. Which transition is eventually taken?

- 1. Time spent in a state before a transition
  - minimum of exponential distributions
  - exponential with parameter given by summation:

$$E(s) = \sum_{s' \in S} R(s, s')$$

  - probability of leaving a state s within [0,t] is $1 - e^{-E(s) \cdot t}$
  - E(s) is the exit rate of state s
  - s is called absorbing if E(s)=0 (no outgoing transitions)

Race conditions…

- 2. Which transition is taken from state s?
  - the choice is independent of the time at which it occurs
  - e.g. if $X_1 \sim \text{Exponential}(\lambda_1)$, $X_2 \sim \text{Exponential}(\lambda_2)$
  - then the probability that $X_1 < X_2$ is $\lambda_1/(\lambda_1+\lambda_2)$
  - more generally, the probability is given by…
- The embedded DTMC: $emb(C)=(S,s_{init},\mathbf{P}^{emb(C)},L)$
  - state space, initial state and labelling as the CTMC
  - for any $s,s' \in S$

$$\mathbf{P}^{emb(C)}(s,s') = \begin{cases} R(s,s')/E(s) & \text{if } E(s) > 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases}$$

- Probability that next state from s is s' given by $\mathbf{P}^{emb(C)}(s,s')$

Two interpretations of a CTMC

- Consider a (non-absorbing) state $s \in S$ with multiple outgoing transitions, i.e. multiple $s' \in S$ with $\mathbf{R}(s,s')>0$

- 1. Race condition
  - each transition triggered after exponentially distributed delay
    - i.e. probability triggered before t time units: $1 - e^{-\mathbf{R}(s,s')\cdot t}$
  - first transition triggered determines the next state

- 2. Separate delay/transition
  - remain in s for delay exponentially distributed with rate E(s)
    - i.e. probability of taking an outgoing transition from s within [0,t] is given by $1-e^{-E(s)\cdot t}$
  - probability that next state is s' is given by $\mathbf{P}^{emb(C)}(s,s')$
    - i.e. $\mathbf{R}(s,s')/E(s) = \mathbf{R}(s,s') \ / \ \Sigma_{s' \in S} \ \mathbf{R}(s,s')$

More on CTMCs…

- Infinitesimal generator matrix Q

$$Q(s,s') = \begin{cases} R(s,s') & s \neq s' \\ -\sum_{s \neq s'} R(s,s') & \text{otherwise} \end{cases}$$

- Alternative definition: a CTMC is:
  - a family of random variables $\{ X(t) \mid t \in \mathbb{R}_{\geq 0} \}$
  - X(t) are observations made at time instant t
  - i.e. X(t) is the state of the system at time instant t
  - which satisfies…

- Memoryless (Markov property)

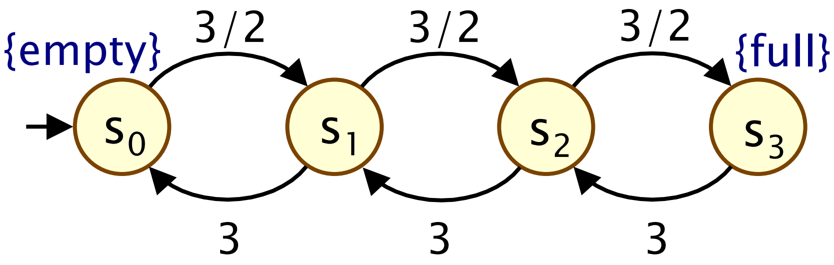
  $\Pr(X(t_k)=s_k \mid X(t_{k-1})=s_{k-1}, \ldots, X(t_0)=s_0) = \Pr(X(t_k)=s_k \mid X(t_{k-1})=s_{k-1})$

Simple CTMC example…

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

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

$s_{init} = s_0$



$AP = \{empty, full\}$

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

$$\mathbf{R} = \begin{bmatrix} 0 & 3/2 & 0 & 0 \\ 3 & 0 & 3/2 & 0 \\ 0 & 3 & 0 & 3/2 \\ 0 & 0 & 3 & 0 \end{bmatrix} \quad \mathbf{P}^{emb(C)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2/3 & 0 & 1/3 & 0 \\ 0 & 2/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} -3/2 & 3/2 & 0 & 0 \\ 3 & -9/2 & 3/2 & 0 \\ 0 & 3 & -9/2 & 3/2 \\ 0 & 0 & 3 & -3 \end{bmatrix}$$

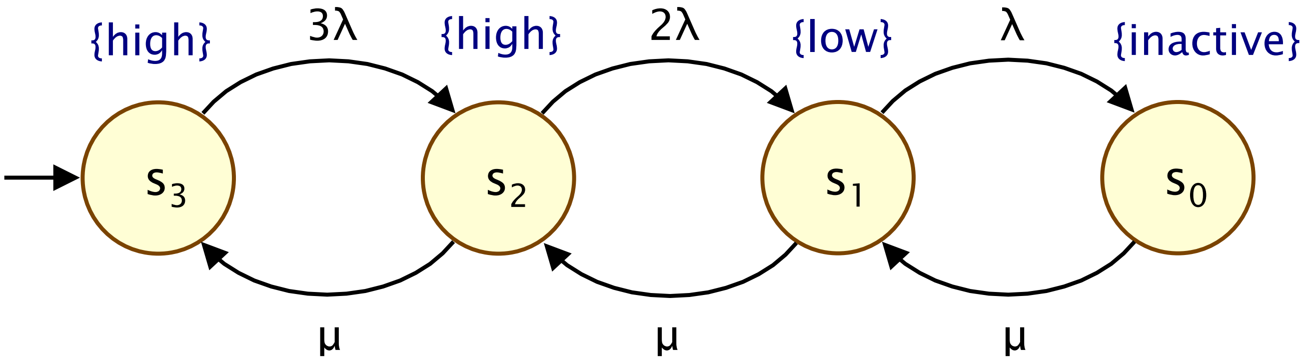
transition rate matrix

embedded DTMC

infinitesimal generator matrix

Example 2

- 3 machines, each can fail independently
  - delay modelled as exponential distributions
  - failure rate λ, i.e. mean-time to failure (MTTF) = 1 / λ
- One repair unit
  - repairs a single machine at rate μ (also exponential)
- State space:
  - $S = \{s_i\}_{i=0..3}$ where s_i indicates i machines operational

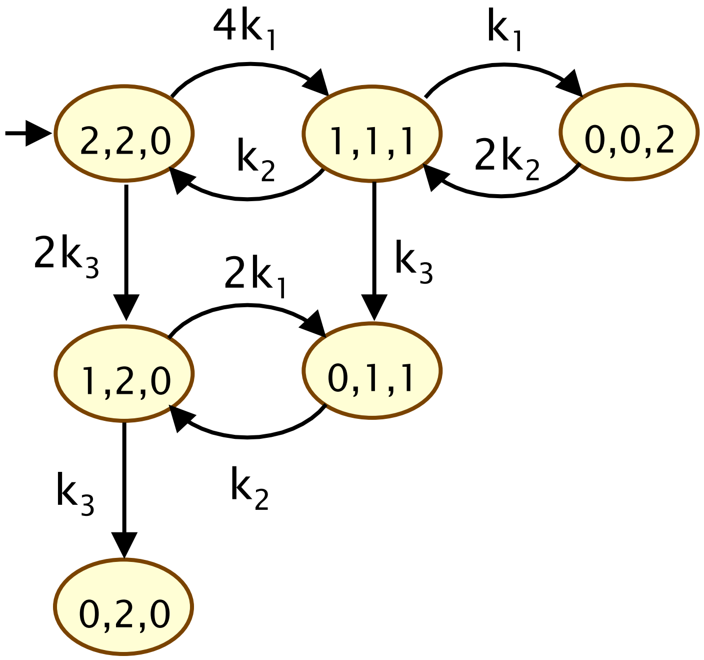


Example 3

- Chemical reaction system: two species A and B
- Two reactions:

$$A + B \underset{k_2}{\overset{k_1}{\rightleftarrows}} AB \qquad A \xrightarrow{k_3}$$

  - reversible reaction under which species A and B bind to form AB (forwards rate = $|A| \cdot |B| \cdot k_1$, backwards rate = $|AB| \cdot k_2$)
  - degradation of A (rate $|A| \cdot k_3$)
  - $|X|$ denotes number of molecules of species X
- CTMC with state space
  - $(|A|,|B|,|AB|)$
  - initially (2,2,0)



Paths of a CTMC

- An infinite path ω is a sequence $s_0t_0s_1t_1s_2t_2\ldots$ such that
  - $R(s_i,s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all $i \in \mathbb{N}$
  - t_i denotes the amount of time spent in s_i
- or a sequence $s_0t_0s_1t_1s_2t_2\ldots t_{k-1}s_k$ such that
  - $R(s_i,s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all $i<k$
  - s_k is absorbing (i.e. $R(s,s') = 0$ for all $s' \in S$)
  - i.e. remain in state s_k indefinitely

- Path(s) denotes all infinite paths starting in state s
- Further notation:
  - time(ω,j) = amount of time spent in the jth state, i.e. t_j
  - ω@t = state occupied at time t:
  - see e.g. [BHHK03, KNP07a] for precise definitions

Recall: Probability spaces

- A σ−algebra (or σ−field) on Ω is a set 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
- Elements of Σ are called measurable sets or events
- Theorem: For any set 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

- Sample space: Path(s) (set of all paths from a state s)
- Events: sets of infinite paths
- Basic events: cylinders
  - cylinders = sets of paths with common finite prefix
  - include time intervals in cylinders

- Finite prefix is a sequence $s_0,I_0,s_1,I_1,\ldots,I_{n-1},s_n$
  - $s_0,s_1,s_2,\ldots,s_n$ sequence of states where $\mathbf{R}(s_i,s_{i+1})>0$ for $i<n$
  - $I_0,I_1,I_2,\ldots,I_{n-1}$ sequence of of non-empty intervals of $\mathbb{R}_{\geq 0}$

- Cylinder $\mathrm{Cyl}(s_0,I_0,s_1,I_1,\ldots,I_{n-1},s_n)$ is the set of infinite paths:
  - $\omega(i)=s_i$ for all $i \leq n$ and $\mathrm{time}(\omega,i) \in I_i$ for all $i < n$

Probability space

- Define probability measure over cylinders inductively
- $\mathrm{Pr}_s(\mathrm{Cyl}(s))=1$
- $\mathrm{Pr}_s(\mathrm{Cyl}(s,I,s_1,I_1,\ldots,I_{n-1},s_n,I',s'))$ equals:

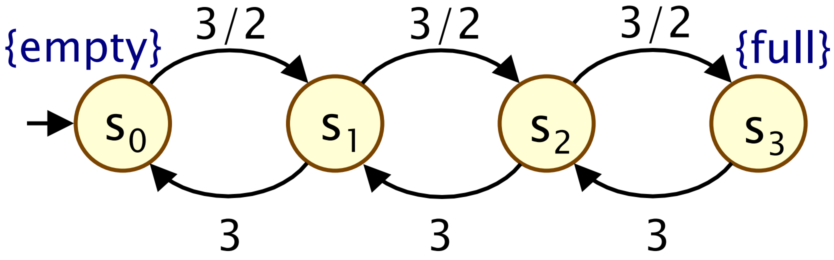
$$\underline{\mathrm{Pr}_s(\mathrm{Cyl}(s,I,s_1,I_1,\ldots,I_{n-1},s_n))} \cdot \underline{P^{emb(C)}(s_n,s')} \cdot \underline{\left(e^{-E(s_n)\cdot\inf I'} - e^{-E(s_n)\cdot\sup I'}\right)}$$

probability of transition from s_n to s’ (defined using embedded DTMC)

probability time spent in state s_n is within the interval I’

Probability space – Example

- Probability of leaving the initial state s_0 and moving to state s_1 within the first 2 time units of operation
- Cylinder $\mathrm{Cyl}(s_0,(0,2],s_1)$



- $\mathrm{Pr}_{s0}(\mathrm{Cyl}(s_0,(0,2],s_1))$

$$= \mathrm{Pr}_{s0}(\mathrm{Cyl}(s_0)) \cdot \mathbf{P}^{emb(C)}(s_0,s_1) \cdot (e^{-E(s0)\cdot 0} - e^{-E(s0)\cdot 2})$$

$$= 1 \cdot \mathbf{P}^{emb(C)}(s_0,s_1) \cdot (e^{-E(s0)\cdot 0} - e^{-E(s0)\cdot 2})$$

$$= 1 \cdot 1 \cdot (e^{-3/2\cdot 0} - e^{-3/2\cdot 2})$$

$$= 1 - e^{-3}$$

$$\approx 0.95021$$

Probability space

- Probability space (Path(s), $\Sigma_{Path(s)}$, Pr_s)   (see [BHHK03])
- Sample space Ω = Path(s)
  - i.e. all infinite paths
- Event set $\Sigma_{Path(s)}$
  - least σ-algebra on Path(s) containing all cylinders sets $Cyl(s_0,I_0,\ldots,I_{n-1},s_n)$ where:
    - $s_0,\ldots,s_n$ ranges over all state sequences with $\mathbf{R}(s_i,s_{i+1})>0$ for all i
    - $I_0,\ldots,I_{n-1}$ ranges over all sequences of non-empty intervals in $\mathbb{R}_{\geq 0}$ (where intervals are bounded by rationals)
- Probability measure Pr_s
  - Pr_s extends uniquely from probability defined over cylinders

Probabilistic reachability

- Probabilistic reachability
  - the probability of reaching a target set $T \subseteq S$
  - measurability:
    - union of all basic cylinders $Cyl(s_0,(0,\infty),s_1,(0,\infty),\ldots,(0,\infty),s_n)$ where $s_n \in T$
    - set of such state sequences $s_0 s_1 \ldots s_n$ is countable

- Time-bounded probabilistic reachability
  - the probability of reaching a target set $T \subseteq S$ within t time units
  - measurability:
    - union of all basic cylinders $Cyl(s_0,I_0,s_1,I_1,\ldots,I_{n-1},s_n)$ where $s_n \in T$ and $\sup(I_0)+\ldots+\sup(I_{n-1}) \leq t$
    - set of such state sequences $s_0 s_1 \ldots s_n$ is countable
    - set of rational-bounded intervals is countable

Summing up…

- Exponential distribution
  - suitable for modelling failures, waiting times, reactions, …
  - nice mathematical properties

- Continuous-time Markov chains
  - transition delays modelled as exponential distributions
  - race condition
  - embedded DTMC
  - generator matrix

- Probability space over paths
  - (untimed and timed) probabilistic reachability

Lecture 9
Continuous-time Markov chains…

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- Transient probabilities
  - uniformisation
- Steady-state probabilities
- CSL: Continuous Stochastic Logic
  - syntax
  - semantics
  - examples

Recall

- Continuous-time Markov chain: $C = (S, s_{init}, \mathbf{R}, L)$
  - $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is the transition rate matrix
  - rates interpreted as parameters of exponential distributions

- Embedded DTMC: $emb(C) = (S, s_{init}, \mathbf{P}^{emb(C)}, L)$

$$\mathbf{P}^{emb(C)}(s,s') = \begin{cases} \mathbf{R}(s,s')/E(s) & \text{if } E(s) > 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases}$$

- Infinitesimal generator matrix

$$\mathbf{Q}(s,s') = \begin{cases} \mathbf{R}(s,s') & s \neq s' \\ -\sum_{s \neq s'} \mathbf{R}(s,s') & \text{otherwise} \end{cases}$$

Transient and steady-state behaviour

- Transient behaviour
  - state of the model at a particular time instant
  - $\underline{\pi}^C_{s,t}(s')$ is probability of, having started in state s, being in state s' at time t (in CTMC C)
  - $\underline{\pi}^C_{s,t}(s') = Pr_s\{ \omega \in Path^C(s) \mid \omega@t=s' \}$

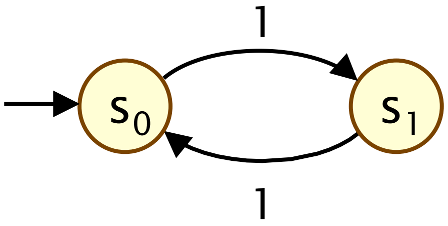
- Steady-state behaviour
  - state of the model in the long-run
  - $\underline{\pi}^C_s(s')$ is probability of, having started in state s, being in state s' in the long run
  - $\underline{\pi}^C_s(s') = \lim_{t\to\infty} \underline{\pi}^C_{s,t}(s')$
  - intuitively: long-run percentage of time spent in each state

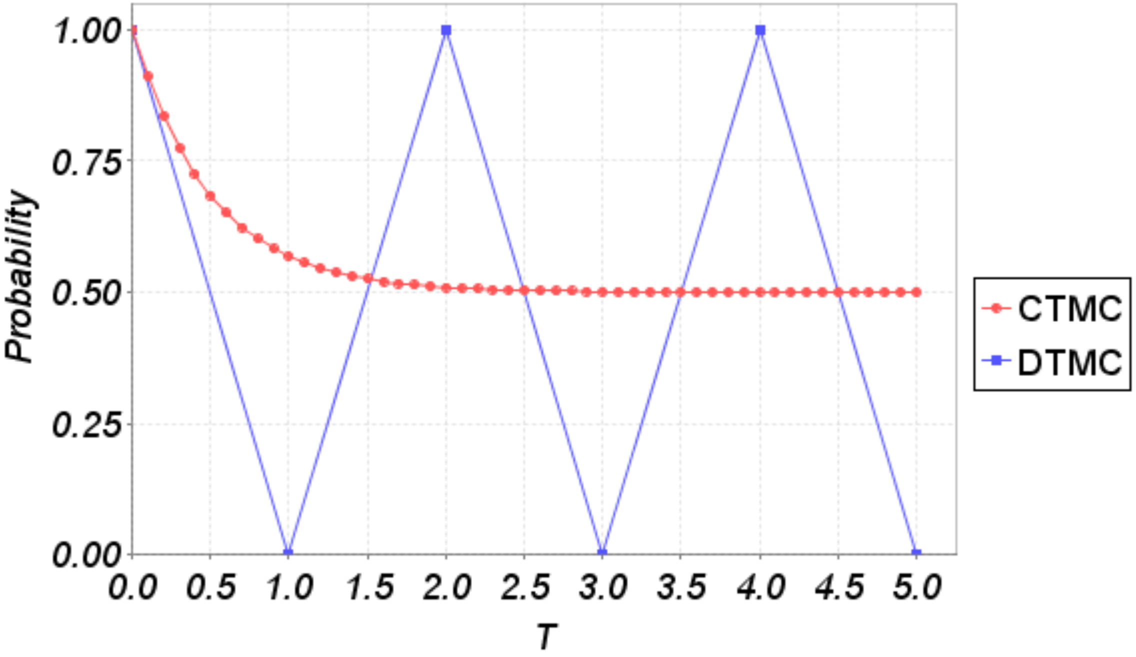
Computing transient probabilities

- Consider a simple example
  - and compare to the case for DTMCs

- What is the probability of being in state s_0 at time t?

- DTMC/CTMC:





Computing transient probabilities

- Π_t – matrix of transient probabilities
  - $\Pi_t(s,s')=\underline{\pi}_{s,t}(s')$

- Π_t solution of the differential equation: $\Pi_t' = \Pi_t \cdot Q$
  - where Q is the infinitesimal generator matrix

- Can be expressed as a matrix exponential and therefore evaluated as a power series

$$\Pi_t = e^{Q \cdot t} = \sum\nolimits_{i=0}^{\infty} (Q \cdot t)^i / i!$$

  - computation potentially unstable
  - probabilities instead computed using uniformisation

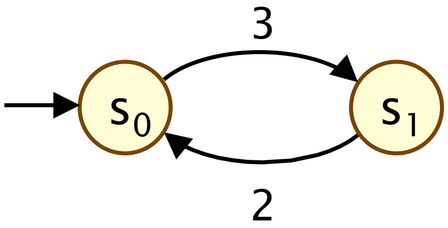
Uniformisation

- We build the uniformised DTMC unif(C) of CTMC C
- If $C = (S, s_{init}, \mathbf{R}, L)$, then $unif(C) = (S, s_{init}, \mathbf{P}^{unif(C)}, L)$
  - set of states, initial state and labelling the same as C
  - $\mathbf{P}^{unif(C)} = \mathbf{I} + \mathbf{Q}/q$
  - $\mathbf{I}$ is the $|S| \times |S|$ identity matrix
  - $q \geq \max \{ E(s) \mid s \in S \}$ is the uniformisation rate

- Each time step (epoch) of uniformised DTMC corresponds to one exponentially distributed delay with rate q
  - if $E(s)=q$ transitions the same as embedded DTMC (residence time has the same distribution as one epoch)
  - if $E(s)<q$ add self loop with probability $1-E(s)/q$ (residence time longer than $1/q$ so one epoch may not be 'long enough')

Uniformisation – Example

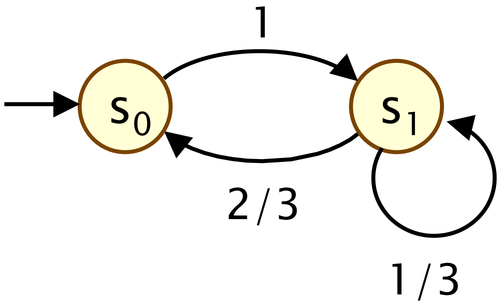
- CTMC C:



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

- Uniformised DTMC unif(C)
  - let uniformisation rate q = $\max_s$ { E(s) } = 3

$$\mathbf{P}^{\text{unif}(C)} = \mathbf{I} + \mathbf{Q}/q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ \frac{2}{3} & -\frac{2}{3} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$



Uniformisation

- Using the uniformised DTMC the transient probabilities can be expressed by:

$$\Pi_t = e^{Q\cdot t} = e^{q\cdot(\mathbf{P}^{unif(C)} - \mathbf{I})\cdot t} = e^{(q\cdot t)\cdot\mathbf{P}^{unif(C)}} \cdot e^{-q\cdot t}$$

$$= e^{-q\cdot t} \cdot \left(\sum_{i=0}^{\infty} \frac{(q\cdot t)^i}{i!} \cdot \left(\mathbf{P}^{unif(C)} \right)^i \right)$$

$$= \sum_{i=0}^{\infty} \left(e^{-q\cdot t} \cdot \frac{(q\cdot t)^i}{i!} \right) \cdot \left(\mathbf{P}^{unif(C)} \right)^i$$

$$= \sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \left(\mathbf{P}^{unif(C)} \right)^i$$

ith Poisson probability with parameter q·t

$\mathbf{P}^{unif(C)}$ is stochastic (all entries in [0,1] & rows sum to 1); therefore computations with **P** are more numerically stable than **Q**

Uniformisation

$$\Pi_t = \sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \left(\mathbf{P}^{unif(C)} \right)^i$$

- $(\mathbf{P}^{unif(C)})^i$ is probability of jumping between each pair of states in i steps

- $\gamma_{q\cdot t,i}$ is the ith Poisson probability with parameter q·t
  - the probability of i steps occurring in time t, given each has delay exponentially distributed with rate q

- Can truncate the (infinite) summation using the techniques of Fox and Glynn [FG88], which allow efficient computation of the Poisson probabilities

Uniformisation

- Computing $\underline{\pi}_{s,t}$ for a fixed state s and time t
  - can be computed efficiently using matrix-vector operations
  - pre-multiply the matrix Π_t by the initial distribution
  - in this case: $\underline{\pi}_{s,0}(s')$ equals 1 if s=s' and 0 otherwise

$$\underline{\pi}_{s,t} = \underline{\pi}_{s,0} \cdot \Pi_t = \underline{\pi}_{s,0} \cdot \sum_{i=0}^{\infty} \gamma_{q \cdot t,i} \cdot \left(\mathbf{P}^{unif(C)} \right)^i$$

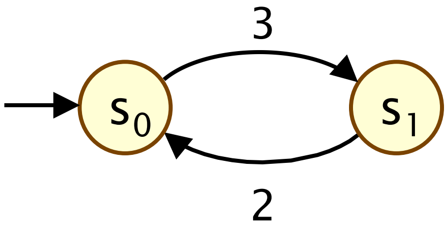
$$= \sum_{i=0}^{\infty} \gamma_{q \cdot t,i} \cdot \underline{\pi}_{s,0} \cdot \left(\mathbf{P}^{unif(C)} \right)^i$$

  - compute iteratively to avoid the computation of matrix powers

$$\left(\underline{\pi}_{s,t} \cdot \mathbf{P}^{unif(C)} \right)^{i+1} = \left(\underline{\pi}_{s,t} \cdot \mathbf{P}^{unif(C)} \right)^{i} \cdot \mathbf{P}^{unif(C)}$$

Uniformisation – Example

- CTMC C, uniformised DTMC for q=3



$$\mathbf{R} = \begin{bmatrix} 0 & 3 \\ 2 & 0 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} -3 & 3 \\ 2 & -2 \end{bmatrix} \quad \mathbf{P}^{\text{unif}(C)} = \begin{bmatrix} 0 & 1 \\ 2/3 & 1/3 \end{bmatrix}$$

- Initial distribution: $\underline{\pi}_{s0,0} = [\ 1, 0\ ]$
- Transient probabilities for time t = 1:

$$\underline{\pi}_{s0,1} = \sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \underline{\pi}_{s0,0} \cdot \left(\mathbf{P}^{\text{unif}(C)}\right)^{i}$$

$$= \gamma_{3,0} \cdot [1,0] \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \gamma_{3,1} \cdot [1,0] \cdot \begin{bmatrix} 0 & 1 \\ 2/3 & 1/3 \end{bmatrix} + \gamma_{3,2} \cdot [1,0] \cdot \begin{bmatrix} 0 & 1 \\ 2/3 & 1/3 \end{bmatrix}^{2} + \ldots$$

$$\approx [\ 0.404043,\ 0.595957\ ]$$

Steady-state probabilities

- Limit $\underline{\pi}^C_s(s') = \lim_{t \to \infty} \underline{\pi}^C_{s,t}(s')$
  - exists for all finite CTMCs
  - (see next slide)

- As for DTMCs, need to consider the underlying graph structure of the Markov chain:
  - reachability (between pairs) of states
  - bottom strongly connected components (BSCCs)
  - one special case to consider: absorbing states are BSCCs
  - note: can do this equivalently on embedded DTMC

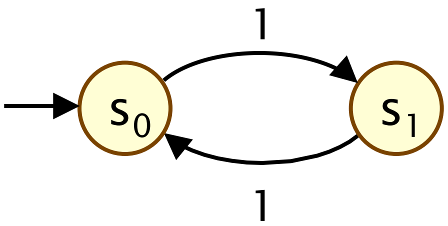
- CTMC is irreducible if all its states belong to a single BSCC; otherwise reducible

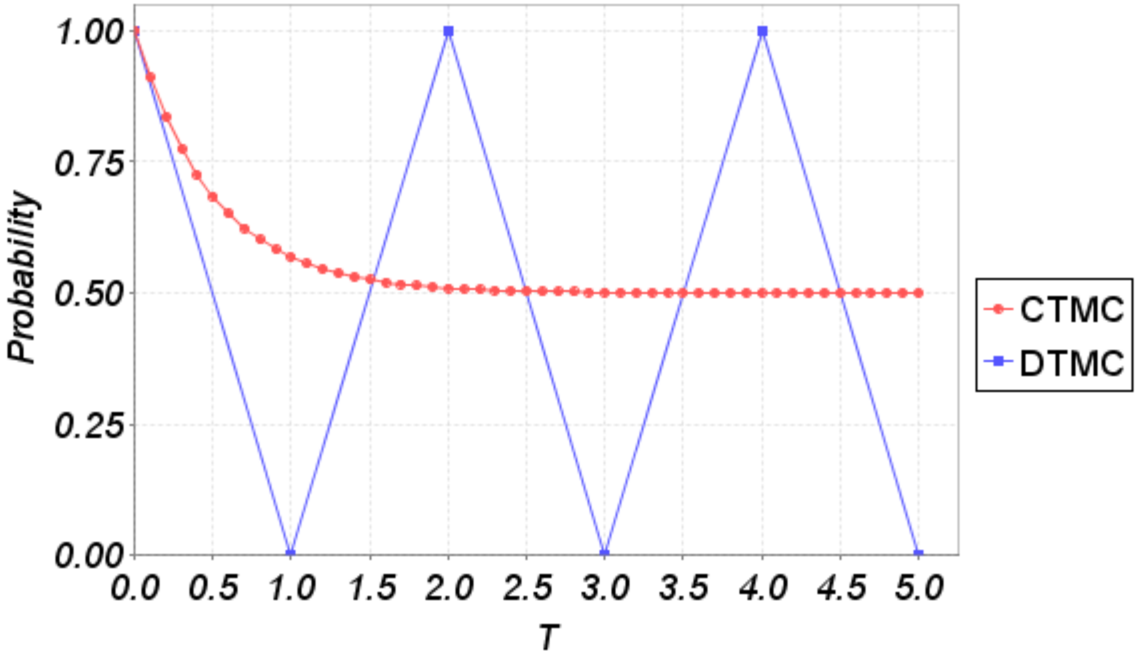
Periodicity

- Unlike for DTMCs, do not need to consider periodicity

- e.g. probability of being in state s_0 at time t?

- DTMC/CTMC:





Irreducible CTMCs

- For an irreducible CTMC:
  - the steady-state probabilities are independent of the starting state: denote the steady state probabilities by $\underline{\pi}^C(s')$

- These probabilities can be computed as
  - the unique solution of the linear equation system:

$$\underline{\pi}^C \cdot \mathbf{Q} = \underline{0} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}^C(s) = 1$$

where **Q** is the infinitesimal generator matrix of C

- Solved by standard means:
  - direct methods, such as Gaussian elimination
  - iterative methods, such as Jacobi and Gauss-Seidel

Balance equations

$$\underline{\pi}^C \cdot \mathbf{Q} = \underline{0} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}^C(s) = 1$$

balance the rate of leaving and entering a state

normalisation

For all s ∈ S:

$$\underline{\pi}^C(s) \cdot (-\Sigma_{s' \neq s} \mathbf{R}(s,s')) + \Sigma_{s' \neq s} \underline{\pi}^C(s') \cdot \mathbf{R}(s',s) = 0$$

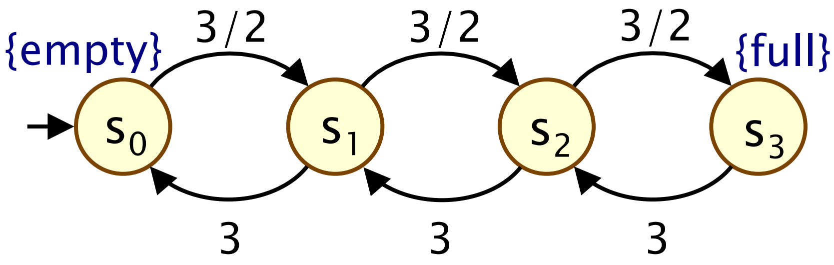
$$\Leftrightarrow$$

$$\underline{\pi}^C(s) \cdot \Sigma_{s' \neq s} \mathbf{R}(s,s') = \Sigma_{s' \neq s} \underline{\pi}^C(s') \cdot \mathbf{R}(s',s)$$

Steady-state – Example

- Solve: $\underline{\pi} \cdot \mathbf{Q} = 0$ and $\Sigma\ \underline{\pi}(s) = 1$

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



$$\begin{array}{rcrcrcrcl} -3/2 \cdot \underline{\pi}(s_0) & + & 3 \cdot \underline{\pi}(s_1) & & & & & = & 0 \\ 3/2 \cdot \underline{\pi}(s_0) & - & 9/2 \cdot \underline{\pi}(s_1) & + & 3 \cdot \underline{\pi}(s_2) & & & = & 0 \\ & & 3/2 \cdot \underline{\pi}(s_1) & - & 9/2 \cdot \underline{\pi}(s_2) & + & 3 \cdot \underline{\pi}(s_3) & = & 0 \\ & & & & 3/2 \cdot \underline{\pi}(s_2) & - & 3 \cdot \underline{\pi}(s_3) & = & 0 \\ \underline{\pi}(s_0) & + & \underline{\pi}(s_1) & + & \underline{\pi}(s_2) & + & \underline{\pi}(s_3) & = & 1 \end{array}$$

$\underline{\pi} = [\ 8/15, 4/15, 2/15, 1/15\ ]$

Reducible CTMCs

- For a reducible CTMC:
  - the steady-state probabilities $\underline{\pi}^C(s')$ depend on start state s
- Find all BSCCs of CTMC, denoted bscc(C)
- Compute:
  - steady-state probabilities $\underline{\pi}^T$ of sub-CTMC for each BSCC T
  - probability $\text{ProbReach}^{emb(C)}(s, T)$ of reaching each T from s
- Then:

$$\underline{\pi}^C_s(s') = \begin{cases} \text{ProbReach}^{emb(C)}(s, T) \cdot \underline{\pi}^T(s') & \text{if } s' \in T \text{ for some } T \in \text{bscc}(C) \\ 0 & \text{otherwise} \end{cases}$$

CSL

- Temporal logic for describing properties of CTMCs
  - CSL = Continuous Stochastic Logic [ASSB00,BHHK03]
  - extension of (non-probabilistic) temporal logic CTL
- Key additions:
  - probabilistic operator P (like PCTL)
  - steady state operator S
- Example: $\mathrm{down} \rightarrow \mathrm{P}_{>0.75}\ [\ \neg\mathrm{fail}\ \mathrm{U}^{[1,2.5]}\ \mathrm{up}\ ]$
  - when a shutdown occurs, the probability of a system recovery being completed between 1 and 2.5 hours without further failure is greater than 0.75
- Example: $\mathrm{S}_{<0.1}[\ \mathrm{insufficient\_routers}\ ]$
  - in the long run, the chance that an inadequate number of routers are operational is less than 0.1

CSL syntax

- CSL syntax:

ψ is true with probability ~p

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

"next"

"time bounded until"

in the "long run" ϕ is true with probability ~p

  - where a is an atomic proposition, I interval of $\mathbb{R}_{\geq 0}$ and $p \in [0,1]$, $\sim \in \{<,>,\leq,\geq\}$
- A CSL formula is always a state formula
  - path formulae only occur inside the P operator

CSL semantics for CTMCs

- CSL formulae interpreted over states of a CTMC
  - $s \vDash \phi$ denotes ϕ is "true in state s" or "satisfied in state s"
- Semantics of state formulae:
  - for a state s of the CTMC $(S, s_{init}, \mathbf{R}, L)$:

  - $s \vDash a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \vDash \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \vDash \phi_1$ and $s \vDash \phi_2$
  - $s \vDash \neg\phi \quad \Leftrightarrow \quad s \vDash \phi$ is false
  - $s \vDash P_{\sim p}\,[\psi] \quad \Leftrightarrow \quad Prob(s, \psi) \sim p$
  - $s \vDash S_{\sim p}\,[\phi] \quad \Leftrightarrow \quad \Sigma_{s' \vDash \phi}\, \underline{\pi}_s(s') \sim p$

Probability of, starting in state s, satisfying the path formula ψ

Probability of, starting in state s, being in state s' in the long run

CSL semantics for CTMCs

- Prob(s, ψ) is the probability, starting in state s, of satisfying the path formula ψ
  - $\text{Prob}(s, \psi) = \Pr_s \{\omega \in \text{Path}_s \mid \omega \vDash \psi \}$

if $\omega(0)$ is absorbing
$\omega(1)$ not defined

- Semantics of path formulae:
  - for a path ω of the CTMC:
  - $\omega \vDash X\ \phi \quad \Leftrightarrow \quad \omega(1)$ is defined and $\omega(1) \vDash \phi$
  - $\omega \vDash \phi_1\ U^I\ \phi_2 \quad \Leftrightarrow \quad \exists t \in I.\ (\ \omega@t \vDash \phi_2 \wedge \forall t'<t.\ \omega@t' \vDash \phi_1)$

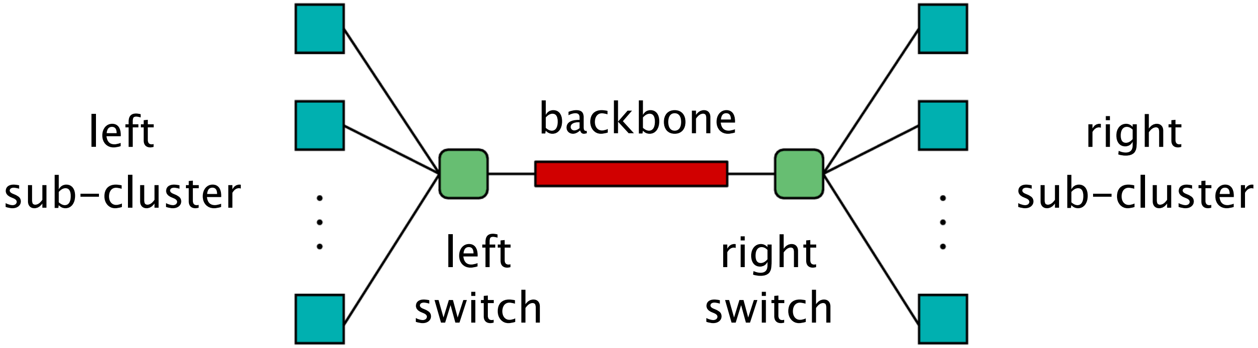
there exists a time instant in the interval I where ϕ_2 is true and ϕ_1 is true at all preceding time instants

More on CSL

- Basic logical derivations:
  - false, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$
- Normal (unbounded) until is a special case
  - $\phi_1 \ U \ \phi_2 \equiv \phi_1 \ U^{[0,\infty)} \ \phi_2$
- Derived path formulae:
  - $F \ \phi \equiv \text{true} \ U \ \phi$, $F^I \ \phi \equiv \text{true} \ U^I \ \phi$
  - $G \ \phi \equiv \neg(F \ \neg\phi)$, $G^I \ \phi \equiv \neg(F^I \ \neg\phi)$
- Negate probabilities: …
  - e.g. $\neg P_{>p} [\ \psi \ ] \equiv P_{\leq p} [\ \psi \ ]$, $\neg S_{\geq p} [\ \phi \ ] \equiv S_{>p} [\ \phi \ ]$
- Quantitative properties
  - of the form $P_{=?} [\ \psi \ ]$ and $S_{=?} [\ \phi \ ]$
  - where P/S is the outermost operator
  - experiments, patterns, trends, …

CSL example – Workstation cluster

- Case study: Cluster of workstations [HHK00]
  - two sub-clusters (N workstations in each cluster)
  - star topology with a central switch
  - components can break down, single repair unit



  - minimum QoS: at least ¾ of the workstations operational and connected via switches
  - premium QoS: all workstations operational and connected via switches

CSL example – Workstation cluster

- $S_{=?}$ [minimum]
  - the probability in the long run of having minimum QoS
- $P_{=?}$ [$F^{[t,t]}$ minimum]
  - the (transient) probability at time instant t of minimum QoS
- $P_{<0.05}$ [$F^{[0,10]}$ $\neg$minimum]
  - the probability that the QoS drops below minimum within 10 hours is less than 0.05
- $\neg$minimum → $P_{<0.1}$ [$F^{[0,2]}$ $\neg$minimum]
  - when facing insufficient QoS, the chance of facing the same problem after 2 hours is less than 0.1

CSL example – Workstation cluster

- $\text{minimum} \rightarrow P_{>0.8}\,[\, \text{minimum}\ U^{[0,t]}\ \text{premium}\ ]$
  - the probability of going from minimum to premium QoS within t hours without violating minimum QoS is at least 0.8
- $P_{=?}\,[\ \neg\text{minimum}\ U^{[t,\infty)}\ \text{minimum}\ ]$
  - the chance it takes more than t time units to recover from insufficient QoS
- $\neg\text{r\_switch\_up} \rightarrow P_{<0.1}\,[\neg\text{r\_switch\_up}\ U\ \neg\text{l\_switch\_up}\ ]$
  - if the right switch has failed, the probability of the left switch failing before it is repaired is less than 0.1
- $P_{=?}\,[\ F^{[2,\infty)}\ S_{>0.9}[\ \text{minimum}\ ]\ ]$
  - the probability of it taking more than 2 hours to get to a state from which the long-run probability of minimum QoS is >0.9

Summing up…

- Transient probabilities (time instant t)
  - computation with uniformisation: efficient iterative method
- Steady-state (long-run) probabilities
  - like DTMCs
  - requires graph analysis
  - irreducible case: solve linear equation system
  - reducible case: steady-state for sub-CTMCs + reachability
- CSL: Continuous Stochastic Logic
  - extension of PCTL for properties of CTMCs

Lecture 10
Model Checking for CTMCs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- CSL model checking
  - basic algorithm
  - untimed properties
  - time-bounded until
  - the S (steady-state) operator

- Rewards
  - reward structures for CTMCs
  - properties: extension of CSL
  - model checking

CSL: Continuous Stochastic Logic

- CSL syntax:
  - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p}\,[\psi] \mid S_{\sim p}\,[\phi]$ (state formulae)
  - $\psi ::= X\,\phi \mid \phi\, U^{I}\, \phi$ (path formulae)

ψ is true with probability ~p

in the “long run” ϕ is true with probability ~p

“next”

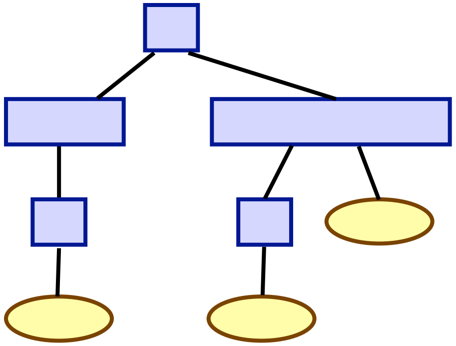
“time bounded until”

  - where a is an atomic proposition, I an interval of $\mathbb{R}_{\geq 0}$, $p \in [0,1]$ and $\sim \in \{<,>,\leq,\geq\}$

CSL model checking for CTMCs

- Algorithm for CSL model checking [BHHK03]
  - inputs: CTMC $C=(S,s_{init},\mathbf{R},L)$, CSL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$, the set of states satisfying ϕ
- Often, also consider quantitative results
  - e.g. compute result of $P_{=?}$ [$F^{[0,t]}$ minimum] for $0 \leq t \leq 100$

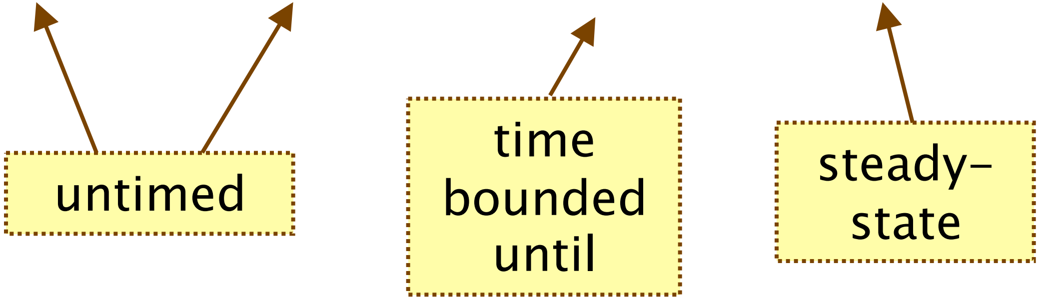
- Basic algorithm similar to PCTL for DTMCs
  - proceeds by induction on parse tree of ϕ
- For the non-probabilistic operators:
  - $Sat(true) = S$
  - $Sat(a) = \{ s \in S \mid a \in L(s) \}$
  - $Sat(\neg\phi) = S \setminus Sat(\phi)$
  - $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2)$



CSL model checking for CTMCs

- Main task: computing probabilities for $P_{\sim p}\,[\cdot]$ and $S_{\sim p}\,[\cdot]$
  - $\phi ::= \mathrm{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid$

    $P_{\sim p}\,[\,X\,\phi\,] \mid P_{\sim p}\,[\,\phi\ U\ \phi\,] \mid P_{\sim p}\,[\,\phi\ U^{I}\ \phi\,] \mid S_{\sim p}\,[\,\phi\,]$



untimed

time bounded until

steady-state

  - where $\phi_1\ U\ \phi_2 \equiv \phi_1\ U^{[0,\infty)}\ \phi_2$

Untimed properties

- Untimed properties can be verified on the embedded DTMC
  - properties of the form: $P_{\sim p}$ [X ϕ] or $P_{\sim p}$ [ϕ_1 U ϕ_2]
  - use algorithms for checking PCTL against DTMCs

- Certain qualitative time-bounded until formulae can also be verified on the embedded DTMC
  - for any (non-empty) interval I

$$s \vDash P_{\sim 0} [\ \phi_1\ U^{I}\ \phi_2\ ] \text{ if and only if } s \vDash P_{\sim 0} [\phi_1\ U^{[0,\infty)}\ \phi_2\ ]$$

  - can use precomputation algorithm Prob0

Model checking – Time-bounded until

- Compute Prob(s, $\phi_1\ U^I\ \phi_2$) for all states where I is an arbitrary interval of the non-negative real numbers
- Note:
  - Prob(s, $\phi_1\ U^I\ \phi_2$) = Prob(s, $\phi_1\ U^{cl(I)}\ \phi_2$)

    where cl(I) denotes the closure of the interval I
  - Prob(s, $\phi_1\ U^{[0,\infty)}\ \phi_2$) = $Prob^{emb(C)}$(s, $\phi_1\ U\ \phi_2$)

    where emb(C) is the embedded DTMC

- Therefore, 3 remaining cases to consider:
  - $I = [0,t]$ for some $t \in \mathbb{R}_{\geq 0}$, $I = [t,t']$ for some $t \leq t' \in \mathbb{R}_{\geq 0}$ and $I = [t,\infty)$ for some $t \in \mathbb{R}_{\geq 0}$

- Two methods: 1. Integral equations; 2. Uniformisation

Time-bounded until (integral equations)

- Computing the probabilities reduces to determining the least solution of the following set of integral equations
  - (note similarity to bounded until for DTMCs)
- Prob(s, $\phi_1 \ U^{[0,t]} \ \phi_2$) equals
  - 1 if $s \in Sat(\phi_2)$,
  - 0 if $s \in Sat(\neg\phi_1 \wedge \neg\phi_2)$
  - and otherwise equals

probability of moving from s to s' at time x

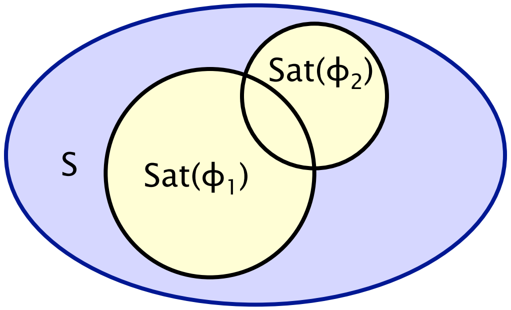
probability, in state s', of satisfying until before t-x time units elapse

$$\int_0^t \sum_{s' \in S} \left(P^{emb(C)}(s,s') \cdot E(s) \cdot e^{-E(s)\cdot x} \right) \cdot Prob(s', \phi_1 \ U^{[0,t-x]} \ \phi_2) \ dx$$

- One possibility: solve these integrals numerically
  - e.g. trapezoidal, Simpson and Romberg integration
  - expensive, possible problems with numerical stability

Time-bounded until (uniformisation)

- Reduction to transient analysis…



- Make all ϕ_2 states absorbing
  - from such a state $\phi_1 \ U^{[0,x]} \ \phi_2$ holds with probability 1

- Make all $\neg\phi_1 \wedge \neg\phi_2$ states absorbing
  - from such a state $\phi_1 \ U^{[0,x]} \ \phi_2$ holds with probability 0

- Formally: Construct CTMC $C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]$
  - where for CTMC $C=(S,s_{init},\mathbf{R},L)$, let $C[\theta]=(S,s_{init},\mathbf{R}[\theta],L)$ where $\mathbf{R}[\theta](s,s')=\mathbf{R}(s,s')$ if $s \notin Sat(\theta)$ and 0 otherwise

Time-bounded until (uniformisation)

- Problem then reduces to calculating transient probabilities of the CTMC $C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]$:

$$\mathrm{Prob}(s, \phi_1\ U^{[0,t]}\ \phi_2) = \sum_{s' \in Sat(\phi_2)} \underline{\pi}_{s,t}^{C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]}(s')$$

transient probability: starting in state s, the probability of being in state s’ at time t

Time-bounded until (uniformisation)

- Can now adapt uniformisation to computing the vector of probabilities $\underline{\text{Prob}}(\phi_1\ U^{[0,t]}\ \phi_2)$
  - recall Π_t is matrix of transient probabilities $\Pi_t(s,s')=\underline{\pi}_{s,t}(s')$
  - computed via uniformisation: $\Pi_t = \sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \left(P^{\text{unif}(C)}\right)^i$
- Combining with: $\text{Prob}(s, \phi_1\ U^{[0,t]}\ \phi_2) = \sum_{s' \in \text{Sat}(\phi_2)} \underline{\pi}_{s,t}^{C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]}(s')$

$$\begin{aligned}\underline{\text{Prob}}(\phi_1\ U^{[0,t]}\ \phi_2) &= \Pi_t^{C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]} \cdot \underline{\phi_2} \\ &= \left(\sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \left(P^{\text{unif}(C[\phi_2][\neg\phi_1 \wedge \neg\phi_2])}\right)^i\right) \underline{\phi_2} \\ &= \sum_{i=0}^{\infty} \left(\gamma_{q\cdot t,i} \cdot \left(P^{\text{unif}(C[\phi_2][\neg\phi_1 \wedge \neg\phi_2])}\right)^i \cdot \underline{\phi_2}\right)\end{aligned}$$

Time–bounded until (uniformisation)

- Have shown that we can calculate the probabilities as:

$$\underline{\text{Prob}}(\phi_1 \ \text{U}^{[0,t]} \ \phi_2) = \sum_{i=0}^{\infty} \left(\gamma_{q \cdot t, i} \cdot \left(\mathbf{P}^{\text{unif}(C[\phi_2][\neg\phi_1 \wedge \neg\phi_2])} \right)^{i} \cdot \underline{\phi_2} \right)$$

- Infinite summation can be truncated using the techniques of Fox and Glynn [FG88]

- Can compute iteratively to avoid matrix powers:

$$\left(\mathbf{P}^{\text{unif}(C)}\right)^{0} \cdot \underline{\phi_2} = \underline{\phi_2}$$

$$\left(\mathbf{P}^{\text{unif}(C)} \right)^{i+1} \cdot \underline{\phi_2} = \mathbf{P}^{\text{unif}(C)} \cdot \left(\left(\mathbf{P}^{\text{unif}(C)} \right)^{i} \cdot \underline{\phi_2} \right)$$

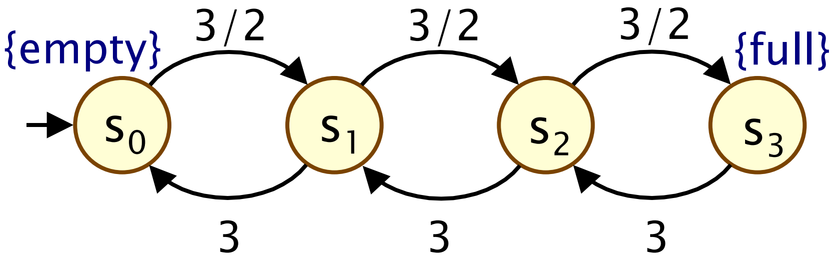
Time-bounded until – Example

- $P_{>0.65}$ [$F^{[0,7.5]}$ full] $\equiv$ $P_{>0.65}$ [true $U^{[0,7.5]}$ full]
  - "probability of the queue becoming full within 7.5 time units"
- State s_3 satisfies full and no states satisfy ¬true
  - in C[full][¬true ∧¬ full] only state s_3 made absorbing

$$\begin{bmatrix} 2/3 & 1/3 & 0 & 0 \\ 2/3 & 0 & 1/3 & 0 \\ 0 & 2/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

matrix of unif(C[full][¬true ∧¬full]) with uniformisation rate $\max_{s \in S} E(s)$ =4.5

s_3 made absorbing



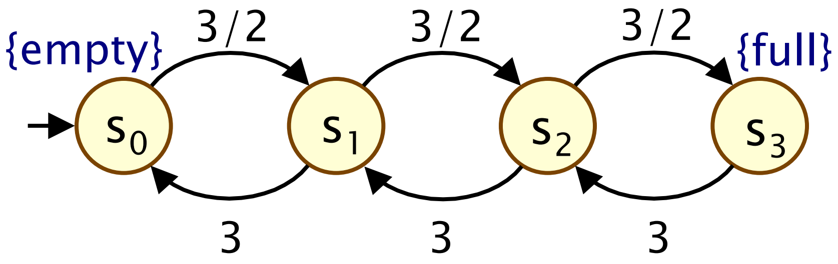
Time-bounded until – Example

- Computing the summation of matrix-vector multiplications

$$\underline{\mathrm{Prob}}(\phi_1\ \mathrm{U}^{[0,t]}\ \phi_2) = \sum_{i=0}^{\infty} \left(\gamma_{q\cdot t,i} \cdot \left(\mathrm{P}^{unif(C[\phi_2][\neg\phi_1 \wedge \neg\phi_2])}\right)^{i} \cdot \underline{\phi_2} \right)$$

  - yields $\underline{\mathrm{Prob}}(\ \mathrm{F}^{[0,7.5]}\ \mathrm{full}\ ) \approx [\ 0.6482,\ 0.6823,\ 0.7811,\ 1\ ]$

- $\mathrm{P}_{>0.65}[\ \mathrm{F}^{[0,7.5]}\ \mathrm{full}\ ]$ satisfied in states s_1, s_2 and s_3



Time-bounded until – $P_{\sim p}[\phi_1 \ U^{[t,t']} \ \phi_2]$

- In this case the computation can be split into two parts:
- 1. Probability of remaining in ϕ_1 states until time t
  - can be computed as transient probabilities on the CTMC where are states satisfying $\neg\phi_1$ have been made absorbing
- 2. Probability of reaching a ϕ_2 state, while remaining in states satisfying ϕ_1, within the time interval [0,t’–t]
  - i.e. computing Prob($\phi_1 \ U^{[0,t'-t]} \ \phi_2$)

$$\mathrm{Prob}(s, \phi_1 \ U^{[t,t']} \ \phi_2) = \sum_{s' \in Sat(\phi_1)} \underline{\pi}_{s,t}^{C[\neg\phi_1]}(s') \cdot \mathrm{Prob}(s', \phi_1 \ U^{[0,t'-t]} \ \phi_2)$$

sum over states satisfying ϕ_1

Probability of reaching state s’ at time t and satisfying ϕ_1 up until this point

probability $\phi_1 \ U^{[0,t'-t]} \ \phi_2$ holds in s’

Time-bounded until – $P_{\sim p}\,[\phi_1\ U^{[t,t']}\ \phi_2]$

- Let $\mathrm{Prob}_{\phi_1}(s,\ \phi_1 U^{[0,t'-t]} \phi_2)$ = $\mathrm{Prob}(s,\ \phi_1 U^{[0,t'-t]} \phi_2)$ if $s \in \mathrm{Sat}(\phi_1)$ and 0 otherwise
- From the previous slide we have:

$$
\begin{aligned}
\underline{\mathrm{Prob}}(\phi_1\ U^{[t,t']}\ \phi_2) &= \Pi_t^{C[\neg\phi_1]} \cdot \underline{\mathrm{Prob}}_{\phi_1}(\phi_1\ U^{[0,t'-t]}\ \phi_2) \\
&= \left(\sum\nolimits_{i=0}^{\infty} \gamma_{q \cdot t,i} \cdot \left(P^{\mathrm{unif}(C[\neg\phi_1])}\right)^i \right) \underline{\mathrm{Prob}}_{\phi_1}(\phi_1\ U^{[0,t'-t]}\ \phi_2) \\
&= \sum\nolimits_{i=0}^{\infty} \left(\gamma_{q \cdot t,i} \cdot \left(P^{\mathrm{unif}(C[\neg\phi_1])}\right)^i \cdot \underline{\mathrm{Prob}}_{\phi_1}(\phi_1\ U^{[0,t'-t]}\ \phi_2) \right)
\end{aligned}
$$

  - summation can be truncated using Fox and Glynn [FG88]
  - can compute iteratively (only scalar and matrix-vector operations)

Time-bounded until – $P_{\sim p}\,[\phi_1\ U^{[t,\infty)}\ \phi_2]$

- Similar to the case for $\phi_1\ U^{[t,t']}\ \phi_2$ except second part is now unbounded, and hence the embedded DTMC can be used
- 1. Probability of remaining in ϕ_1 states until time t
- 2. Probability of reaching a ϕ_2 state, while remaining in states satisfying ϕ_1
  - i.e. computing Prob($\phi_1\ U^{[0,\infty)}\ \phi_2$)

$$\mathrm{Prob}(s,\phi_1\ U^{[t,\infty]}\ \phi_2) = \sum_{s'\in Sat(\phi_1)} \underline{\pi}_{s,t}^{C[\neg\phi_1]}(s')\cdot \mathrm{Prob}^{emb(C)}(s',\phi_1\ U\ \phi_2)$$

sum over states satisfying ϕ_1

Probability of reaching state s’ at time t and satisfying ϕ_1 up until this point

probability $\phi_1\ U^{[0,\infty)}\ \phi_2$ holds in s’

Time-bounded until – $P_{\sim p}\,[\phi_1\ U^{[t,\infty)}\ \phi_2]$

- Letting $Prob_{\phi 1}(s,\ \phi_1 U^{[0,\infty)}\phi_2) = Prob(s,\ \phi_1 U^{[0,\infty)}\phi_2)$ if $s \in Sat(\phi_1)$ and 0 otherwise, we have:

$$\underline{Prob}(\phi_1\ U^{[t,\infty]}\ \phi_2) \;=\; \Pi_t^{C[\neg\phi_1]} \cdot \underline{Prob}_{\phi_1}^{emb(C)}(\phi_1\ U\ \phi_2)$$

$$= \left(\sum_{i=0}^{\infty} \gamma_{q\cdot t,i} \cdot \left(P^{unif(C[\neg\phi_1])}\right)^i\right) \underline{Prob}_{\phi_1}^{emb(C)}(\phi_1\ U\ \phi_2)$$

$$= \sum_{i=0}^{\infty} \left(\gamma_{q\cdot t,i} \cdot \left(P^{unif(C[\neg\phi_1])}\right)^i \cdot \underline{Prob}_{\phi_1}^{emb(C)}(\phi_1\ U\ \phi_2)\right)$$

  - summation can be truncated using Fox and Glynn [FG88]
  - can compute iteratively (only scalar and matrix-vector operations

Model Checking – $S_{\sim p}[\ \phi\ ]$

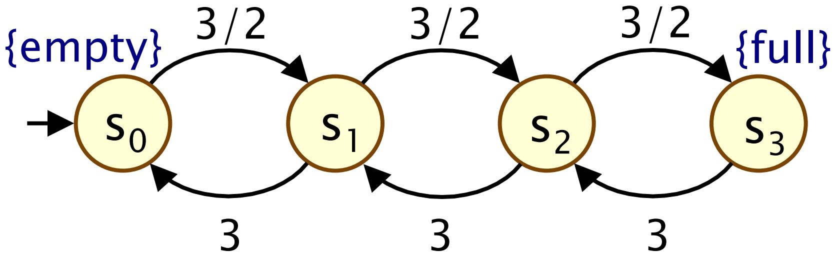
- A state s satisfies the formula $S_{\sim p}[\phi]$ if $\Sigma_{s' \models \phi}\ \underline{\pi}^C_s(s') \sim p$
  - $\underline{\pi}^C_s(s')$ is probability, having started in state s, of being in state s' in the long run
- Thus reduces to computing and then summing steady-state probabilities for the CTMC

- If CTMC is irreducible:
  - solution of one linear equation system
- If CTMC is reducible:
  - determine set of BSCCs for the CTMC
  - solve two linear equation systems for each BSCC T
  - one to obtain the vector $\underline{\text{ProbReach}}^{emb(C)}(T)$
  - the other to compute the steady state probabilities $\underline{\pi}^T$ for T

$S_{\sim p}$ [ϕ] – Example

- $S_{<0.1}$[full]
- CTMC is irreducible (comprises a single BSCC)
  - steady state probabilities independent of starting state
  - can be computed by solving $\underline{\pi}\cdot\mathbf{Q}=0$ and $\Sigma\ \underline{\pi}(s)=1$

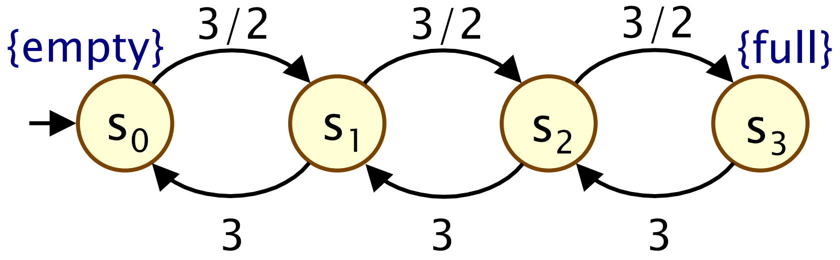
$$\mathbf{Q} = \begin{bmatrix} -3/2 & 3/2 & 0 & 0 \\ 3 & -9/2 & 3/2 & 0 \\ 0 & 3 & -9/2 & 3/2 \\ 0 & 0 & 3 & -3 \end{bmatrix}$$



$S_{\sim p}[\ \phi\ ]$ – Example

$$\begin{aligned}
-3/2\cdot\underline{\pi}(s_0) + 3\cdot\underline{\pi}(s_1) &= 0\\
3/2\cdot\underline{\pi}(s_0) - 9/2\cdot\underline{\pi}(s_1) + 3\cdot\underline{\pi}(s_2) &= 0\\
3/2\cdot\underline{\pi}(s_1) - 9/2\cdot\underline{\pi}(s_2) + 3\cdot\underline{\pi}(s_3) &= 0\\
3/2\cdot\underline{\pi}(s_2) - 3\cdot\underline{\pi}(s_3) &= 0
\end{aligned}$$

$$\underline{\pi}(s_0) + \underline{\pi}(s_1) + \underline{\pi}(s_2) + \underline{\pi}(s_3) = 1$$



- solution: $\underline{\pi} = [\ 8/15, 4/15, 2/15, 1/15\ ]$
- $\Sigma_{s' \vDash \mathrm{Sat(full)}}\ \underline{\pi}\,(s') = 1/15 < 0.1$
- so all states satisfy $S_{<0.1}[\ \mathrm{full}\ ]$

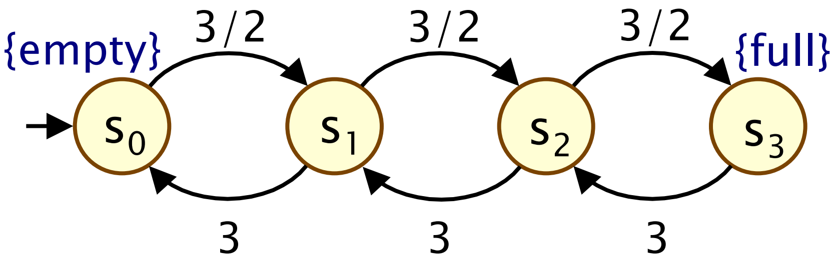
Rewards (or costs)

- Like DTMCs, we can augment CTMCs with rewards
  - real-valued quantities assigned to states and/or transitions
  - can be interpreted in two ways: instantaneous/cumulative
  - properties considered here: expected value of rewards
  - formal property specifications in an extension of CSL

- For a CTMC $(S,s_{init},\mathbf{R},L)$, a reward structure is a pair $(\underline{\rho},\iota)$
  - $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ is a vector of state rewards
  - $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a matrix of transition rewards

- For cumulative reward-based properties of CTMCs
  - state rewards interpreted as rate at which reward gained
  - if the CTMC remains in state s for $t \in \mathbb{R}_{>0}$ time units, a reward of $t \cdot \underline{\rho}(s)$ is acquired

Reward structures – Examples



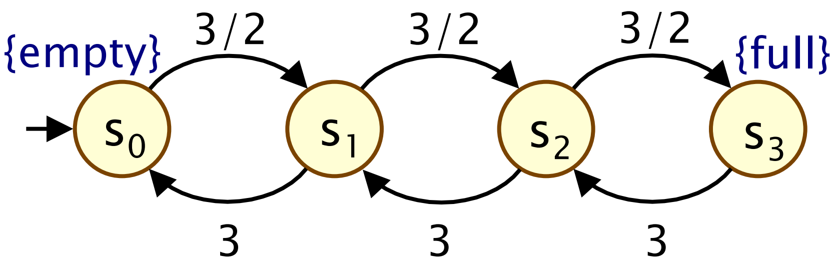
instantaneous

- Example: “size of message queue”
  - $\underline{\rho}(s_i)=i$ and $\iota(s_i,s_j)=0\ \forall i,j$

cumulative

- Example: “time for which queue is not full”
  - $\underline{\rho}(s_i)=1$ for $i<3$, $\underline{\rho}(s_3)=0$ and $\iota(s_i,s_j)=0\ \forall i,j$

Reward structures – Examples



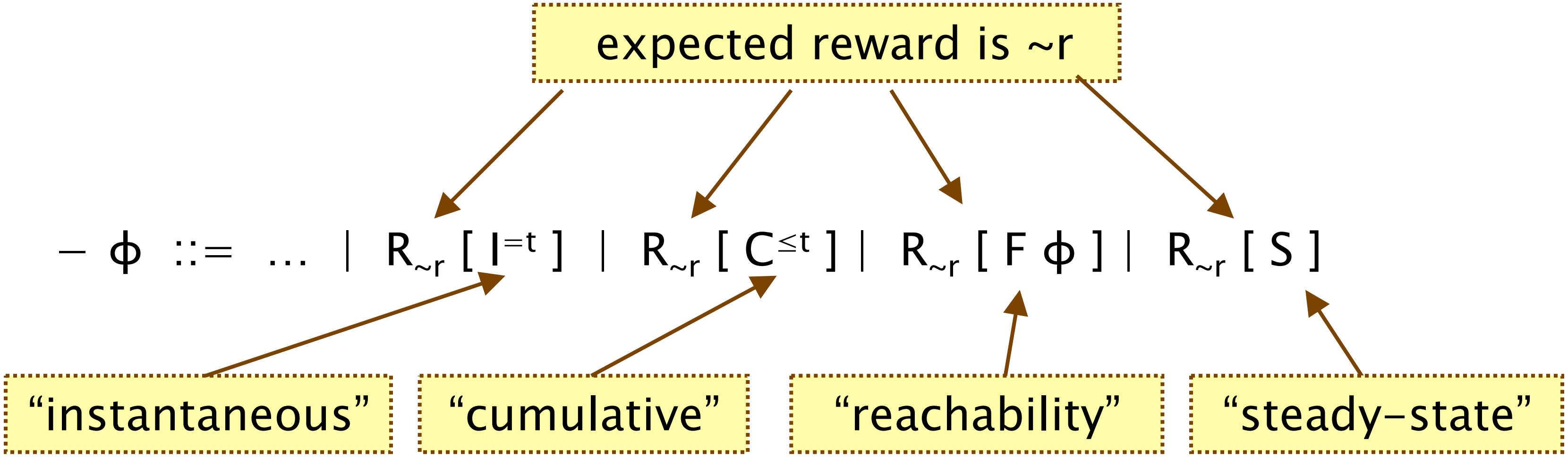
cumulative

- Example: “number of requests served”

$$\rho = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \iota = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

CSL and rewards

- PRISM extends CSL to incorporate reward-based properties
  - adds R operator like the one added to PCTL



expected reward is ~r

  - ϕ ::= … | $R_{\sim r}$ [$I^{=t}$] | $R_{\sim r}$ [$C^{\leq t}$] | $R_{\sim r}$ [F ϕ] | $R_{\sim r}$ [S]

“instantaneous” “cumulative” “reachability” “steady-state”

  - where r,t $\in \mathbb{R}_{\geq 0}$, ~ $\in \{<,>,\leq,\geq\}$

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

Types of reward formulae

- Instantaneous: $R_{\sim r}$ [$I^{=t}$]
  - the expected value of the reward at time-instant t is ~r
  - "the expected queue size after 6.7 seconds is at most 2"
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$]
  - the expected reward cumulated up to time-instant t is ~r
  - "the expected requests served within the first 4.5 seconds of operation is less than 10"
- Reachability: $R_{\sim r}$ [F ϕ]
  - the expected reward cumulated before reaching ϕ is ~r
  - "the expected requests served before the queue becomes full"
- Steady-state $R_{\sim r}$ [S]
  - the long-run average expected reward is ~r
  - "expected long-run queue size is at least 1.2"

Reward properties in PRISM

- Quantitative form:
  - e.g. $R_{=?}$ [$C^{\leq t}$]
  - what is the expected reward cumulated up to time-instant t?

- Add labels to R operator to distinguish between multiple reward structures defined on the same CTMC
  - e.g. $R_{\{num\_req\}=?}$ [$C^{\leq 4.5}$]
  - "the expected number of requests served within the first 4.5 seconds of operation"
  - e.g. $R_{\{pow\}=?}$ [$C^{\leq 4.5}$]
  - "the expected power consumption within the first 4.5 seconds of operation"

Reward formula semantics

- Formal semantics of the four reward operators:
  - $s \models R_{\sim r}[\,I^{=t}\,]$ $\Leftrightarrow$ $Exp(s, X_{I=t}) \sim r$
  - $s \models R_{\sim r}[\,C^{\leq t}\,]$ $\Leftrightarrow$ $Exp(s, X_{C\leq t}) \sim r$
  - $s \models R_{\sim r}[\,F\,\Phi\,]$ $\Leftrightarrow$ $Exp(s, X_{F\Phi}) \sim r$
  - $s \models R_{\sim r}[\,S\,]$ $\Leftrightarrow$ $\lim_{t\to\infty}(\,1/t \cdot Exp(s, X_{C\leq t})\,) \sim r$

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

Reward formula semantics

- Definition of random variables:
  - path $\omega = s_0 t_0 s_1 t_1 s_2 \ldots$

state of ω at time t

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

time spent in state s_i

time spent in state s_{j_t} before t time units have elapsed

$$X_{C\leq t}(\omega) = \sum_{i=0}^{j_t-1} \left(t_i \cdot \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) \right) + \left(t - \sum_{i=0}^{j_t-1} t_i \right) \cdot \underline{\rho}(s_{j_t})$$

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

  - where $j_t = \min\{ j \mid \Sigma_{i \leq j} t_i \geq t \}$ and $k_\phi = \min\{ i \mid s_i \vDash \phi \}$

Model checking reward formulae

- Instantaneous: $R_{\sim r}$ [$I^{=t}$]
  - reduces to transient analysis (state of the CTMC at time t)
  - use uniformisation
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$]
  - extends approach for time-bounded until
  - based on uniformisation
- Reachability: $R_{\sim r}$ [F ϕ]
  - can be computed on the embedded DTMC
  - reduces to solving a system of linear equations
- Steady-state: $R_{\sim r}$ [S]
  - similar to steady state formulae $S_{\sim r}$ [ϕ]
  - graph based analysis (compute BSCCs)
  - solve systems of linear equations (compute steady state probabilities of each BSCC)

CSL model checking complexity

- For model checking of a CTMC complexity:
  - linear in $|\Phi|$ and polynomial in $|S|$
  - linear in $q \cdot t_{max}$ (t_{max} is maximum finite bound in intervals)
- $P_{\sim p}[\Phi_1 \ U^{[0,\infty)} \ \Phi_2]$, $S_{\sim p}[\Phi]$, $R_{\sim r}$ [F Φ] and $R_{\sim r}$ [S]
  - require solution of linear equation system of size $|S|$
  - can be solved with Gaussian elimination: cubic in $|S|$
  - precomputation algorithms (max $|S|$ steps)
- $P_{\sim p}[\Phi_1 \ U^I \ \Phi_2]$, $R_{\sim r}$ $[C^{\leq t}]$ and $R_{\sim r}$ $[I^{=t}]$
  - at most two iterative sequences of matrix-vector products
  - operation is quadratic in the size of the matrix, i.e. $|S|$
  - total number of iterations bounded by Fox and Glynn
  - the bound is linear in the size of $q \cdot t$ (q uniformisation rate)

Summing up…

- Model checking a CSL formula ϕ on a CTMC
  - recursive: bottom-up traversal of parse tree of ϕ
- Main work: computing probabilities for P and S operators
  - untimed ($X\ \Phi$, $\Phi_1\ U\ \Phi_2$): perform on embedded DTMC
  - time-bounded until: use uniformisation-based methods, rather than more expensive solution of integral equations
  - other forms of time-bounded until, i.e. $[t_1,t_2]$ and $[t,\infty)$, reduce to two sequential computations like for $[0,t]$
  - S operator: summation of steady-state probabilities
- Rewards – similar to DTMCs
  - except for continuous-time accumulation of state rewards
  - extension of CSL with R operator
  - model checking of R comparable with that of P

Lecture 11
Counterexamples + Bisimulation

Dr. Dave Parker

Department of Computer Science
University of Oxford

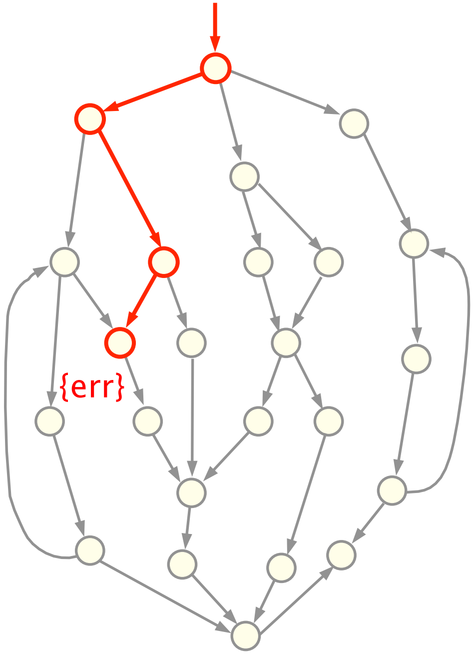
Overview

- Counterexamples
  - non-probabilistic model checking
  - counterexamples for PCTL + DTMCs
  - computing smallest counterexamples

- Bisimulation
  - bisimulation equivalences: DTMCs, CTMCs
  - preservation of logics: PCTL, CSL
  - bisimulation minimisation

Non probabilistic counterexamples

- Counterexamples (for non-probabilistic model checking)
  - generated when model checking a (universal) property fails
  - trace through model illustrating why property does not hold
  - major advantage of the model checking approach
  - bug finding vs. verification
- Example:
  - CTL property AG ¬err
  - (or equivalently, ¬EF err)
  - ("an error state is never reached")
  - counterexample is a finite trace to a state satisfying err
  - alternatively, this is a witness to the satisfaction of formula EF err



Counterexamples for DTMCs?

- PCTL example: $P_{<0.01}$ [F err]
  - "the probability of reaching an error state is less than 0.01"
  - what is a counterexample for $s \not\models P_{<0.01}$ [F err] ?
  - not necessarily illustrated by a single trace to an err state
  - in fact, "counterexample" is a set of paths satisfying F err whose combined measure is greater than or equal to 0.01
- Alternative approach to "debugging" seen so far:
  - probabilistic model checker provides actual probabilities
  - e.g. queries of the form $P_{=?}$ [F err]
  - anomalous behaviour identified by examining trends
  - e.g. $P_{=?}$ [$F^{\leq T}$ err] for T=0,…,100
- This lecture: DTMC counterexamples in style of [HK07]
  - also some work done on CTMC/MDP counterexamples

DTMC notation

- DTMC: $D = (S, s_{init}, \mathbf{P}, L)$
- $Path(s)$ = set of all infinite paths starting in state s
- $Pr_s : \Sigma_{Path(s)} \rightarrow [0,1]$ = probability measure over infinite paths
  - where $\Sigma_{Path(s)}$ is the σ-algebra on Path(s)
  - defined in terms of probabilities for finite paths
- $\mathbf{P}_s(\omega)$ = probability for finite path $\omega = ss_1 \ldots s_n$
  - $\mathbf{P}_s(s) = 1$
  - $\mathbf{P}_s(ss_1 \ldots s_n) = \mathbf{P}(s,s_1) \cdot \mathbf{P}(s_1,s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1},s_n)$
  - extend notation to sets: $\mathbf{P}_s(C)$ for set of finite paths C
  - $\mathbf{P}_s$ extends uniquely to Pr_s
- $Path(s, \psi) = \{ \omega \in Path(s) \mid \omega \vDash \psi \}$
  - $Prob(s, \psi) = Pr_s(Path(s, \psi))$
- $Path_{fin}(s, \psi)$ = set of finite paths from s satisfying ψ

Counterexamples for DTMCs

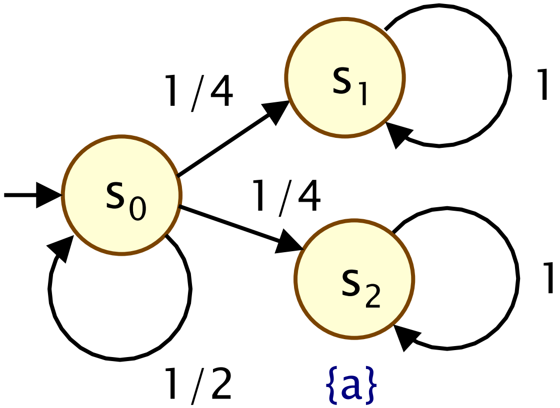
- Consider PCTL properties of the form:
  - $P_{\leq p} [\Phi_1 \ U^{\leq k} \ \Phi_2]$, where $k \in \mathbb{N} \cup \{\infty\}$
  - i.e. bounded or unbounded until formulae with closed upper probability bounds

- Refutation:
  - $s \nvDash P_{\leq p} [\Phi_1 \ U^{\leq k} \ \Phi_2]$
  - $\Leftrightarrow Pr_s(Path(s, \Phi_1 \ U^{\leq k} \ \Phi_2)) > p$
  - i.e. total probability mass of $\Phi_1 \ U^{\leq k} \ \Phi_2$ paths exceeds p

- Since the property is an until formula
  - this is evidenced by a set of finite paths

Counterexamples for DTMCs

- A counterexample for $P_{\leq p}$ [Φ_1 $U^{\leq k}$ Φ_2] in state s is:
  - a set C of finite paths such that $C \subseteq Path_{fin}(s, \psi)$ and $P_s(C) > p$

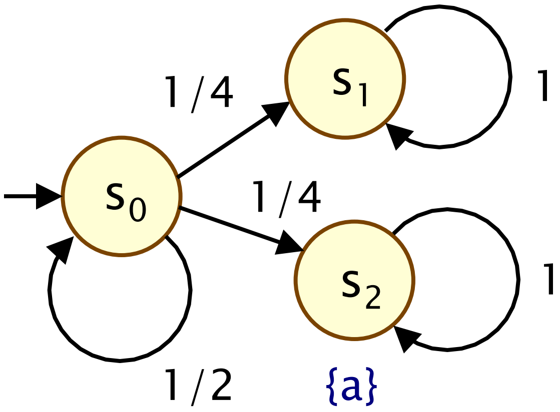


- Example
  - Consider the PCTL formula:
  - $P_{\leq 0.3}$ [F a]
  - This is not satisfied in s_0
  - Prob(s_0, F a) = 1/4+1/8+1/16+… = 1/2
  - A counterexample: C = { s_0s_2, $s_0s_0s_2$ }
  - $P_{s0}(C) = 1/4 + (1/2)(1/4) = 3/8 = 0.375$

Finiteness of counterexamples

- There is always a finite counterexample for:
  - $s \nvDash P_{\leq p} [\Phi_1 \ U^{\leq k} \ \Phi_2]$

- On the other hand, consider this DTMC:
  - and the PCTL formula:
  - $P_{<1/2} [F \ a]$
  - $\text{Prob}(s_0, F \ a) = 1/4+1/8+1/16+\ldots$
    $= 1/2$
  - $s_0 \nvDash P_{<1/2} [F \ a]$
  - counterexample would require infinite set of paths
  - $\{ (s_0)^i s_2 \}_{i \in \mathbb{N}}$

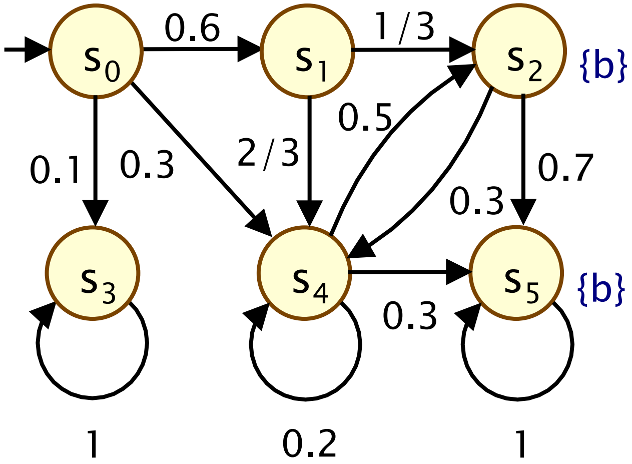


Counterexamples for DTMCs

- Aim: counterexamples should be succinct, comprehensible
- Set of all counterexamples:
  - $CX_p(s,\psi)$ = set of all counterexamples for $P_{\leq p}$ $[\psi]$ in state s
- Minimal counterexample
  - counterexample C with $|C| \leq |C'|$ for all $C' \in CX_p(s,\psi)$
- "Smallest" counterexample
  - minimal counterexample C with $\mathbf{P}(C) \geq \mathbf{P}(C')$ for all minimal $C' \in CX_p(s,\psi)$
  - reduces to finding…
- Strongest (most probable) evidence
  - finite path ω in $Path_{fin}(s, \psi)$ such that $\mathbf{P}(\omega) \geq \mathbf{P}(\omega')$ for all $\omega' \in Path_{fin}(s, \psi)$
  - i.e. contributes most to violation of PCTL formula

Example

- PCTL formula: $P_{\leq 1/2}$ [F b]
  - $s_0 \nvDash P_{\leq 1/2}$ [F b]
  - since Prob(s_0, F b) = 0.9



- Counterexamples:
  - $C_1 = \{ s_0s_1s_2, s_0s_1s_4s_2, s_0s_1s_4s_5, s_0s_4s_2 \}$
    - $P_{s0}(C_1) = 0.2+0.2+0.12+0.15 = 0.67$ (not minimal)
  - $C_2 = \{ s_0s_1s_2, s_0s_1s_4s_2, s_0s_1s_4s_5 \}$
    - $P_{s0}(C_2) = 0.2+0.2+0.12 = 0.52$ (not “smallest”)
  - $C_3 = \{ s_0s_1s_2, s_0s_1s_4s_2, s_0s_4s_2 \}$
    - $P_{s0}(C_3) = 0.2+0.2+0.15 = 0.55$ (“smallest”)

Weighted digraphs

- A weighted directed graph is a tuple $G = (V, E, w)$ where:
  - V is a set of vertices
  - $E \subseteq V \times V$ is a set of edges
  - $w : E \rightarrow \mathbb{R}_{\geq 0}$ is a weight function

- Finite path ω in G
  - is a sequence of vertices $v_0 v_1 v_2 \dots v_n$ such that $(v_i, v_{i+1}) \in E \ \forall i \geq 0$
  - the distance of $\omega = v_0 v_1 v_2 \dots v_n$ is: $\Sigma_{i=0 \dots n-1} w(v_i, v_{i+1})$

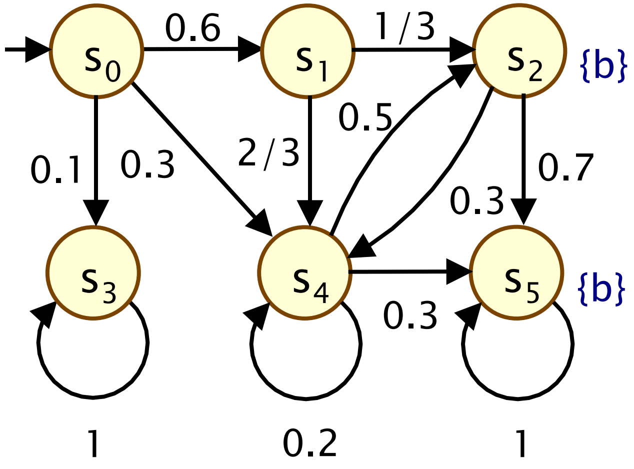
- Shortest path problem
  - given a weighted digraph, find a path between two vertices v_1 and v_2 with the smallest distance
  - i.e. a path ω s.t. $d(\omega) \leq d(\omega')$ for all other such paths ω'

Finding strongest evidences

- Reduction to graph problem…
- Step 1: Adapt the DTMC
  - make states satisfying $\neg\Phi_1 \wedge \neg\Phi_2$ absorbing
    - (i.e. replace all outgoing transitions with a single self-loop)
  - add an extra state t and replace all transitions from any Φ_2 state with a single transition to t (with probability 1)
- Step 2: Convert new DTMC into a weighted digraph
  - for the (adapted) DTMC $D = (S, s_{init}, \mathbf{P}, L)$:
  - corresponding graph is $G_D = (V, E, w)$ where:
  - $V = S$ and $E = \{ (s,s') \in S \times S \mid \mathbf{P}(s,s') > 0 \}$
  - $w(s,s') = \log(1 / \mathbf{P}(s,s'))$
- Key idea: for any two paths ω and ω' in D (and in G_D)
  - $\mathbf{P}_s(\omega') \geq \mathbf{P}_s(\omega)$ if and only if $d(\omega') \leq d(\omega)$

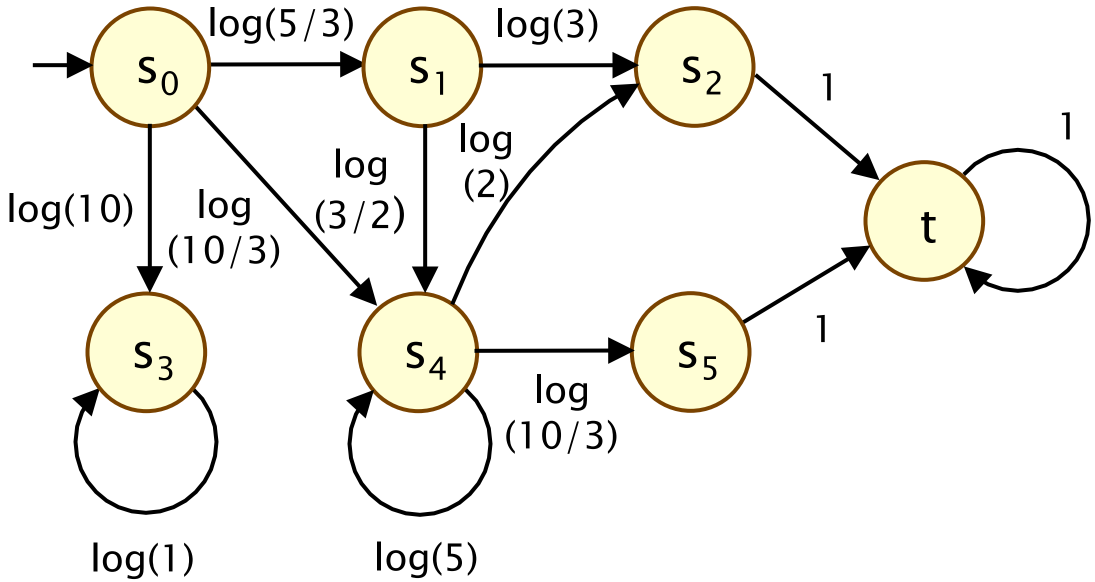
Example…

- PCTL formula: $P_{\leq 1/2}$ [F b]



DTMC

weighted digraph



Finding strongest evidences

- To find strongest evidence in DTMC D
  - analyse corresponding digraph
- For unbounded until formula $P_{\leq p}$ [Φ_1 U Φ_2]
  - solve shortest path problem in digraph (target t)
  - polynomial time algorithms exist
    - e.g. Dijsktra's algorithm can be implemented in $O(|E|+|V|\cdot\log|V|)$
- For bounded until formula $P_{\leq p}$ [Φ_1 $U^{\leq k}$ Φ_2]
  - solve special case of the constrained shortest path problem
  - also solvable in polynomial time
- Generation of smallest counterexamples
  - based on computation of k shortest paths
  - k can be computed on the fly

Other cases

- Lower bounds on probabilities
  - i.e. $s \not\models P_{\geq p} [\Phi_1 \; U^{\leq k} \; \Phi_2]$
  - negate until formula to reverse probability bound
  - solvable with BSCC computation + probabilistic reachability
  - for details, see [HK07]

- Continuous-time Markov chains
  - these techniques can be extended to CTMCs and CSL [HK07b]
  - naïve approach: apply DTMC techniques to uniformised DTMC
  - modifications required to get smaller counterexamples
  - another possibility: directed search based techniques [AHL05]

Bisimulation

- Identifies models with the same branching structure
  - i.e. the same stepwise behaviour
  - each model can simulate the actions of the other
  - guarantees that models satisfy many of the same properties

- Uses of bisimulation:
  - show equivalence between a model and its specification
  - state space reduction: bisimulation minimisation

- Formally, bisimulation is an equivalence relation over states
  - bisimilar states must have identical labelling and identical stepwise behaviour

Equivalence relations

- Let R be a relation over some set S
  - i.e. $R \subseteq S \times S$
  - we write s_1 R s_2 as shorthand for $(s_1,s_2) \in R$

- R is an equivalence relation iff:
  - R is reflexive, i.e. s R s
  - R is symmetric, i.e. if s_1 R s_2 then s_2 R s_1
  - R is transitive, i.e. if s_1 R s_2 and s_2 R s_3 then s_1 R s_3

- R partitions S:
  - equivalence classes: $[s]_R = \{ s' \in S \mid s' R s \}$
  - the quotient of S under R is denoted $S/R = \{ [s]_R \mid s \in S \}$

Bisimulation on DTMCs

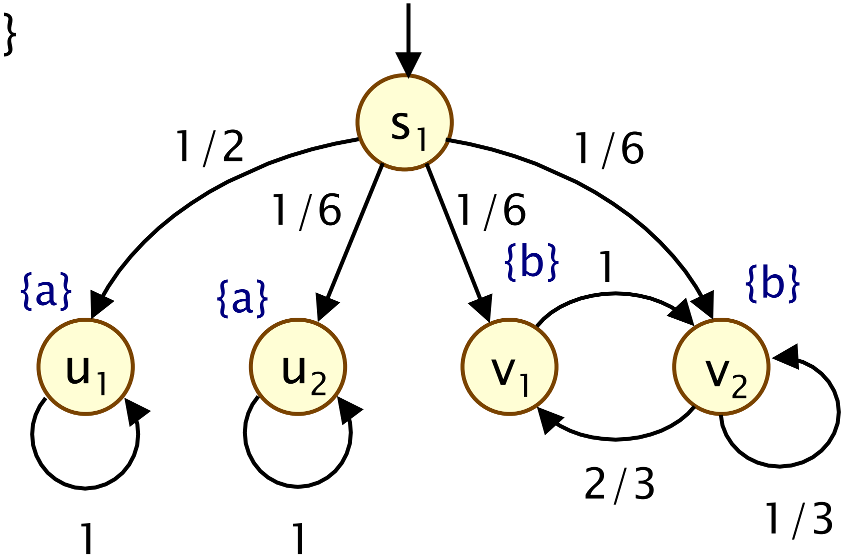
- Consider a DTMC D = $(S, s_{init}, \mathbf{P}, L)$
- Some notation:
  - $\mathbf{P}(s,T) = \Sigma_{s' \in T}\, \mathbf{P}(s,s')$ for $T \subseteq S$
- An equivalence relation R on S is a probabilistic bisimulation on D if and only if for all s_1 R s_2:
  - $L(s_1) = L(s_2)$
  - $\mathbf{P}(s_1, T) = \mathbf{P}(s_2, T)$ for all $T \in S/R$ (i.e. for all equivalence classes of R)
- States s_1 and s_2 are bisimulation-equivalent (or bisimilar)
  - if there exists a probabilistic bisimulation R on D with s_1 R s_2
  - denoted $s_1 \sim s_2$

Simple example

- Bisimulation relation ~

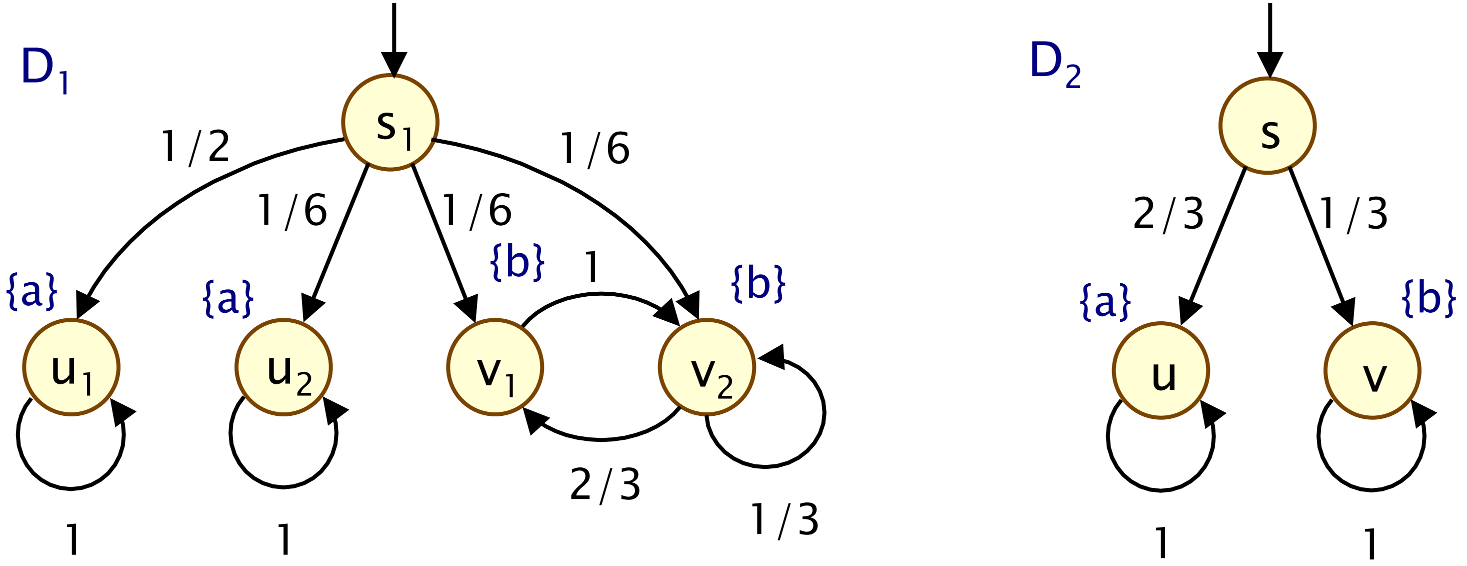
- Quotient of S under ~
  - $\{ \{s_1\}, \{u_1, u_2\}, \{v_1, v_2\} \}$

- Bisimilar states:
  - $u_1 \sim u_2$
  - $v_1 \sim v_2$



Bisimulation on DTMCs

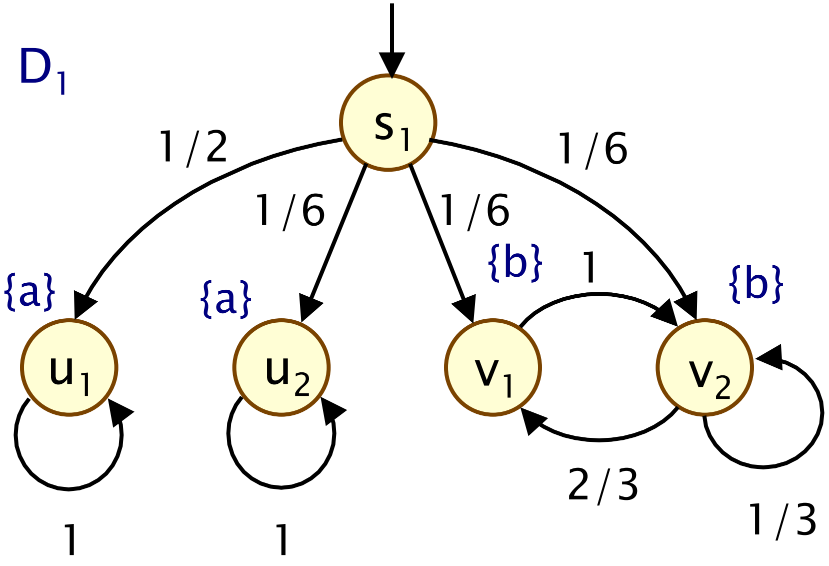
- Bisimulation between DTMCs D_1 and D_2
  - $D_1 \sim D_2$ if they have bisimilar initial states
- Formally:
  - state labellings for D_1 and D_2 over same set of atomic prop.s
  - bisimulation relation is over disjoint union of D_1 and D_2

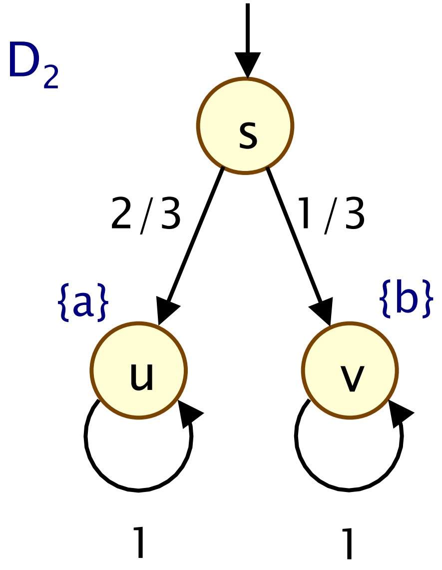


Simple example

- Bisimilar states:
  - $u_1 \sim u_2 \sim u$
  - $v_1 \sim v_2 \sim v$
  - $s_1 \sim s$

Bisimilar DTMCs: $D_1 \sim D_2$



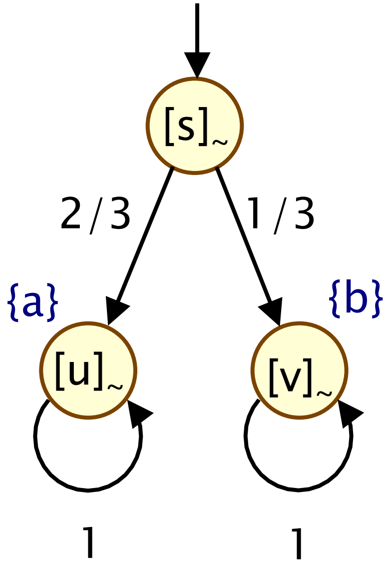


Quotient DTMC

- For a DTMC D = $(S,s_{init},\mathbf{P},L)$ and probabilistic bisimulation ~

- Quotient DTMC is
  - $D/\sim = (S',s'_{init},\mathbf{P}',L')$

- where:
  - $S' = S/\sim = \{ [s]_\sim \mid s \in S \}$
  - $s'_{init} = [s_{init}]_\sim$
  - $\mathbf{P}'([s]_\sim, [s']_\sim) = \mathbf{P}(s, [s']_\sim)$
  - $L'([s]_\sim) = L(s)$



well defined since bisimulation ensures $\mathbf{P}(s, [s']_\sim)$ same for all s in $[s]_\sim$

Bisimulation and PCTL

- Probabilistic bisimulation preserves all PCTL formulae

- For all states s and s':

$$s \sim s'$$

$$\Leftrightarrow$$

for all PCTL formulae Φ, $s \vDash \Phi$ if and only if $s' \vDash \Phi$

- Note also:
  - every pair of non-bisimilar states can be distinguished with some PCTL formula
  - $\sim$ is the coarsest relation with this property
  - in fact, bisimulation also preserves all PCTL* formulae

CTMC bisimulation

- Check equivalence of rates, not probabilities…

- An equivalence relation R on S is a probabilistic bisimulation on CTMC C=(S,s_{init},**R**,L) if and only if for all s_1 R s_2:
  - $L(s_1) = L(s_2)$
  - $\mathbf{R}(s_1, T) = \mathbf{R}(s_2, T)$ for all classes T in S/R

- Alternatively, check:
  - $L(s_1) = L(s_2)$, $\mathbf{P}^{emb(C)}(s_1, T) = \mathbf{P}^{emb(C)}(s_2, T)$, $E(s_1) = E(s_2)$

- Bisimulation on CTMCs preserves CSL
  - (see [BHHK03] and also [DP03])

Bisimulation minimisation

- More efficient to perform PCTL/CSL model checking on the quotient DTMC/CTMC
  - assuming quotient model can be constructed efficiently
  - (see [KKZJ07] for experimental results on this)

- Bisimulation minimisation
  - algorithm to construct quotient model
  - based on partition refinement
  - repeated splitting of an initially coarse partition
  - final partition is coarsest bisimulation wrt. initial partition
  - (optimisations/variants possible by changing initial partition)
  - complexity: $O(|\mathbf{P}|\cdot\log|S| + |AP|\cdot|S|)$ [DHS'03]
    - assuming suitable data structure used (splay trees)

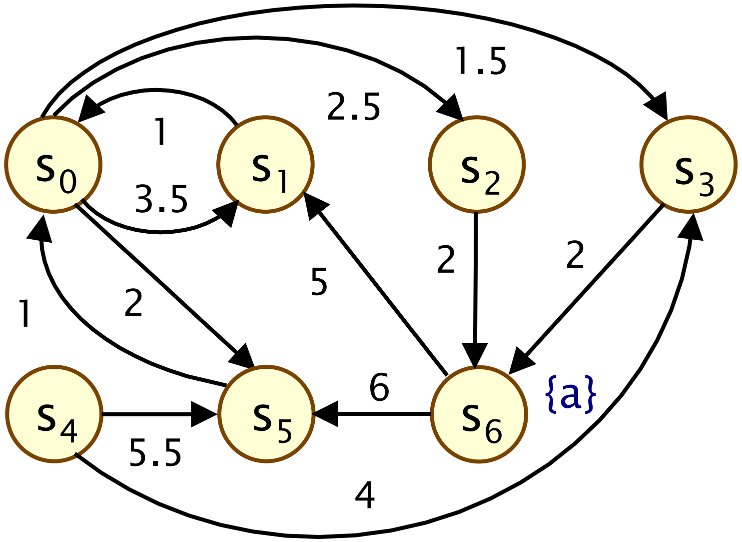
Bisimulation minimisation

- 1. Start with initial partition
  - say $\Pi = \{ \{ s \in S \mid L(s)=lab \} \mid lab \in 2^{AP} \}$
- 2. Find a splitter $T \in \Pi$ for some block $B \in \Pi$
  - a splitter T is a block such that probability of going to T differs for some states in block B
  - i.e. $\exists s,s' \in B \,.\, \mathbf{P}(s,T) \neq \mathbf{P}(s',T)$
- 3. Split B into sub-blocks
  - such that **P**(s,T) is the same for all states in each sub-block
- 4. Repeat steps 2/3 until no more splitters exist
  - i.e. no change to partition Π

replace **P** with **R** for CTMCs

CTMC example

- Consider model checking $P_{\sim p}$ [$F^{[0,t]}$ a] on this CTMC:



Minimisation:

Π_0: $B_1=\{s_0,s_1,s_2,s_3,s_4,s_5\}$, $B_2=\{s_6\}$

B_2 is a splitter for B_1

(since e.g. $\mathbf{R}(s_1,B_2)=0\neq2=\mathbf{R}(s_2,B_2)$)

Π_1: $B_1=\{s_0,s_1,s_4,s_5\}$, $B_2=\{s_6\}$, $B_3=\{s_2,s_3\}$

B_3 is a splitter for B_1

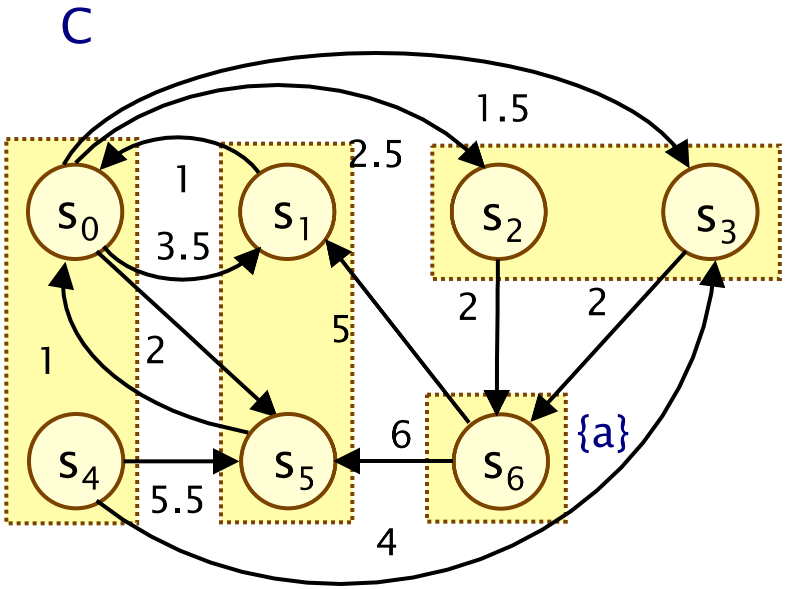
(since e.g. $\mathbf{R}(s_1,B_3)=0\neq4=\mathbf{R}(s_0,B_3)$)

Π_2: $B_1=\{s_1,s_5\}$, $B_2=\{s_6\}$, $B_3=\{s_2,s_3\}$, $B_4=\{s_0,s_4\}$

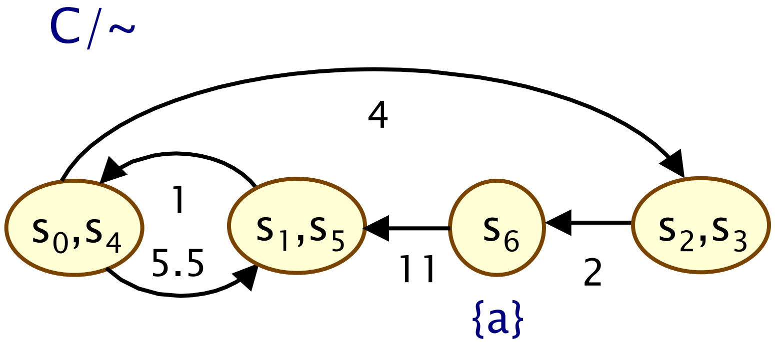
No more splitters…

$S/\sim$ = { $\{s_1,s_5\}$, $\{s_6\}$, $\{s_2,s_3\}$, $\{s_0,s_4\}$ }

CTMC example...



$S/\sim = \{ \{s_1,s_5\}, \{s_6\}, \{s_2,s_3\}, \{s_0,s_4\} \}$



$$\mathrm{Prob}^{C}(s_0, F^{[0,t]}\ a) \;=\; \mathrm{Prob}^{C/\sim}(\{s_0,s_4\}, F^{[0,t]}\ a)$$

Summing up…

- Counterexamples
  - essential ingredient of non-probabilistic model checking
  - counterexamples for PCTL + DTMCs
    - finite set of paths showing $\nvDash P_{\leq p} [\ \Phi_1\ U^{\leq k}\ \Phi_2\ ]$
  - computing smallest counterexamples
    - reduction to well-known graph problems

- Bisimulation
  - relates states/Markov chains with identical labelling and identical stepwise behaviour
  - preserves PCTL, CSL, …
  - bisimulation minimisation: automated reduction to quotient model

Lecture 12
Markov Decision Processes

Dr. Dave Parker

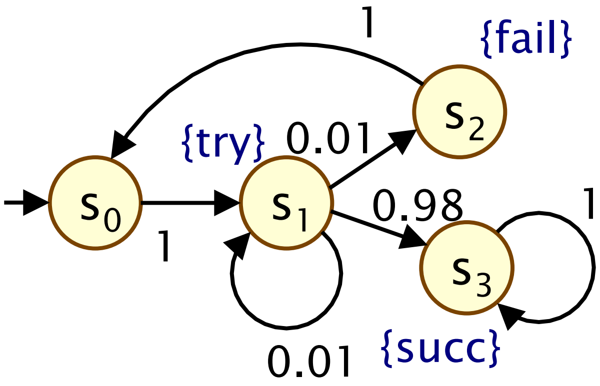
Department of Computer Science
University of Oxford

Overview

- Nondeterminism
- Markov decision processes (MDPs)
- Paths, probabilities and adversaries
- End components

Recap: DTMCs

- Discrete-time Markov chains (DTMCs)
  - discrete state space, transitions are discrete time-steps
  - from each state, choice of successor state (i.e. which transition) is determined by a discrete probability distribution



- DTMCs are fully probabilistic
  - well suited to modelling, for example, simple random algorithms or synchronous probabilistic systems where components move in lock-step

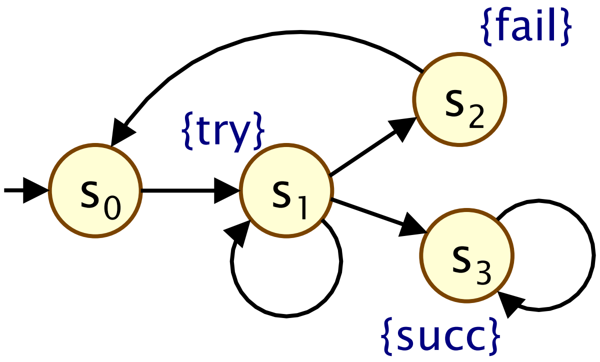
Nondeterminism

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

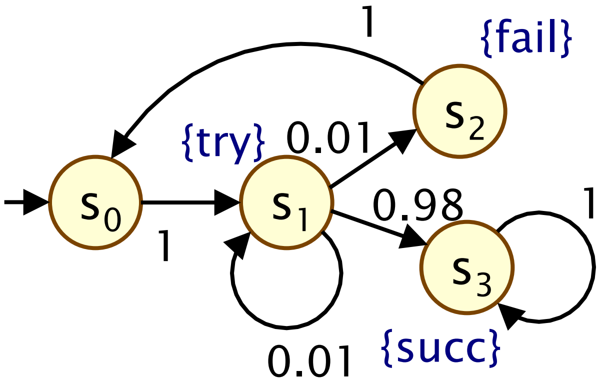
- Concurrency – scheduling of parallel components
  - e.g. randomised distributed algorithms – multiple probabilistic processes operating asynchronously
- Unknown environments
  - e.g. probabilistic security protocols – unknown adversary
- Underspecification – unknown model parameters
  - e.g. a probabilistic communication protocol designed for message propagation delays of between d_{min} and d_{max}
- Abstraction
  - e.g. partition DTMC into similar (but not identical) states

Probability vs. nondeterminism

- Labelled transition system
  - (S,s_0,R,L) where $R \subseteq S \times S$
  - choice is nondeterministic



- Discrete-time Markov chain
  - (S,s_0,P,L) where $P : S \times S \rightarrow [0,1]$
  - choice is probabilistic



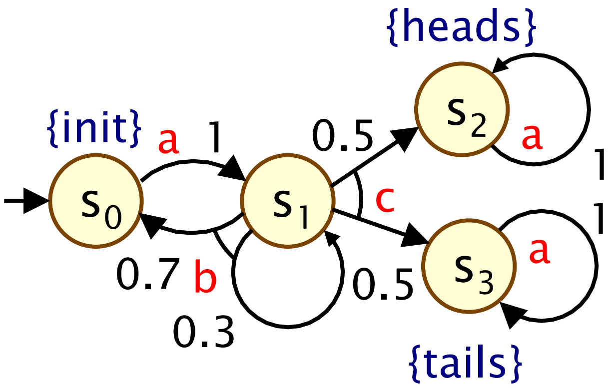
- How to combine?

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

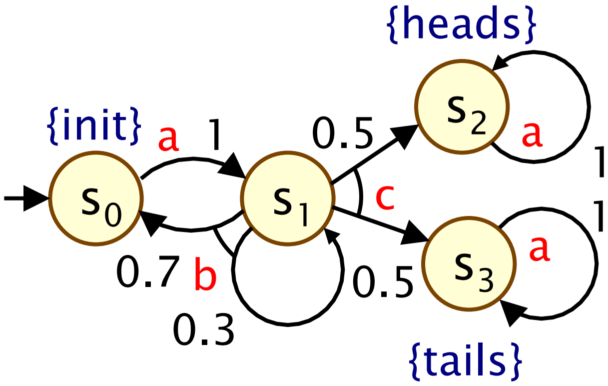


Markov decision processes

- Formally, an MDP M is a tuple $(S, s_{init}, \mathbf{Steps}, L)$ where:
  - S is a finite set of states ("state space")
  - $s_{init} \in S$ is the initial state
  - $\mathbf{Steps} : S \rightarrow 2^{Act \times Dist(S)}$ is the transition probability function

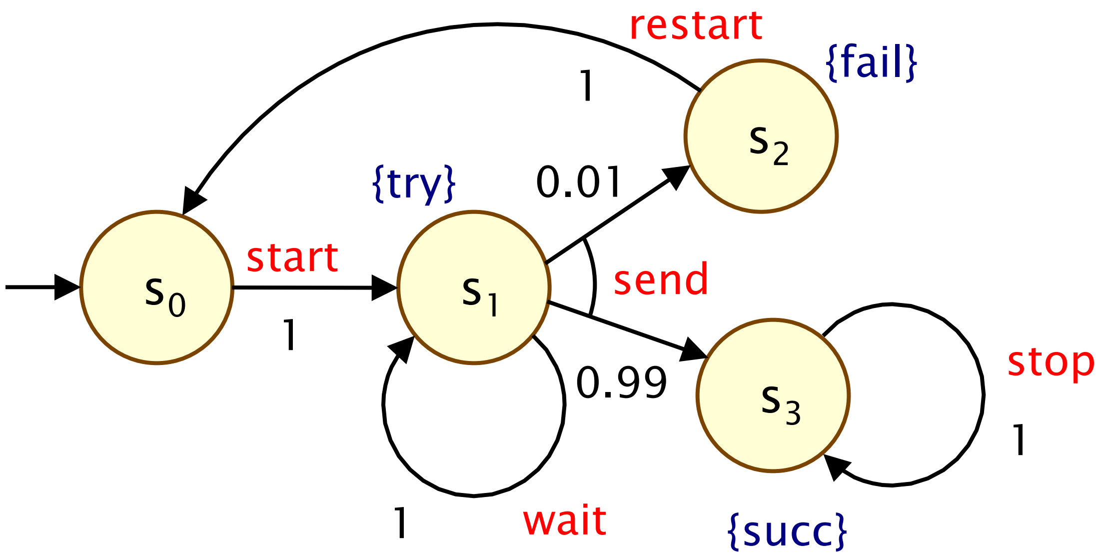
    where Act is a set of actions and Dist(S) is the set of discrete probability distributions over the set S
  - $L : S \rightarrow 2^{AP}$ is a labelling with atomic propositions

- Notes:
  - Steps(s) is always non-empty, i.e. no deadlocks
  - the use of actions to label distributions is optional



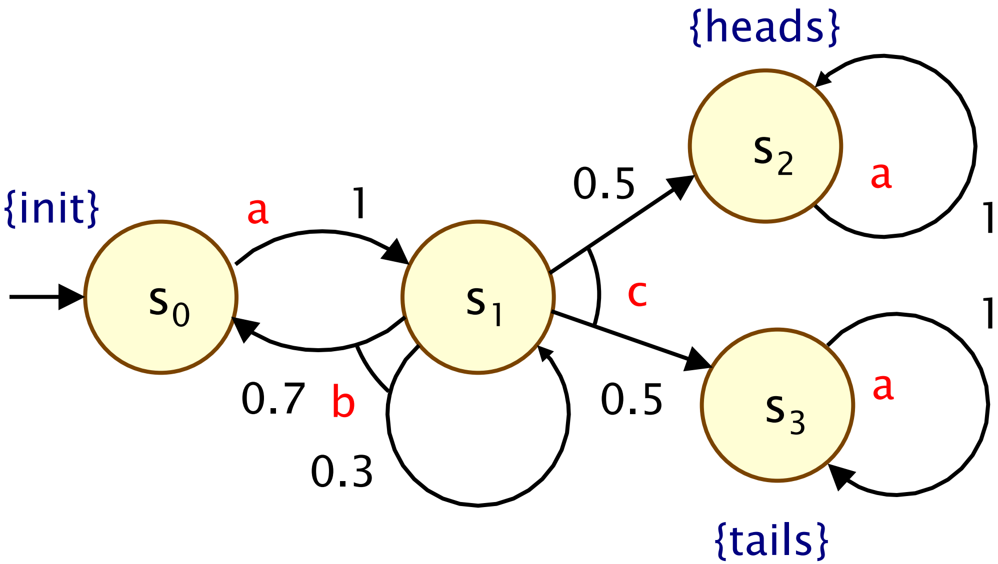
Simple MDP example

- Modification of the simple DTMC 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



Simple MDP example 2

- Another simple MDP example with four states
  - from state s_0, move directly to s_1 (action a)
  - in state s_1, nondeterministic choice between actions b and c
  - action b gives a probabilistic choice: self-loop or return to s_0
  - action c gives a 0.5/0.5 random choice between heads/tails



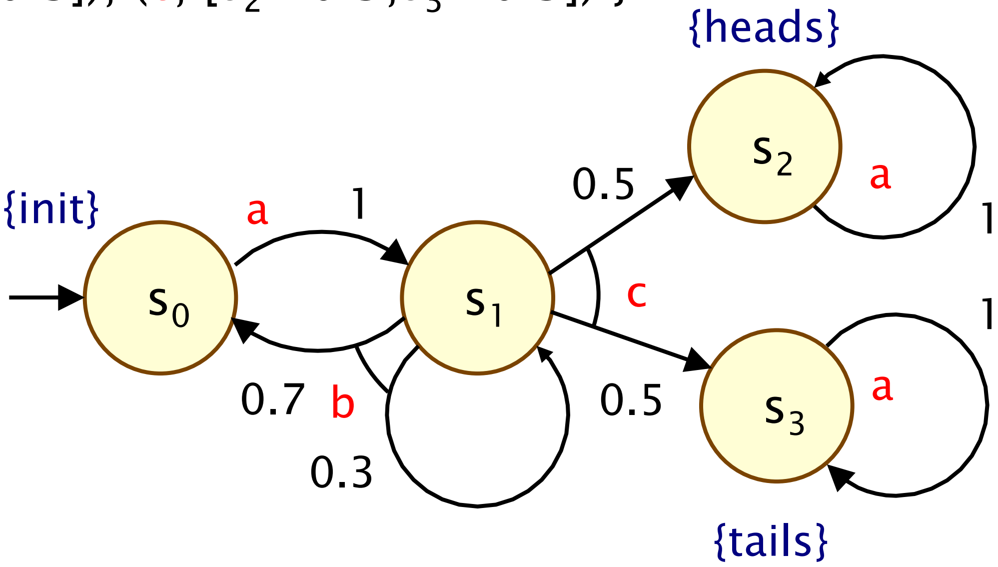
Simple MDP example 2

$M = (S, s_{init}, \mathbf{Steps}, L)$

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

$AP = \{init, heads, tails\}$
$L(s_0)=\{init\}$,
$L(s_1)=\varnothing$,
$L(s_2)=\{heads\}$,
$L(s_3)=\{tails\}$

$\mathbf{Steps}(s_0) = \{ (a, [s_1 \mapsto 1]) \}$
$\mathbf{Steps}(s_1) = \{ (b, [s_0 \mapsto 0.7, s_1 \mapsto 0.3]), (c, [s_2 \mapsto 0.5, s_3 \mapsto 0.5]) \}$
$\mathbf{Steps}(s_2) = \{ (a, [s_2 \mapsto 1]) \}$
$\mathbf{Steps}(s_3) = \{ (a, [s_3 \mapsto 1]) \}$



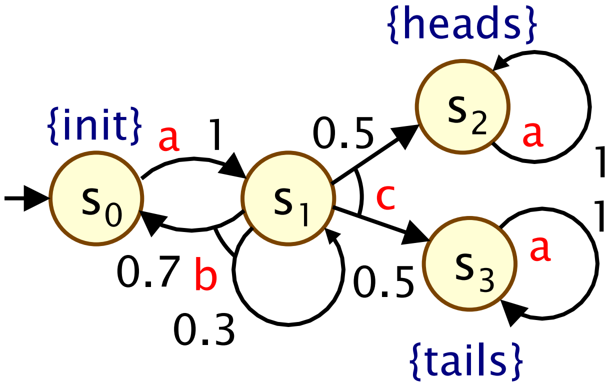
The transition probability function

- It is often useful to think of the function **Steps** as a matrix
  - non-square matrix with |S| columns and $\Sigma_{s \in S}$ |**Steps**(s)| rows

- Example (for clarity, we omit actions from the matrix)

Steps(s_0) = { (a, $s_1 \mapsto 1$) }
Steps(s_1) = { (b, [$s_0 \mapsto 0.7, s_1 \mapsto 0.3$]), (c, [$s_2 \mapsto 0.5, s_3 \mapsto 0.5$]) }
Steps(s_2) = { (a, $s_2 \mapsto 1$) }
Steps(s_3) = { (a, $s_3 \mapsto 1$) }

$$\text{Steps} = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$



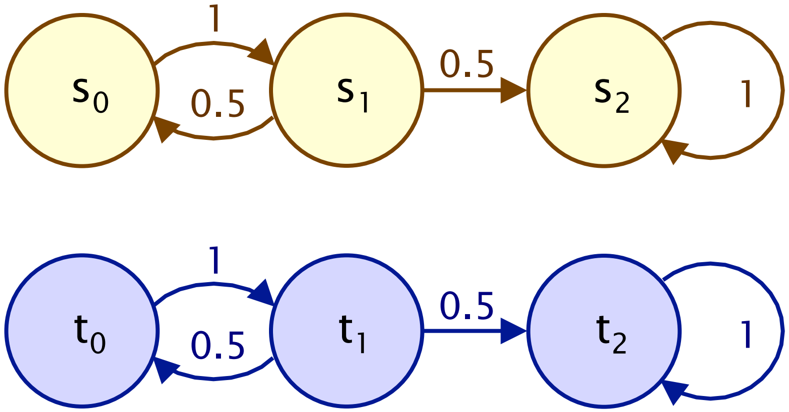
Example – Parallel composition

Asynchronous parallel composition of two 3-state DTMCs

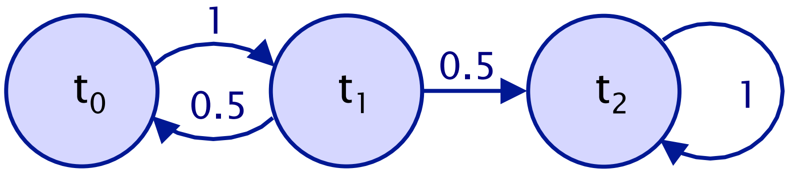
PRISM code:

```
module M1
    s : [0..2] init 0;
    [] s=0 -> (s'=1);
    [] s=1 -> 0.5:(s'=0) + 0.5:(s'=2);
    [] s=2 -> (s'=2);
endmodule
```

```
module M2 = M1 [ s=t ] endmodule
```

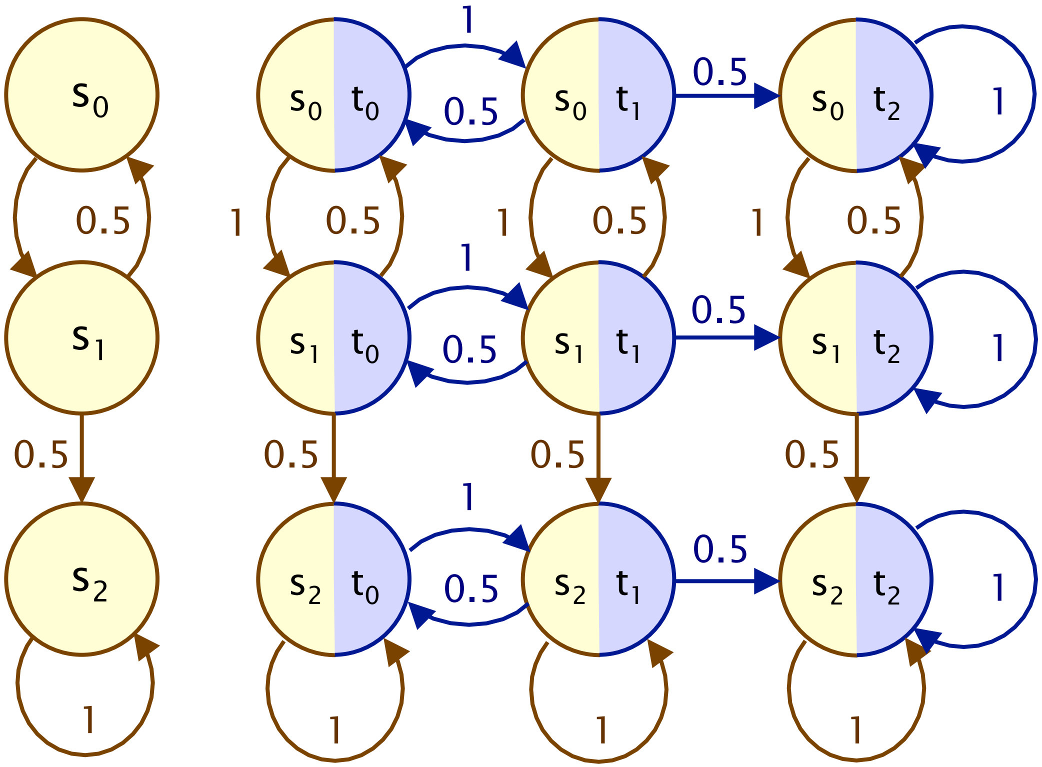


Example – Parallel composition



Asynchronous parallel composition of two 3-state DTMCs

Action labels omitted here

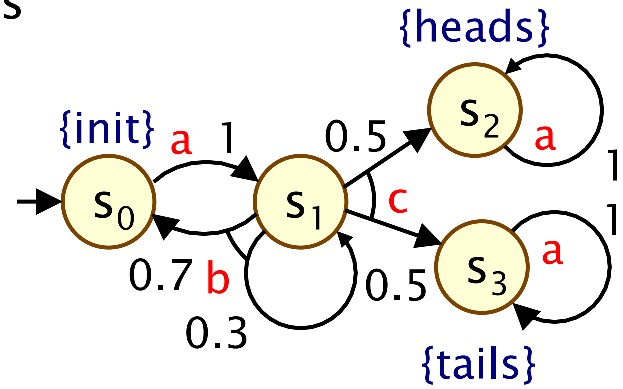


Paths and probabilities

- A (finite or infinite) path through an MDP
  - 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 $(a_i,\mu_i) \in$ **Steps**(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

- Path(s) = set of all paths through MDP starting in state s
  - $\text{Path}_{\text{fin}}(s)$ = set of all finite paths from s

- Paths resolve both nondeterministic and probabilistic choices
  - how to reason about probabilities?



Adversaries

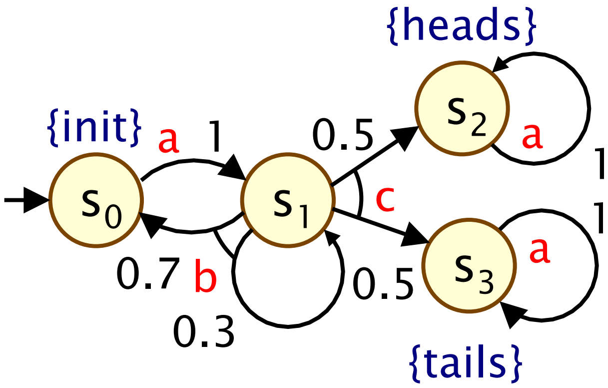
- To consider the probability of some behaviour of the MDP
  - first need to resolve the nondeterministic choices
  - …which results in a DTMC
  - …for which we can define a probability measure over paths

- An adversary resolves nondeterministic choice in an MDP
  - also known as "schedulers", "policies" or "strategies"
- Formally:
  - an adversary σ of an MDP M is a function mapping every finite path $\omega = s_0(a_0,\mu_0)s_1 \ldots s_n$ to an element $\sigma(\omega)$ of **Steps**(s_n)
  - i.e. resolves nondeterminism based on execution history

- Adv (or Adv_M) denotes the set of all adversaries

Adversaries – Examples

- Consider the previous example MDP
  - note that s_1 is the only state for which $|\mathbf{Steps}(s)| > 1$
  - 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

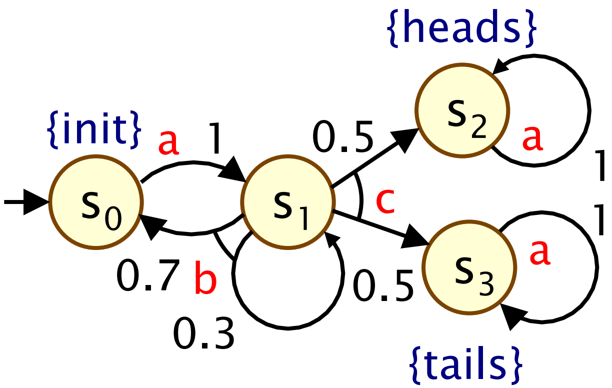


- Adversary σ_1
  - picks action c the first time
  - $\sigma_1(s_0 s_1)=(c,\mu_c)$
- Adversary σ_2
  - picks action b the first time, then c
  - $\sigma_2(s_0 s_1)=(b,\mu_b)$, $\sigma_2(s_0 s_1 s_1)=(c,\mu_c)$, $\sigma_2(s_0 s_1 s_0 s_1)=(c,\mu_c)$

(Note: actions/distributions omitted from paths for clarity)

Adversaries and paths

- $Path^{\sigma}(s) \subseteq Path(s)$
  - (infinite) paths from s where nondeterminism resolved by σ
  - i.e. paths $s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$
  - for which $\sigma(s_0(a_0,\mu_0)s_1\ldots s_n)) = (a_n,\mu_n)$



- Adversary σ_1
  - (picks action c the first time)
  - $Path^{\sigma_1}(s_0) = \{ s_0s_1s_2^{\omega}, s_0s_1s_3^{\omega} \}$
- Adversary σ_2
  - (picks action b the first time, then c)
  - $Path^{\sigma_2}(s_0) = \{ s_0s_1s_0s_1s_2^{\omega}, s_0s_1s_0s_1s_3^{\omega}, s_0s_1s_1s_2^{\omega}, s_0s_1s_1s_3^{\omega} \}$

Induced DTMCs

- Adversary σ for MDP induces an infinite-state DTMC D^σ

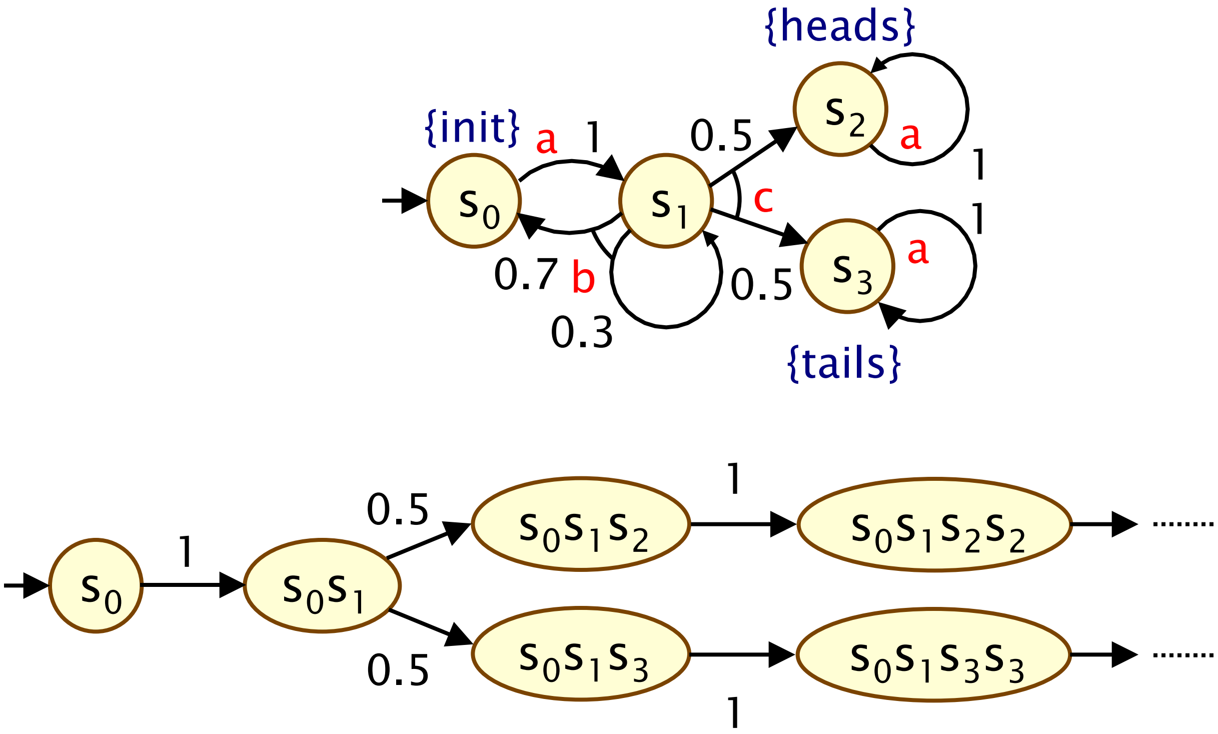
- $D^\sigma = (Path^\sigma_{fin}(s), s, \mathbf{P}^\sigma_s)$ where:
  - states of the DTMC are the finite paths of σ starting in state s
  - initial state is s (the path starting in s of length 0)
  - $\mathbf{P}^\sigma_s(\omega,\omega') = \mu(s')$ if $\omega' = \omega(a, \mu)s'$ and $\sigma(\omega) = (a,\mu)$
  - $\mathbf{P}^\sigma_s(\omega,\omega') = 0$ otherwise

- 1-to-1 correspondence between $Path^\sigma(s)$ and paths of D^σ

- This gives us a probability measure Pr^σ_s over $Path^\sigma(s)$
  - from probability measure over paths of D^σ

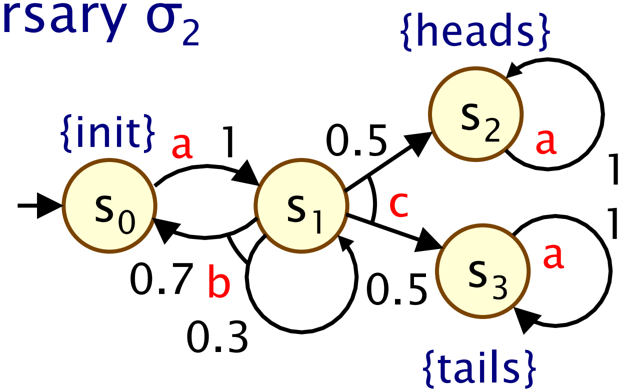
Adversaries – Examples

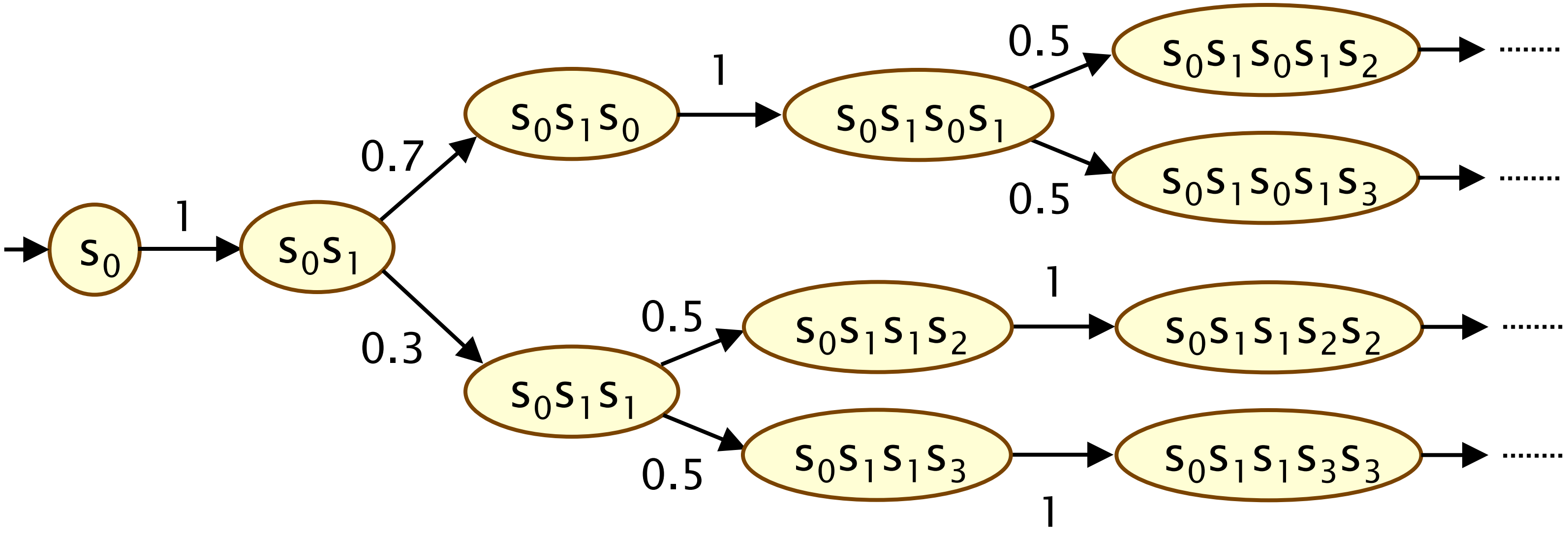
- Fragment of induced DTMC for adversary σ_1
  - σ_1 picks action c the first time



Adversaries – Examples

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





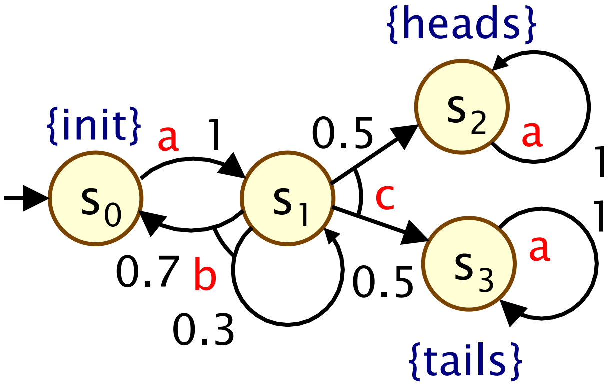
MDPs and probabilities

- $\mathrm{Prob}^{\sigma}(s, \psi) = \mathrm{Pr}^{\sigma}_{s} \{ \omega \in \mathrm{Path}^{\sigma}(s) \mid \omega \vDash \psi \}$
  - for some path formula ψ
  - e.g. $\mathrm{Prob}^{\sigma}(s, \mathrm{F}\ \mathrm{tails})$

- MDP provides best-/worst-case analysis
  - based on lower/upper bounds on probabilities
  - over all possible adversaries

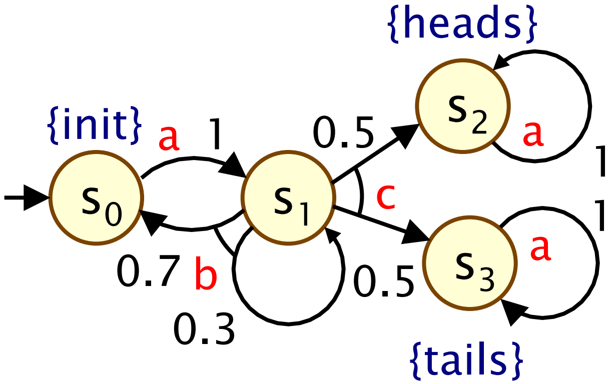
$$p_{\min}(s,\psi) = \inf_{\sigma \in Adv} \mathrm{Prob}^{\sigma}(s,\psi)$$

$$p_{\max}(s,\psi) = \sup_{\sigma \in Adv} \mathrm{Prob}^{\sigma}(s,\psi)$$

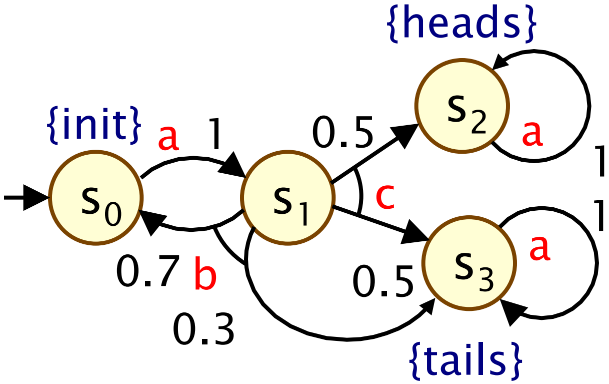


Examples

- $\text{Prob}^{\sigma1}(s_0, \text{F tails}) = 0.5$
- $\text{Prob}^{\sigma2}(s_0, \text{F tails}) = 0.5$
  - (where σ_i picks b i−1 times then c)
- …
- $p_{max}(s_0, \text{F tails}) = 0.5$
- $p_{min}(s_0, \text{F tails}) = 0$

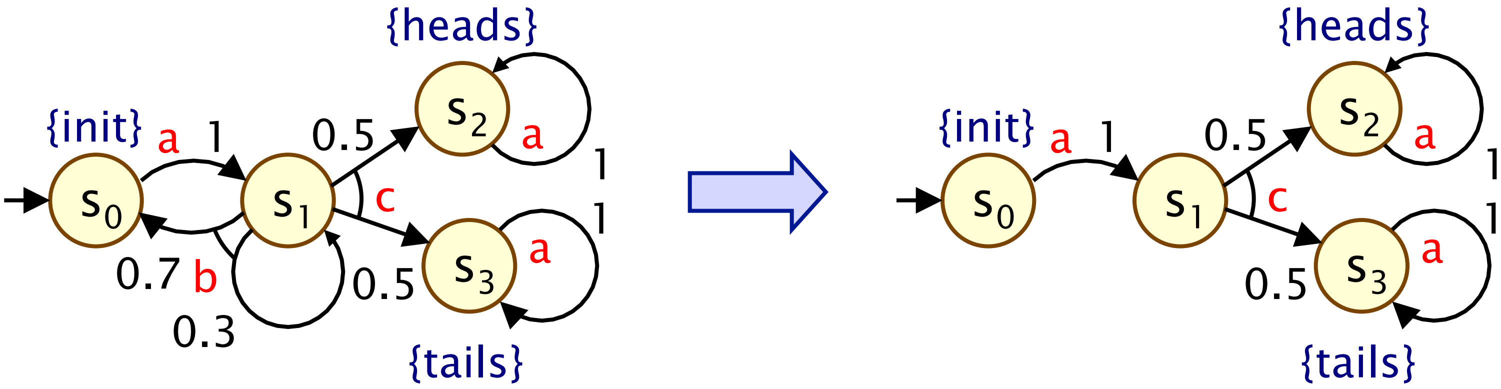


- $\text{Prob}^{\sigma1}(s_0, \text{F tails}) = 0.5$
- $\text{Prob}^{\sigma2}(s_0, \text{F tails})$
  $= 0.3+0.7\cdot0.5 = 0.65$
- $\text{Prob}^{\sigma3}(s_0, \text{F tails})$
  $= 0.3+0.7\cdot0.3+0.7\cdot0.7\cdot0.5 = 0.755$
- …
- $p_{max}(s_0, \text{F tails}) = 1$
- $p_{min}(s_0, \text{F tails}) = 0.5$



Memoryless adversaries

- Memoryless adversaries always pick same choice in a state
  - also known as: positional, Markov, simple
  - formally, $\sigma(s_0(a_0,\mu_0)s_1 \ldots s_n)$ depends only on s_n
  - can write as a mapping from states, i.e. $\sigma(s)$ for each $s \in S$
  - induced 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



Other classes of adversary

- Finite-memory adversary
  - finite number of modes, which can govern choices made
  - formally defined by a deterministic finite automaton
  - induced DTMC (for finite MDP) again mapped to finite DTMC

- Randomised adversary
  - maps finite paths $s_0(a_1,\mu_1)s_1 \ldots s_n$ in MDP to a probability distribution over element of **Steps**(s_n)
  - generalises deterministic schedulers
  - still induces a (possibly infinite state) DTMC

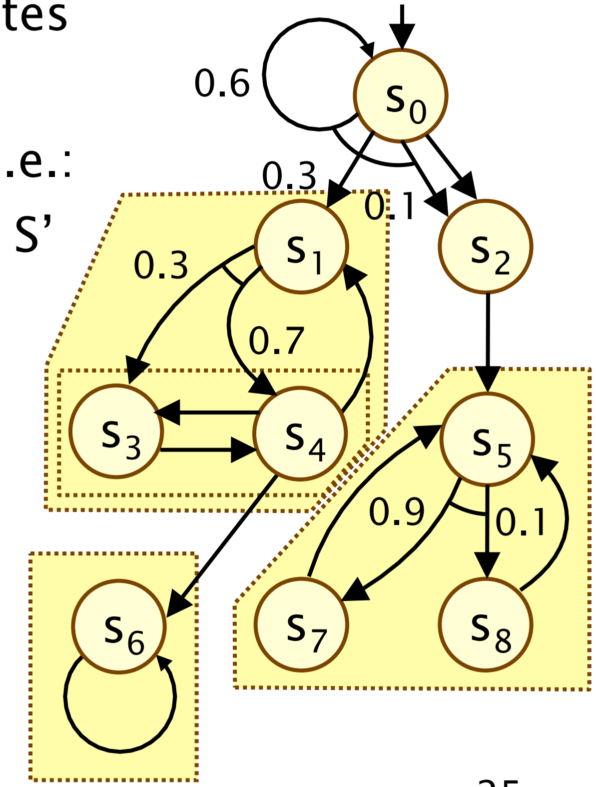
- Fair adversary
  - fairness assumptions on resolution of nondeterminism

End components

- Consider an MDP M = $(S, s_{init}, \mathbf{Steps}, L)$

- A sub-MDP of M is a pair (S',**Steps**') where:
  - S' ⊆ S is a (non-empty) subset of M's states
  - **Steps**'(s) ⊆ **Steps**(s) for each s ∈ S'
  - is closed under probabilistic branching, i.e.:
  - { s' | μ(s')>0 for some (a,μ)∈**Steps**'(s) } ⊆ S'

- An end component of M is a strongly connected sub-MDP

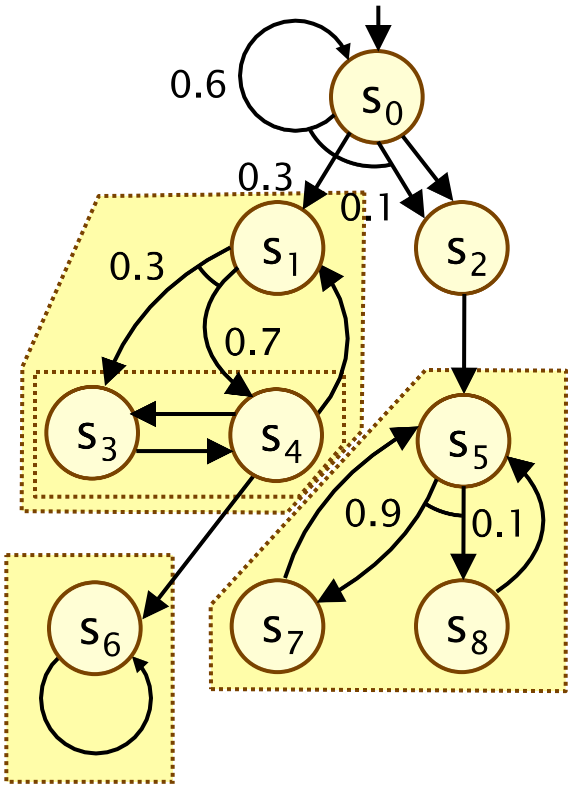


End components

- For finite MDPs…

- For every end component, there is an adversary which, with probability 1, forces the MDP to remain in the end component and visit all its states infinitely often

- Under every adversary σ, with probability 1 an end component will be reached and all of its states visited infinitely often
  - (analogue of fundamental property of finite DTMCs)



Summing up…

- Nondeterminism
  - concurrency, unknown environments/parameters, abstraction
- Markov decision processes (MDPs)
  - discrete-time + probability and nondeterminism
  - nondeterministic choice between multiple distributions
- Adversaries
  - resolution of nondeterminism only
  - induced set of paths and (infinite state DTMC)
  - induces DTMC yields probability measure for adversary
  - best-/worst-case analysis: minimum/maximum probabilities
  - memoryless adversaries
- End components
  - long-run behaviour: analogue of BSCCs for DTMCs

Probabilistic Model Checking Michaelmas Term 2011

Lecture 13
Reachability in MDPs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Recall – MDPs

- Markov decision process: $M = (S, s_{init}, \mathbf{Steps}, L)$
- Adversary $\sigma \in Adv$ resolves nondeterminism
- σ induces set of paths $Path^\sigma(s)$ and DTMC D^σ
- D^σ yields probability space Pr^σ_s over $Path^\sigma(s)$
- $Prob^\sigma(s, \psi) = Pr^\sigma_s \{ \omega \in Path^\sigma(s) \mid \omega \vDash \psi \}$
- MDP yields minimum/maximum probabilities:

$$p_{min}(s,\psi) = \inf_{\sigma \in Adv} Prob^\sigma(s,\psi)$$

$$p_{max}(s,\psi) = \sup_{\sigma \in Adv} Prob^\sigma(s,\psi)$$

Probabilistic reachability

- Minimum and maximum probability of reaching target set
  - target set = all states labelled with atomic proposition a

$$p_{min}(s, F\ a) = \inf_{\sigma \in Adv} Prob^{\sigma}(s, F\ a)$$

$$p_{max}(s, F\ a) = \sup_{\sigma \in Adv} Prob^{\sigma}(s, F\ a)$$

- Vectors: $\underline{p}_{min}(F\ a)$ and $\underline{p}_{max}(F\ a)$
  - minimum/maximum probabilities for all states of MDP

Overview

- Qualitative probabilistic reachability
  - case where $p_{min}>0$ or $p_{max}>0$

- Optimality equation

- Memoryless adversaries suffice
  - finitely many adversaries to consider

- Computing reachability probabilities
  - value iteration (fixed point computation)
  - linear programming problem
  - policy iteration

Qualitative probabilistic reachability

- Consider the problem of determining states for which $p_{min}(s, F\ a)$ or $p_{max}(s, F\ a)$ is zero (or non-zero)
  - max case: $S^{max=0} = \{ s \in S \mid p_{max}(s, F\ a) = 0 \}$
  - this is just (non-probabilistic) reachability

```
R := Sat(a)
done := false
while (done = false)
    R' = R ∪ { s ∈ S | ∃(a,μ)∈Steps(s) . ∃s'∈R . μ(s')>0}
    if (R'=R) then done := true
    R := R'
endwhile
return S\R
```

Qualitative probabilistic reachability

- Min case: $S^{min=0} = \{ s \in S \mid p_{min}(s, F\ a) = 0 \}$

note: quantification
over all choices

```
$R := Sat(a)$
$done := false$
while $(done = false)$
    $R' = R \cup \{ s \in S \mid \forall (a,\mu) \in Steps(s) .\ \exists s' \in R .\ \mu(s') > 0 \}$
    if $(R' = R)$ then $done := true$
    $R := R'$
endwhile
return $S \setminus R$
```

Optimality (min)

- The values $p_{min}(s, F\ a)$ are the unique solution of the following equations:

$$x_s = \begin{cases} 1 & \text{if } s \in \mathrm{Sat}(a) \\ 0 & \text{if } s \in S^{min=0} \\ \min\left\{ \sum_{s' \in S} \mu(s') \cdot x_{s'} \mid (a,\mu) \in \textbf{Steps}(s) \right\} & \text{otherwise} \end{cases}$$

optimal solution for state s uses optimal solution for successors s’

$S^{min=0}$

$=$

$\{ s \mid p_{min}(s, F\ a)=0 \}$

- This is an instance of the Bellman equation
  - (basis of dynamic programming techniques)

Optimality (max)

- Likewise, the values $p_{max}(s, F\ a)$ are the unique solution of the following equations:

$$x_s = \begin{cases} 1 & \text{if } s \in Sat(a) \\ 0 & \text{if } s \in S^{max=0} \\ \max \left\{ \sum_{s' \in S} \mu(s') \cdot x_{s'} \mid (a,\mu) \in \mathbf{Steps}(s) \right\} & \text{otherwise} \end{cases}$$

$S^{max=0}$

$=$

$\{ s \mid p_{max}(s, F\ a)=0 \}$

Memoryless adversaries

- Memoryless adversaries suffice for probabilistic reachability
  - i.e. there exist memoryless adversaries σ_{min} & σ_{max} such that:
  - $\mathrm{Prob}^{\sigma_{min}}(s, F\ a) = p_{min}(s, F\ a)$ for all states $s \in S$
  - $\mathrm{Prob}^{\sigma_{max}}(s, F\ a) = p_{max}(s, F\ a)$ for all states $s \in S$

- Construct adversaries from optimal solution:

$$\sigma_{min}(s) = \mathrm{argmin}\left\{ \sum_{s' \in S} \mu(s') \cdot p_{min}(s', F\ a) \mid (a, \mu) \in \mathbf{Steps}(s) \right\}$$

$$\sigma_{max}(s) = \mathrm{argmax}\left\{ \sum_{s' \in S} \mu(s') \cdot p_{max}(s', F\ a) \mid (a, \mu) \in \mathbf{Steps}(s) \right\}$$

Computing reachability probabilities

- Several approaches…

- 1. Value iteration
  - approximate with iterative solution method
  - corresponds to fixed point computation

Preferable in practice, e.g. in PRISM

- 2. Reduction to a linear programming (LP) problem
  - solve with linear optimisation techniques
  - exact solution using well-known methods

better complexity; good for small examples

- 3. Policy iteration
  - iteration over adversaries

Method 1 – Value iteration (min)

- For minimum probabilities $p_{min}(s, F\ a)$ it can be shown that:
  - $p_{min}(s, F\ a) = \lim_{n\rightarrow\infty} x_s^{(n)}$ where:

$$x_s^{(n)} = \begin{cases} 1 & \text{if } s \in Sat(a) \\ 0 & \text{if } s \in S^{min=0} \\ 0 & \text{if } s \in S^? \text{ and } n = 0 \\ \min\left\{ \sum_{s' \in S} \mu(s') \cdot x_{s'}^{(n-1)} \mid (a,\mu) \in \mathbf{Steps}(s) \right\} & \text{if } s \in S^? \text{ and } n > 0 \end{cases}$$

  - where: $S^? = S \setminus (\ Sat(a) \cup S^{min=0}\ )$

- Approximate iterative solution technique
  - iterations terminated when solution converges sufficiently

Method 1 – Value iteration (max)

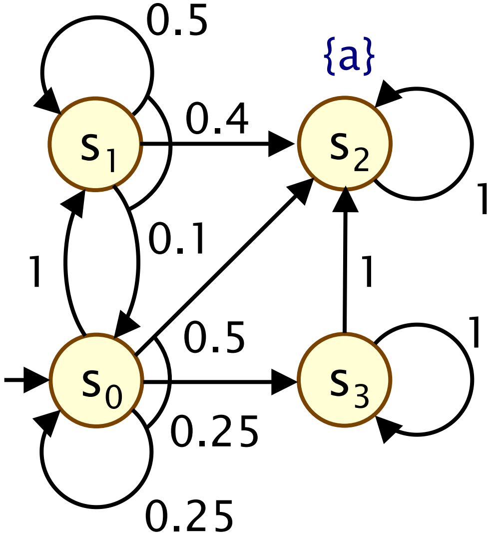
- Value iteration applies to maximum probabilities in the same way…
  - $p_{max}(s, F\ a) = \lim_{n\to\infty} x_s^{(n)}$ where:

$$x_s^{(n)} = \begin{cases} 1 & \text{if } s \in Sat(a) \\ 0 & \text{if } s \in S^{max=0} \\ 0 & \text{if } s \in S^{?} \text{ and } n = 0 \\ \max\left\{ \sum_{s' \in S} \mu(s') \cdot x_{s'}^{(n-1)} \mid (a, \mu) \in \mathbf{Steps}(s) \right\} & \text{if } s \in S^{?} \text{ and } n > 0 \end{cases}$$

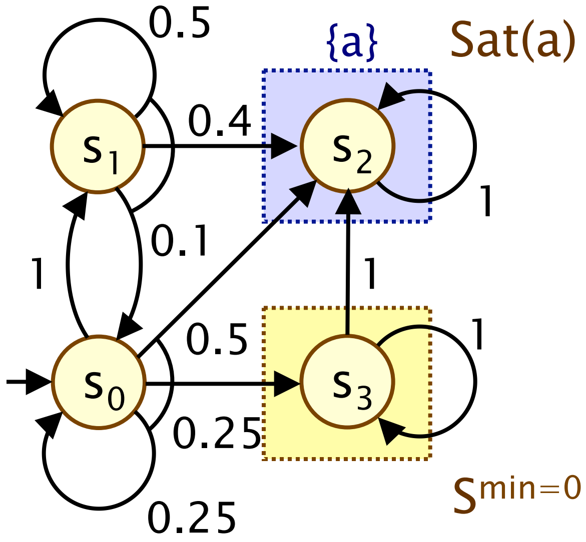
  - where: $S^{?} = S \setminus (Sat(a) \cup S^{max=0})$

Example

- Minimum/maximum probability of reaching an a-state



Example – Value iteration (min)



Compute: $p_{min}(s_i, F\ a)$

$Sat(a) = \{s_2\}$, $S^{min=0} = \{s_3\}$, $S^? = \{s_0, s_1\}$

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

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

n=1: $[\ \min(1\cdot 0,\ 0.25\cdot 0+0.25\cdot 0+0.5\cdot 1),$

$0.1\cdot 0+0.5\cdot 0+0.4\cdot 1,\ 1,\ 0\ ]$

$= [\ 0, 0.4, 1, 0\ ]$

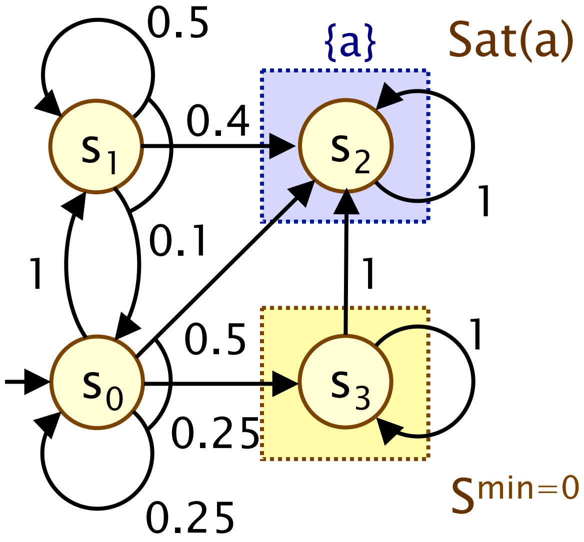
n=2: $[\ \min(1\cdot 0.4, 0.25\cdot 0+0.25\cdot 0+0.5\cdot 1),$

$0.1\cdot 0+0.5\cdot 0.4+0.4\cdot 1,\ 1,\ 0\ ]$

$=[\ 0.4, 0.6, 1, 0\ ]$

n=3: …

Example - Value iteration (min)



$\underline{p}_{min}(F\ a)$

=

[2/3, 14/15, 1, 0]

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

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]

…

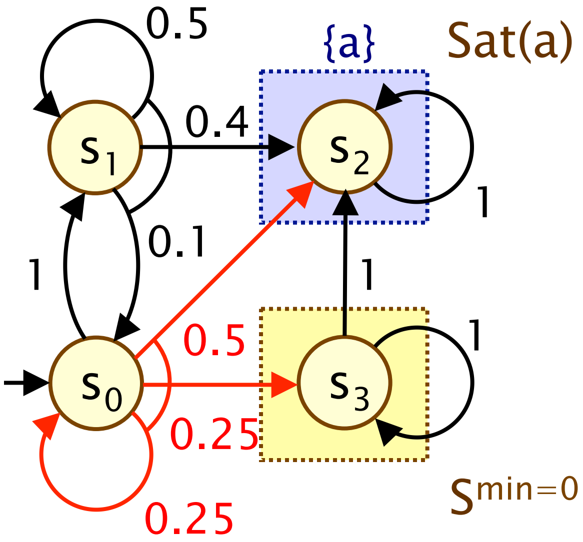
n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

≈ [2/3, 14/15, 1, 0]

Generating an optimal adversary

- Min adversary σ_{min}



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

…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

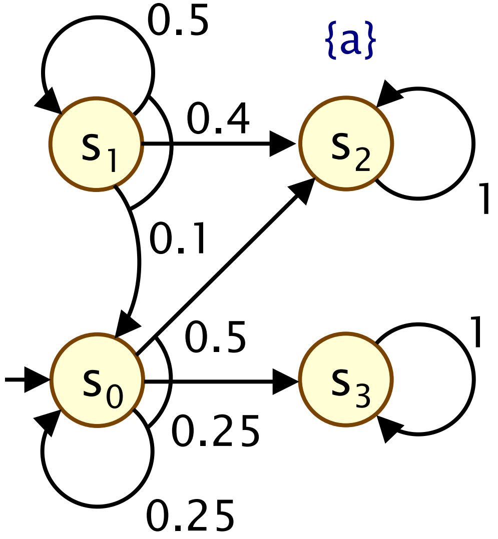
$\approx [\ 2/3,\ 14/15,\ 1,\ 0\ ]$

s_0 : $\min(1\cdot14/15,\ 0.5\cdot1+0.25\cdot0+0.25\cdot2/3)$

$=\min(14/15,\ 2/3)$

Generating an optimal adversary

- DTMC $D^{\sigma_{min}}$



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

…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

$\approx$ [2/3, 14/15, 1, 0]

s_0 : min(1·14/15, 0.5·1+0.25·0+0.25·2/3)

=min(14/15, 2/3)

Value iteration as a fixed point

- Can view value iteration as a fixed point computation over vectors of probabilities $\underline{y} \in [0,1]^S$, e.g. for minimum:

$$F(\underline{y})(s) = \begin{cases} 1 & \text{if } s \in Sat(a) \\ 0 & \text{if } s \in S^{\min=0} \\ \min\left\{ \sum_{s' \in S} \mu(s') \cdot \underline{y}(s') \mid (a,\mu) \in \mathbf{Steps}(s) \right\} & \text{otherwise} \end{cases}$$

- Let:
  - $\underline{x}^{(0)} = \underline{0}$ (i.e. $\underline{x}^{(0)}(s) = 0$ for all s)
  - $\underline{x}^{(n+1)} = F(\underline{x}^{(n)})$
- Then:
  - $\underline{x}^{(0)} \leq \underline{x}^{(1)} \leq \underline{x}^{(2)} \leq \underline{x}^{(3)} \leq \ldots$
  - $\underline{p}_{\min}(F\ a) = \lim_{n\to\infty} \underline{x}^{(n)}$

Linear programming

- Linear programming
  - optimisation of a linear objective function
  - subject to linear (in)equality constraints

- General form:
  - n variables: $x_1, x_2, \dots, x_n$
  - maximise (or minimise):
    - $c_1x_1+c_2x_2+\dots+c_nx_n$
  - subject to constraints
    - $a_{11}x_1+a_{12}x_2+\dots a_{1n}x_n \leq b_1$
    - $a_{21}x_1+a_{22}x_2+\dots a_{2n}x_n \leq b_2$
    - …
    - $a_{m1}x_1+a_{m2}x_2+\dots a_{mn}x_n \leq b_m$

Many standard solution techniques exist, e.g. Simplex, ellipsoid method, interior point method

In matrix/vector form:
Maximise (or minimise) $\underline{c} \cdot \underline{x}$ subject to $\mathbf{A} \cdot \underline{x} \leq \underline{b}$

Method 2 – Linear programming problem

- Min probabilities $p_{min}(s, F\ a)$ can be computed as follows:
  - $p_{min}(s, F\ a) = 1$ if $s \in Sat(a)$
  - $p_{min}(s, F\ a) = 0$ if $s \in S^{min=0}$
  - values for remaining states in the set $S^? = S \setminus (Sat(a) \cup S^{no})$ can be obtained as the unique solution of the following linear programming problem:

$$\boxed{\begin{gathered}\text{maximize } \sum_{s\ \in S^?} x_s \text{ subject to the constraints:}\\ x_s \le \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in Sat(a)} \mu(s')\\ \text{for all } s \in S^? \text{ and for all } (a,\mu) \in \textbf{Steps}(s)\end{gathered}}$$

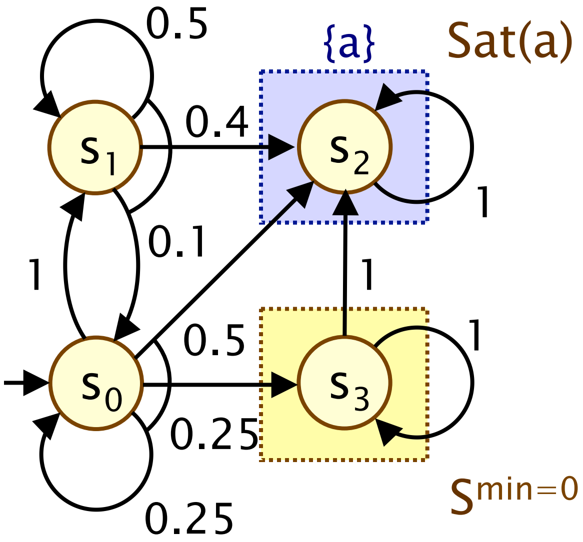
Linear programming problem (max)

- Max probabilities $p_{max}(s, F\ a)$ can be computed as follows:
  - $p_{max}(s, F\ a) = 1$ if $s \in Sat(a)$
  - $p_{max}(s, F\ a) = 0$ if $s \in S^{max=0}$
  - values for remaining states in the set $S^? = S \setminus (Sat(a) \cup S^{no})$ can be obtained as the unique solution of the following linear programming problem:

$$
\boxed{
\begin{array}{l}
\text{minimize } \sum_{s \in S^?} x_s \text{ subject to the constraints:} \\
\qquad x_s \geq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in Sat(a)} \mu(s') \\
\text{for all } s \in S^? \text{ and for all } (a, \mu) \in \mathbf{Steps}(s)
\end{array}
}
$$

Differences from min case

Example – Linear programming (min)



Let $x_i = p_{min}(s_i, F\ a)$

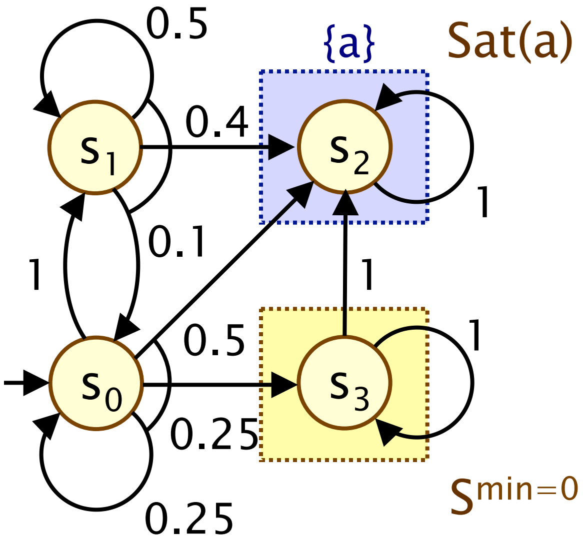
Sat(a): $x_2=1$, $S^{min=0}$: $x_3=0$

For $S^? = \{s_0, s_1\}$:

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 – Linear programming (min)



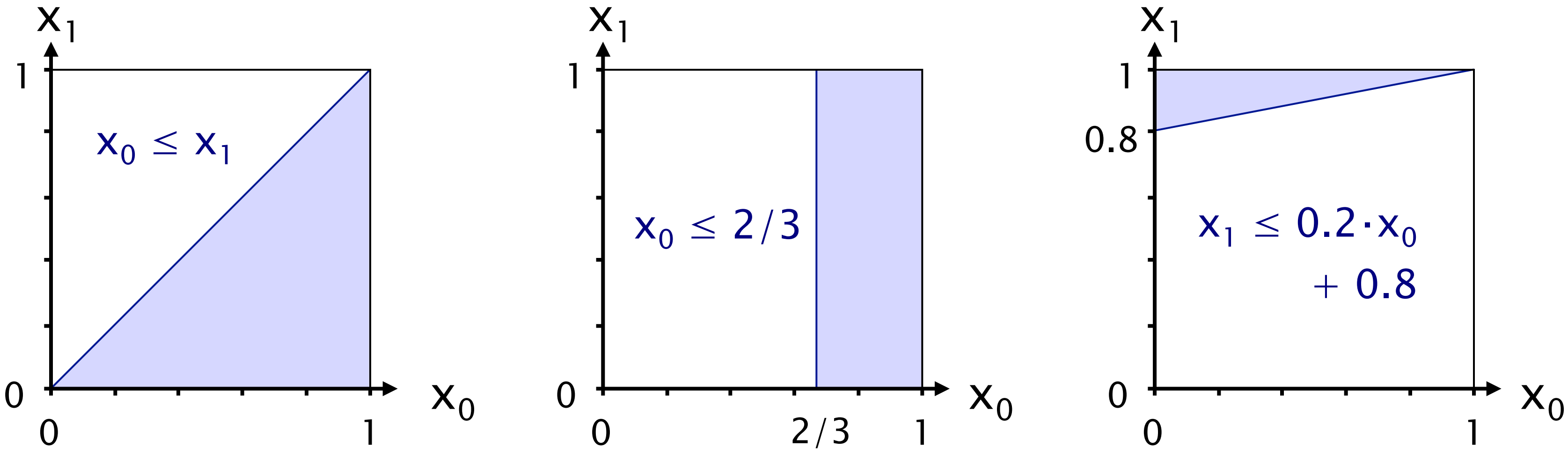
Let $x_i = p_{min}(s_i, F\ a)$

Sat(a): $x_2=1$, $S^{min=0}$: $x_3=0$

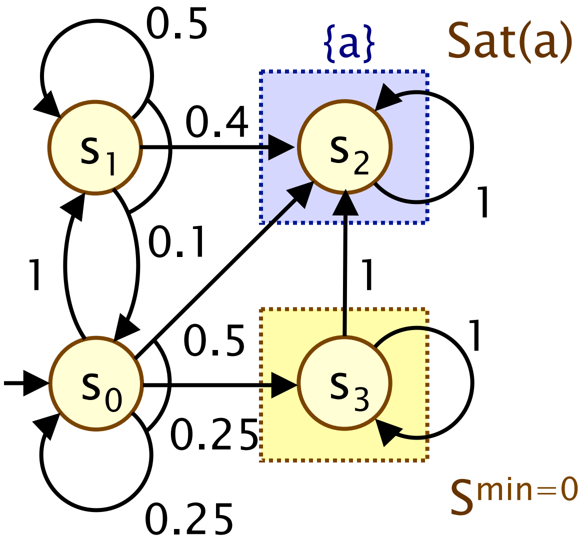
For $S^? = \{s_0, s_1\}$:

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 – Linear programming (min)



Let $x_i = p_{min}(s_i, F\ a)$

Sat(a): $x_2=1$, $S^{min=0}$: $x_3=0$

For $S^? = \{s_0, s_1\}$:

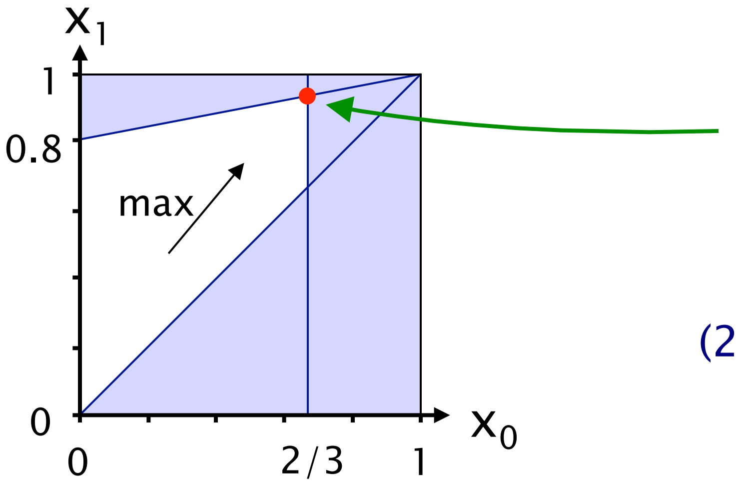
Maximise x_0+x_1 subject to constraints:

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

$\underline{p}_{min}(F\ a)$

$=$

$[\ 2/3, 14/15, 1, 0\ ]$



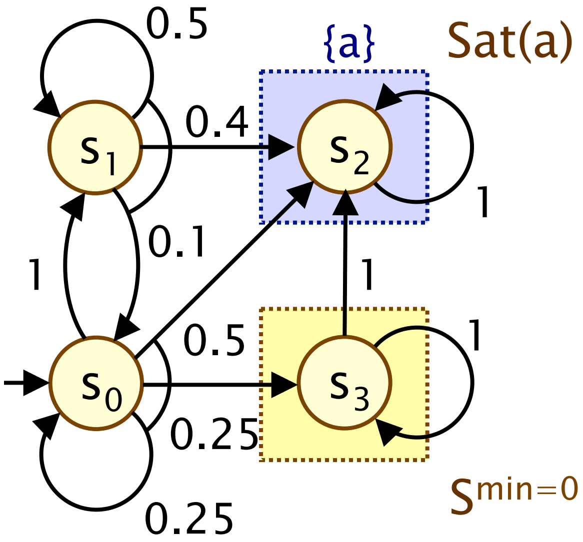
Solution:

(x_0, x_1)

$=$

$(2/3, 14/15)$

Example – Linear programming (min)



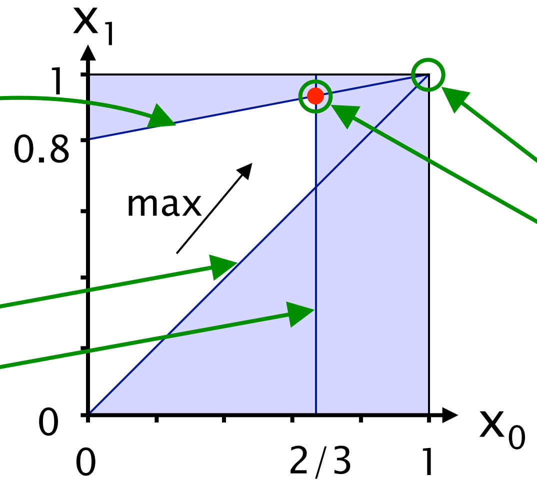
Let $x_i = p_{min}(s_i, F\ a)$

Sat(a): $x_2=1$, $S^{min=0}$: $x_3=0$

For $S^? = \{s_0, s_1\}$:

Maximise x_0+x_1 subject to constraints:

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



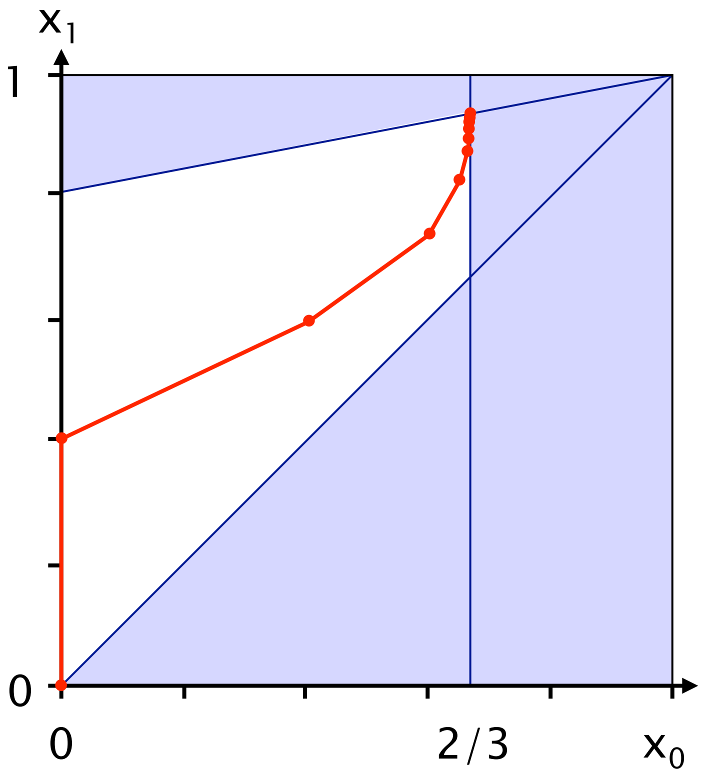
$x_1 \leq 0.2 \cdot x_0 + 0.8$

$x_0 \leq x_1$

$x_0 \leq 2/3$

Two memoryless adversaries

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]

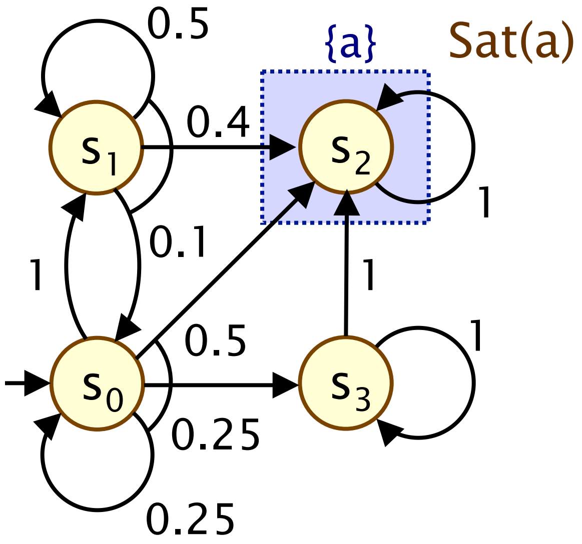
…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

$\approx$ [2/3, 14/15, 1, 0]

Example – Linear programming (max)



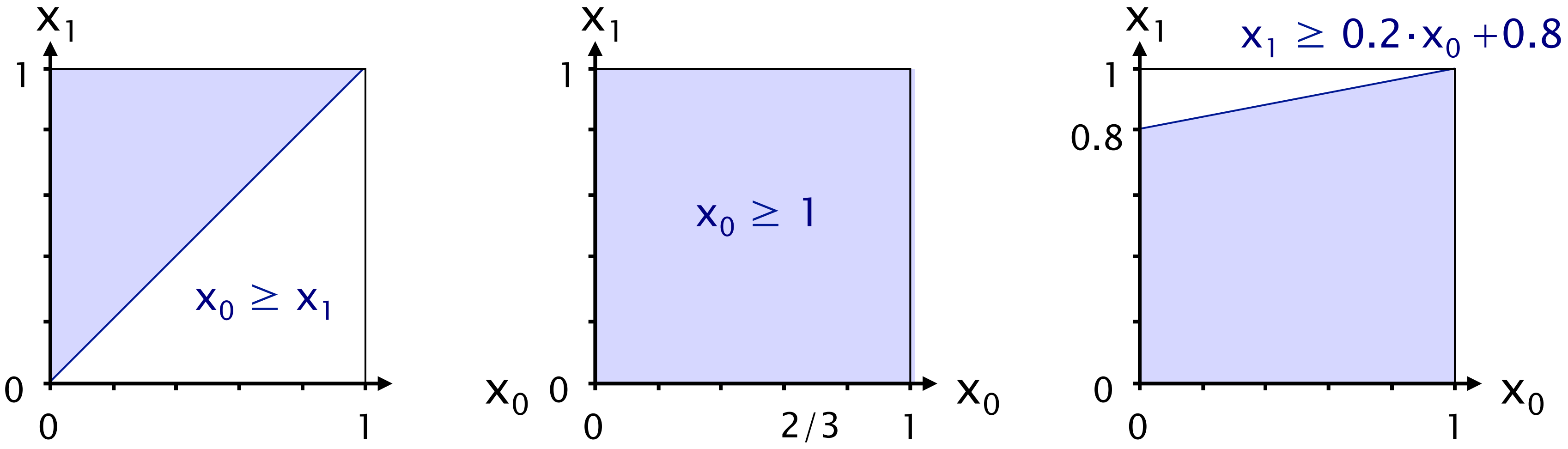
Let $x_i = p_{max}(s_i, F\ a)$

Sat(a): $x_2=1$, $S^{max=0} = \varnothing$

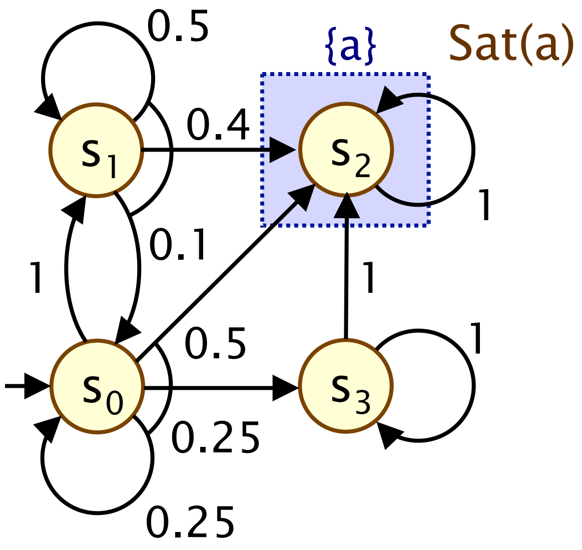
For $S^? = \{s_0, s_1, s_3\}$:

Minimise $x_0+x_1+x_3$ subject to constraints:

- $x_0 \geq x_1$
- $x_0 \geq 2/3 + 1/3x_3$
- $x_1 \geq 0.2 \cdot x_0 + 0.8$
- $x_3 \geq x_2$
- $x_3 \geq x_3$



Example – Linear programming (max)



Let $x_i = p_{max}(s_i, F\ a)$

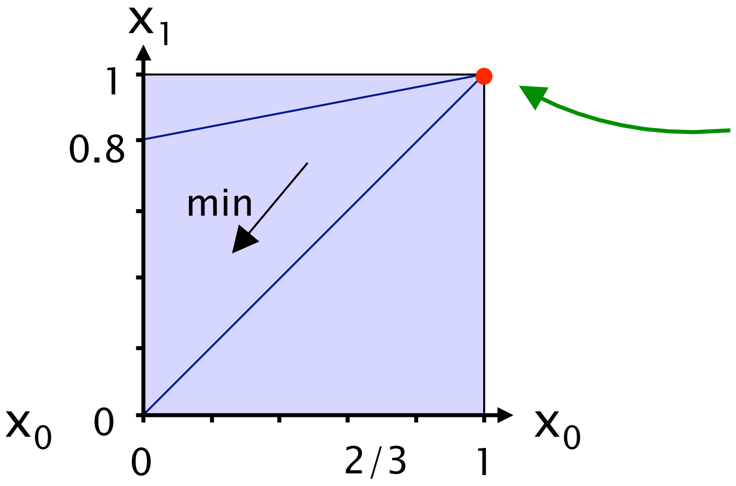
Sat(a): $x_2=1$, $S^{max=0} = \varnothing$

For $S^? = \{s_0, s_1, s_3\}$:

Minimise $x_0+x_1+x_3$ subject to constraints:

- $x_0 \geq x_1$
- $x_0 \geq 2/3 + 1/3x_3$
- $x_1 \geq 0.2\cdot x_0 + 0.8$

- $x_3 \geq x_2$
- $x_3 \geq x_3$



(only feasible) solution:

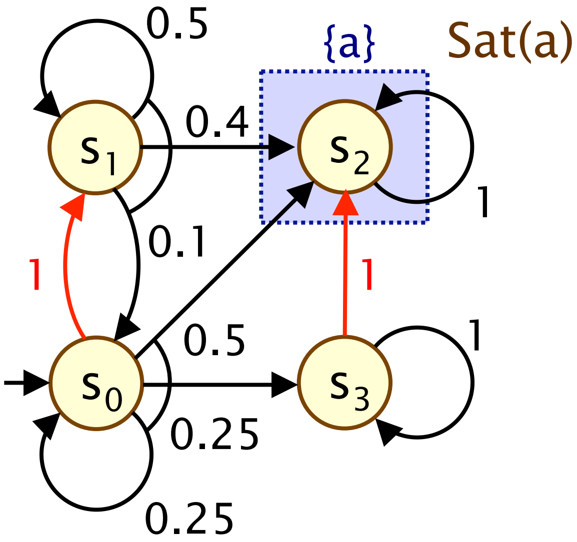
(x_0, x_1, x_2)

$=$

$(1, 1, 1)$

Generating an adversary

- Max adversary σ_{max}



Let $x_i = p_{max}(s_i, F\ a)$

Sat(a): $x_2=1$, $S^{max=0} = \varnothing$

For $S^? = \{s_0, s_1, s_3\}$:

Minimise $x_0+x_1+x_3$ subject to constraints:

- $x_0 \geq x_1$
- $x_0 \geq 2/3 + 1/3x_3$
- $x_1 \geq 0.2 \cdot x_0 + 0.8$
- $x_3 \geq x_2$
- $x_3 \geq x_3$

Solution:

- $(x_0, x_1, x_3) = (1, 1, 1)$

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 $\underline{\text{Prob}}^{\sigma}$(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 min/max probabilities) each time

Method 3 – Policy iteration

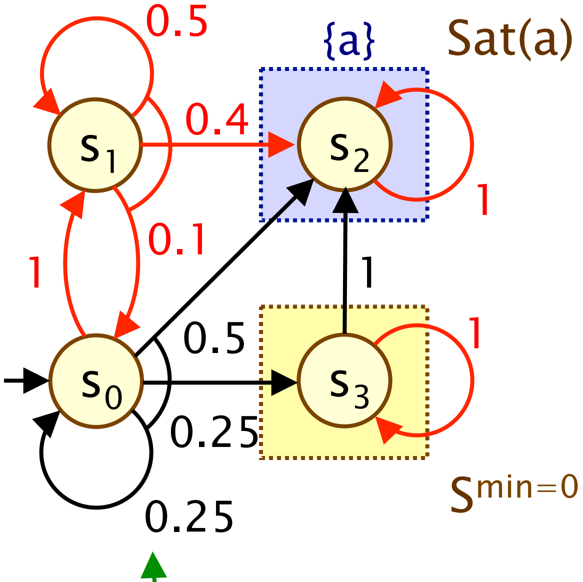
- 1. Start with an arbitrary (memoryless) adversary σ
  - pick an element of **Steps**(s) for each state s ∈ S
- 2. Compute the reachability probabilities Prob$^\sigma$(F a) for σ
  - probabilistic reachability on a DTMC
  - i.e. solve linear equation system
- 3. Improve the adversary in each state

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

$$\sigma'(s) = \text{argmax}\left\{ \sum_{s' \in S} \mu(s') \cdot \text{Prob}^{\sigma}(s', \text{F}\, a) \mid (a, \mu) \in \textbf{Steps}(s) \right\}$$

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

Example – Policy iteration (min)



Arbitrary adversary σ:

Compute: Prob$^\sigma$(F a)

Let x_i = Prob$^\sigma$(s_i, 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:

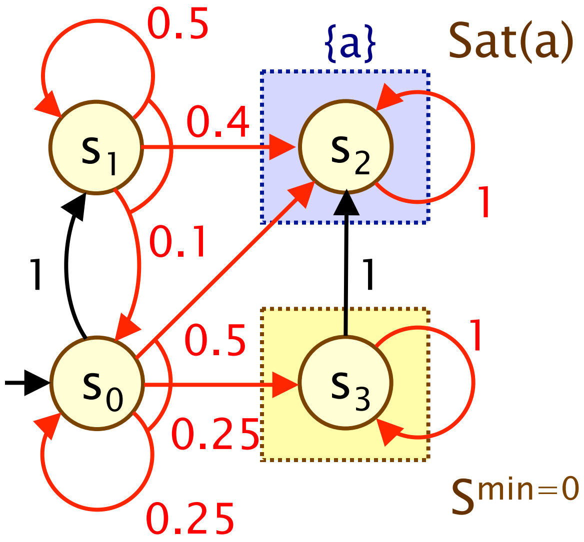
Prob$^\sigma$(F a) = [1, 1, 1, 0]

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 (min)



Refined adversary σ' :

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

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

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

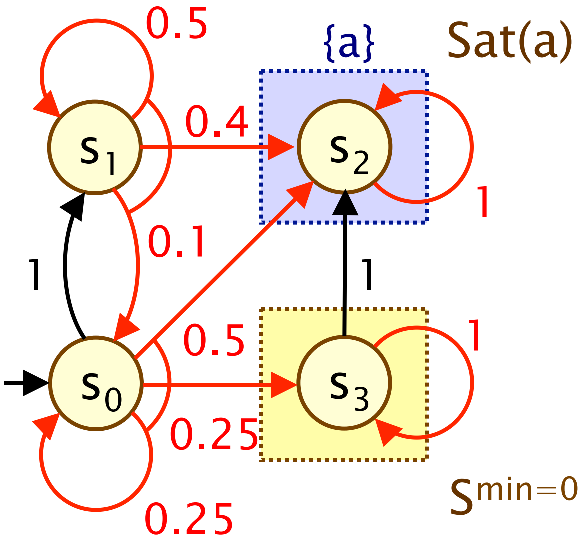
- $x_0 = 0.25 \cdot x_0 + 0.5$
- $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

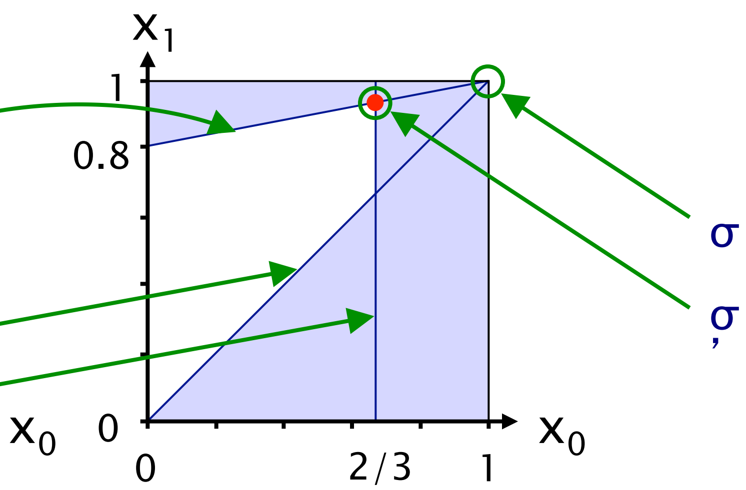
Solution:

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

This is optimal

Example - Policy iteration (min)





$x_1 = 0.2 \cdot x_0 + 0.8$

$x_0 = x_1$

$x_0 = 2/3$

Summing up…

- Probabilistic reachability in MDPs
- Qualitative case: min/max probability > 0
  - simple graph-based computation
  - need to do this first, before other computation methods
- Memoryless adversaries suffice
  - reduction to finite number of adversaries
- Computing reachability probabilities… (and generation of optimal adversary)
- 1. Value iteration
  - approximate; iterative; fixed point computation
- 2. Reduce to linear programming problem
  - good for small examples; doesn't scale well
- 3. Policy iteration

Lecture 14
Model Checking for MDPs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- PCTL for MDPs
  - syntax, semantics, examples
- PCTL model checking
  - next, bounded until, until
  - precomputation algorithms
  - value iteration, linear optimisation
  - examples
- Costs and rewards

PCTL

- Temporal logic for describing properties of MDPs
  - identical syntax to the logic PCTL for DTMCs

ψ is true with probability ~p

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

- $\psi \ ::= \ X\,\phi \quad \mid \quad \phi\ U^{\leq k}\ \phi \quad \mid \quad \phi\ U\ \phi$ (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, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

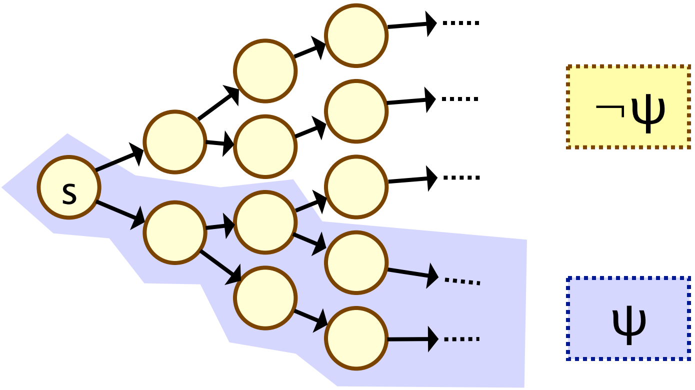
PCTL semantics for MDPs

- 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 and of path formulas are identical to those for DTMCs:
- For a state s of the MDP $(S, s_{init}, \mathbf{Steps}, L)$:
  - $s \vDash a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \vDash \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \vDash \phi_1$ and $s \vDash \phi_2$
  - $s \vDash \neg\phi \quad \Leftrightarrow \quad s \vDash \phi$ is false
- For a path $\omega = s_0(a_1,\mu_1)s_1(a_2,\mu_2)s_2\ldots$ 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 $s \vDash P_{\sim p} [\ \psi\ ] \Leftrightarrow Prob^{\sigma}(s, \psi) \sim p$ for all adversaries σ
  - where $Prob^{\sigma}(s, \psi) = Pr^{\sigma}_{s} \{ \omega \in Path^{\sigma}(s) \mid \omega \vDash \psi \}$



$Prob^{\sigma}(s, \psi) \sim p$

Minimum and maximum probabilities

- Letting:
  - $p_{max}(s, \psi) = \sup_{\sigma \in Adv} Prob^{\sigma}(s, \psi)$
  - $p_{min}(s, \psi) = \inf_{\sigma \in Adv} Prob^{\sigma}(s, \psi)$

- We have:
  - if $\sim \in \{\geq,>\}$, then $s \vDash P_{\sim p} [\psi] \;\Leftrightarrow\; p_{min}(s, \psi) \sim p$
  - if $\sim \in \{<,\leq\}$, then $s \vDash P_{\sim p} [\psi] \;\Leftrightarrow\; p_{max}(s, \psi) \sim p$

- 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

Classes of adversary

- A more general semantics for PCTL over MDPs
  - parameterise by a class of adversaries Adv*

- Only change is:
  - $s \models_{Adv^*} P_{\sim p} [\psi] \;\Leftrightarrow\; Prob^{\sigma}(s, \psi) \sim p$ for all adversaries $\sigma \in Adv^*$

- Original semantics obtained by taking Adv* = Adv

- Alternatively, take Adv* to be the set of all fair adversaries
  - path fairness: if a state occurs on a path infinitely often, then each non-deterministic choice occurs infinitely often
  - see e.g. [BK98]

PCTL derived operators

- Many of the same equivalences as for DTMCs, e.g.:
  - $F\ \phi \equiv \text{true}\ U\ \phi$ (eventually)
  - $F^{\leq k}\ \phi \equiv \text{true}\ U^{\leq k}\ \phi$
  - $G\ \phi \equiv \neg(F\ \neg\phi) \equiv \neg(\text{true}\ U\ \neg\phi)$ (always)
  - $G^{\leq k}\ \phi \equiv \neg(F^{\leq k}\ \neg\phi)$
  - etc.

- But… for example:
  - $P_{\geq p}\ [\ \psi\ ] \not\equiv \neg P_{<p}\ [\ \psi\ ]$ (negation + probability)

- Duality between min/max:
  - for any path formula ψ: $p_{min}(s, \psi) = 1 - p_{max}(s, \neg\psi)$
  - so, for example: $P_{\geq p}\ [\ G\ \phi\ ] \equiv P_{\leq 1-p}\ [\ F\ \neg\phi\ ]$

Qualitative properties

- PCTL can express qualitative properties of MDPs
  - like for DTMCs, can relate these to CTL's AF and EF operators
  - need to be careful with "there exists" and adversaries

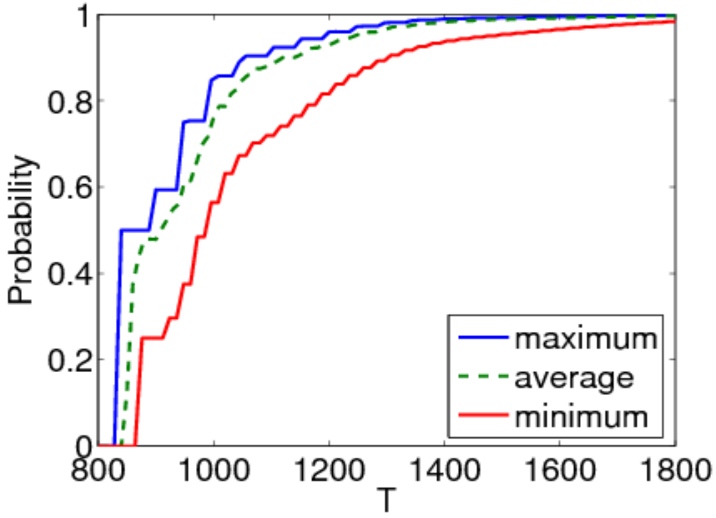
- $P_{\geq 1}$ [F ϕ] is (similar to but) weaker than AF ϕ
  - $P_{\geq 1}$ [F ϕ] $\Leftrightarrow$ $Prob^{\sigma}(s, F\ \phi) \geq 1$ for all adversaries σ
  - recall that "probability$\geq$1" is weaker than "for all"

- We can construct an equivalence for EF ϕ
  - EF $\phi \not\equiv P_{>0}$[F ϕ]
  - but:
  - EF $\phi \equiv \neg P_{\leq 0}$[F ϕ]

Quantitative properties

- For PCTL properties with P as the outermost operator
  - PRISM allows a quantitative form
  - for MDPs, there are 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?"
  - model checking is no harder since compute the values of $p_{min}(s, ψ)$ or $p_{max}(s, ψ)$ anyway
  - useful to spot patterns/trends

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



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?"

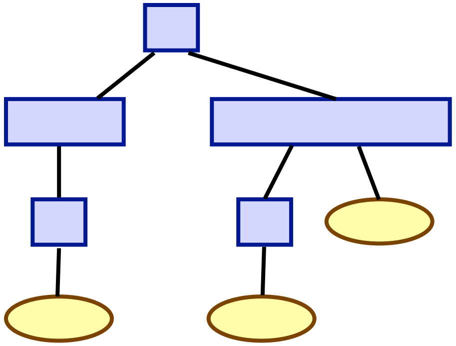
- CSMA/CD communication protocol
  - $P_{max=?}$ [F collisions=k]
  - "what is the maximum probability of k collisions?"

- Self-stabilisation protocols
  - $P_{min=?}$ [$F^{\leq t}$ stable]
  - "what is the minimum probability of reaching a stable state within k steps?"

PCTL model checking for MDPs

- Algorithm for PCTL model checking [BdA95]
  - inputs: MDP $M=(S,s_{init},\mathbf{Steps},L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = set of states satisfying ϕ
- Often, also consider quantitative results
  - e.g. compute result of $P_{min=?}$ [$F^{\leq t}$ stable] for $0 \leq t \leq 100$

- Basic algorithm same as PCTL for DTMCs
  - proceeds by induction on parse tree of ϕ
- For the non-probabilistic operators:
  - $Sat(true) = S$
  - $Sat(a) = \{ s \in S \mid a \in L(s) \}$
  - $Sat(\neg\phi) = S \setminus Sat(\phi)$
  - $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2)$



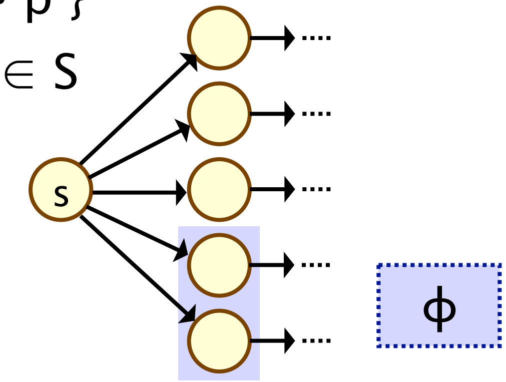
PCTL model checking for MDPs

- Main task: model checking $P_{\sim p}$ [ψ] formulae
  - reduces to computation of min/max probabilities
  - i.e. $p_{min}(s, \psi)$ or $p_{max}(s, \psi)$ for all $s \in S$
  - dependent on whether $\sim \in \{\geq, >\}$ or $\sim \in \{<, \leq\}$

- Three cases:
  - next ($X\ \phi$)
  - bounded until ($\phi_1\ U^{\leq k}\ \phi_2$)
  - unbounded until ($\phi_1\ U\ \phi_2$)

PCTL next for MDPs

- Computation of probabilities for PCTL next operator
- Consider case of minimum probabilities…
  - $\text{Sat}(P_{\sim p}[\ X\ \phi\ ]) = \{\ s \in S \mid p_{\min}(s,\ X\ \phi) \sim p\ \}$
  - need to compute $p_{\min}(s,\ X\ \phi)$ for all $s \in S$
- Recall in the DTMC case
  - sum outgoing probabilities for transitions to ϕ-states
  - $\text{Prob}(s,\ X\ \phi) = \Sigma_{s' \in \text{Sat}(\phi)}\ \mathbf{P}(s,s')$



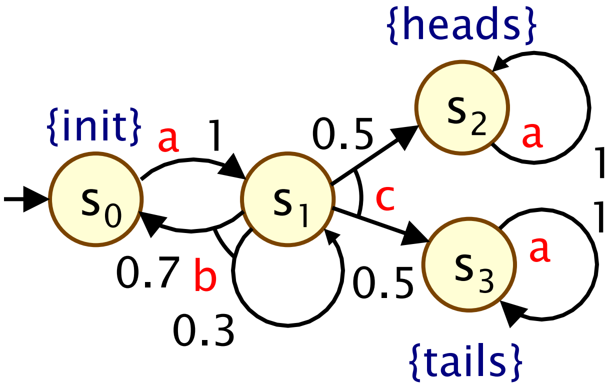
- For MDPs, perform computation for each distribution available in s and then take minimum:
  - $p_{\min}(s,\ X\ \phi) = \min\{\ \Sigma_{s' \in \text{Sat}(\phi)}\ \mu(s') \mid (a,\mu) \in \text{Steps}(s)\ \}$

- Maximum probabilities case is analogous

PCTL next – Example

- Model check: $P_{\geq 0.5}$ [X heads]
  - lower probability bound so minimum probabilities required
  - Sat (heads)= $\{s_2\}$
  - e.g. $p_{min}(s_1, \text{X heads}) = \min(0, 0.5) = 0$
  - can do all at once with matrix-vector multiplication:

$$\text{Steps} \cdot \underline{\text{heads}} = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \left[\begin{array}{c} 0 \\ \hline 0 \\ 0.5 \\ \hline 1 \\ \hline 0 \end{array}\right]$$



- Extracting the minimum for each state yields
  - $\underline{p}_{min}(\text{X heads}) = [0, 0, 1, 0]$
  - $Sat(P_{\geq 0.5} [\text{ X heads }]) = \{s_2\}$

PCTL bounded until for MDPs

- Computation of probabilities for PCTL $U^{\leq k}$ operator
- Consider case of minimum probabilities…
  - $Sat(P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]) = \{\ s \in S \mid p_{min}(s, \phi_1\ U^{\leq k}\ \phi_2) \sim p\ \}$
  - need to compute $p_{min}(s, \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$
- First identify (some) states where probability is 1 or 0
  - $S^{yes} = Sat(\phi_2)$ and $S^{no} = S \setminus (Sat(\phi_1) \cup Sat(\phi_2))$
- Then solve the recursive equations:

$$p_{min}(s, \phi_1\, U^{\leq k}\, \phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ 0 & \text{if } s \in S^{?} \text{ and } k = 0 \\ \min\left\{ \sum_{s' \in S} \mu(s') \cdot p_{min}(s', \phi_1\, U^{\leq k-1}\, \phi_2) \mid (a,\mu) \in Steps(s) \right\} & \text{if } s \in S^{?} \text{ and } k > 0 \end{cases}$$

- Maximum probabilities case is analogous

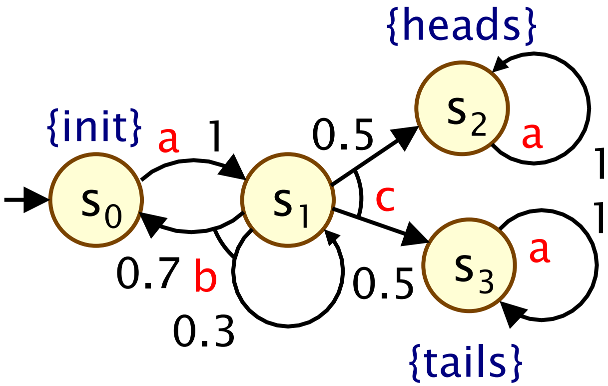
PCTL bounded until for MDPs

- Simultaneous computation of vector $\underline{p}_{min}(\phi_1\ U^{\leq k}\ \phi_2)$
  - i.e. probabilities $p_{min}(s, \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$

- Recursive definition in terms of matrices and vectors
  - similar to DTMC case
  - requires k matrix-vector multiplications
  - in addition requires k minimum operations

PCTL bounded until – Example

- Model check: $P_{<0.95}$ [$F^{\leq 3}$ init] ≡ $P_{<0.95}$ [true $U^{\leq 3}$ init]
  - upper probability bound so maximum probabilities required
  - Sat (true) = S and Sat (init) = $\{s_0\}$
  - $S^{yes} = \{s_0\}$ and $S^{no} = \varnothing$
  - $S^? = \{s_1, s_2, s_3\}$
- The vector of probabilities is computed successively as:
  - $\underline{p}_{max}(\text{true } U^{\leq 0} \text{ init }) = [\ 1, 0, 0, 0\ ]$
  - $\underline{p}_{max}(\text{true } U^{\leq 1} \text{ init }) = [\ 1, 0.7, 0, 0\ ]$
  - $\underline{p}_{max}(\text{true } U^{\leq 2} \text{ init }) = [\ 1, 0.91, 0, 0\ ]$
  - $\underline{p}_{max}(\text{true } U^{\leq 3} \text{ init }) = [\ 1, 0.973, 0, 0\ ]$
- Hence, the result is:
  - Sat($P_{<0.95}$ [$F^{\leq 3}$ init]) = $\{ s_2, s_3 \}$



PCTL until for MDPs

- Computation of probabilities for all $s \in S$:
  - $p_{min}(s, \phi_1 \text{ U } \phi_2)$ or $p_{max}(s, \phi_1 \text{ U } \phi_2)$
- Essentially the same as computation of reachability probabilities (see previous lecture)
  - just need to consider additional ϕ_1 constraint
- Overview:
  - precomputation:
    - identify states where the probability is 0 (or 1)
  - several options to compute remaining values:
    - value iteration
    - reduction to linear programming

PCTL until for MDPs – Precomputation

- Determine all states for which probability is 0
  - min case: $S^{no} = \{ s \in S \mid p_{min}(s, \phi_1 \text{ U } \phi_2) = 0 \}$ – Prob0E
  - max case: $S^{no} = \{ s \in S \mid p_{max}(s, \phi_1 \text{ U } \phi_2) = 0 \}$ – Prob0A
- Determine all states for which probability is 1
  - min case: $S^{yes} = \{ s \in S \mid p_{min}(s, \phi_1 \text{ U } \phi_2) = 1 \}$ – ~~Prob1A~~
  - max case: $S^{yes} = \{ s \in S \mid p_{max}(s, \phi_1 \text{ U } \phi_2) = 1 \}$ – Prob1E

not covered here

- Like for DTMCs:
  - identifying 0 states required (for uniqueness of LP problem)
  - identifying 1 states is optional (but useful optimisation)
- Advantages of precomputation
  - reduces size of numerical computation problem
  - gives exact results for the states in S^{yes} and S^{no} (no round-off)
  - suffices for model checking of qualitative properties

PCTL until for MDPs – Prob0E

- Minimum probabilities 0
  - S^{no} = { $s \in S$ | $p_{min}(s, \phi_1 \text{ U } \phi_2) = 0$ } = $Sat(\neg P_{>0} [\ \phi_1 \text{ U } \phi_2\ ])$

```
PROB0E(Sat(ϕ1), Sat(ϕ2))
R := Sat(ϕ2)
done := false
while (done = false)
        R′ := R ∪ {s ∈ Sat(ϕ1) | ∀μ ∈ Steps(s) . ∃s′ ∈ R . μ(s′) > 0}
        if (R′ = R) then done := true
        R := R′
endwhile
return S\R
```

PCTL until for MDPs – Prob0A

- Maximum probabilities 0
  - $S^{no} = \{ s \in S \mid p_{max}(s, \phi_1 \text{ U } \phi_2)=0 \}$

```
PROB0A($Sat(\phi_1)$, $Sat(\phi_2)$)
$R := Sat(\phi_2)$
$done :=$ false
while ($done =$ false)
    $R' := R \cup \{s \in Sat(\phi_1) \mid \exists \mu \in Steps(s) \,.\, \exists s' \in R \,.\, \mu(s') > 0\}$
    if ($R' = R$) then $done :=$ true
    $R := R'$
endwhile
return $S \backslash R$
```

PCTL until for MDPs – Prob1E

- Maximum probabilities 1
  - S^{yes} = { $s \in S$ | $p_{max}(s, \phi_1 \text{ U } \phi_2)=1$ } = $Sat(\neg P_{<1} [\phi_1 \text{ U } \phi_2])$
- Prob1E algorithm (see next slide)
  - two nested loops (double fixed point)
  - result, stored in R, will be S^{yes}; initially R is S
  - iteratively remove (some) states u with $p_{max}(u, \phi_1 \text{ U } \phi_2)<1$
    - i.e. remove (some) states for which, under no adversary σ, is $Prob^{\sigma}(s, \phi_1 \text{ U } \phi_2)=1$
  - done by inner loop which computes subset R' of R
    - R' contains ϕ_1-states with a probability distribution for which all transitions stay within R and at least one eventually reaches ϕ_2
  - note: after first iteration, R contains:
    - { s | $Prob^A(s, \phi_1 \text{ U } \phi_2)>0$ for some A }
    - essentially: execution of Prob0A and removal of S^{no} from R

PCTL until for MDPs – Prob1E

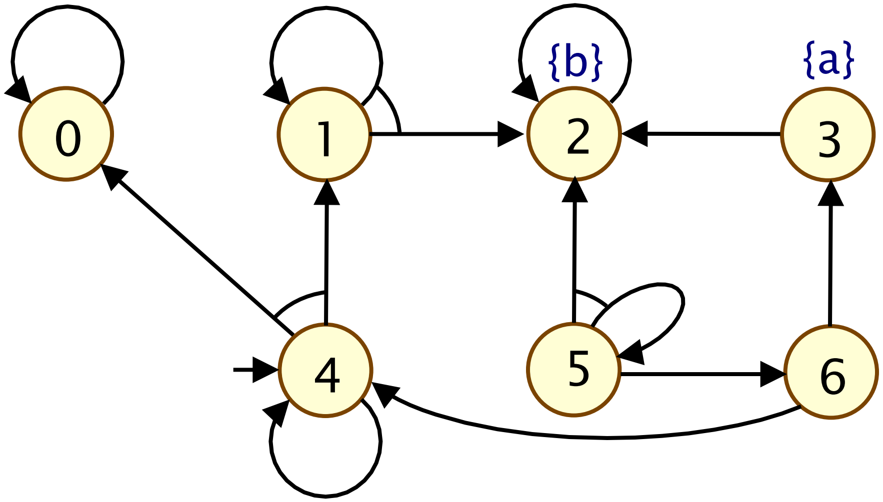
```
PROB1E($Sat(\phi_1), Sat(\phi_2)$)
$R := S$
$done :=$ false
while ($done =$ false)
    $R' := Sat(\phi_2)$
    $done' :=$ false
    while ($done' =$ false)
        $R'' := R' \cup \{s \in Sat(\phi_1) \mid \exists \mu \in Steps(s) \,.$
                $(\forall s' \in S \,.\, \mu(s') > 0 \rightarrow s' \in R) \wedge (\exists s' \in R' \,.\, \mu(s') > 0)\}$
        if ($R'' = R'$) then $done' :=$ true
        $R' := R''$
    endwhile
    if ($R' = R$) then $done :=$ true
    $R := R'$
endwhile
return $R$
```

Prob1E – Example

- $S^{yes} = \{ s \in S \mid p_{max}(s, \neg a \text{ U } b) = 1 \}$

- R = { 0, 1, 2, 3, 4 ,5 6 }
  - R' = {2} ; R' = {1, 2, 5} ; R' = {1, 2, 4, 5} ; R' = {1, 2, 4, 5, 6}
- R = { 1, 2, 4, 5, 6 }
  - R' = {2} ; R' = {1, 2, 5}
- R = { 1, 2, 5 }
  - R' = {2} ; R' = {1, 2, 5}
- R = { 1, 2, 5 }

- $S^{yes} = \{ 1, 2, 5 \}$



PCTL until for MDPs – Prob1A

- Minimum probabilities 1
  - $S^{yes} = \{ s \in S \mid p_{min}(s, \phi_1 \text{ U } \phi_2)=1 \}$

- Can also be done with a graph-based algorithm

- Details omitted here

- For minimum probabilities, just take $S^{yes} = Sat(\phi_2)$
  - recall that computing states for which probability=1 is just an optimisation: it is not required for correctness

PCTL until for MDPs

- Min/max probabilities for the remaining states, i.e. $S^? = S \setminus (S^{yes} \cup S^{no})$, can be computed using either…

- 1. Value iteration
  - approximate iterative solution method
  - preferable in practice for efficiency reasons

- 2. Reduction to a linear optimisation problem
  - solve with well-known linear programming (LP) techniques
    - Simplex, ellipsoid method, interior point method
  - yields exact solution in finite number of steps

- NB: Policy iteration also possible but not considered here

Method 1 – Value iteration (min)

- Minimum probabilities satisfy:
  - $p_{min}(s, \phi_1 \text{ U } \phi_2) = \lim_{n\to\infty} x_s^{(n)}$ where:

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

- Approximate iterative solution:
  - compute vector $\underline{x}^{(n)}$ for "sufficiently large" n
  - in practice: terminate iterations when some pre-determined convergence criteria satisfied
  - e.g. $\max_s | \underline{x}^{(n)}(s) - \underline{x}^{(n-1)}(s)) | < \varepsilon$ for some tolerance ε

Method 1 – Value iteration (max)

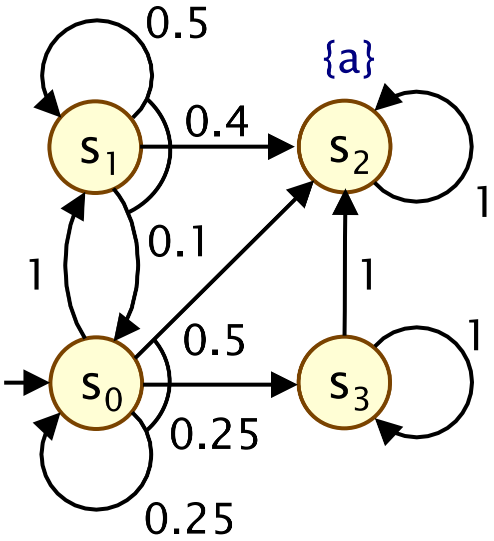
- Similarly, maximum probabilities satisfy:
  - $p_{max}(s, \phi_1 \text{ U } \phi_2) = \lim_{n\to\infty} x_s^{(n)}$ where:

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

- …and can be approximated iteratively

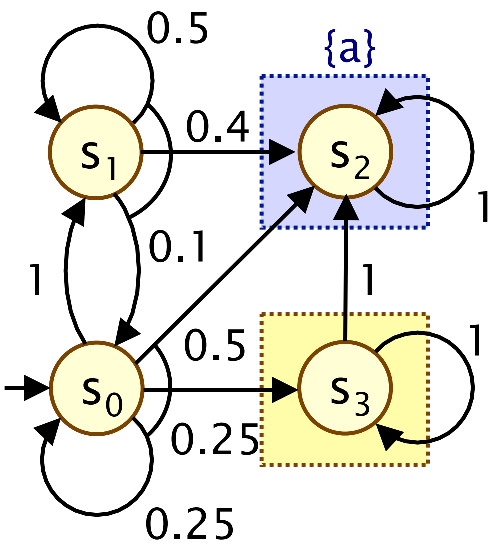
PCTL until – Example

- Model check: $P_{>0.5}$ [F a] ≡ $P_{>0.5}$ [true U a]
  - lower probability bound so minimum probabilities required



PCTL until – Example

- Model check: $P_{>0.5}$ [F a] ≡ $P_{>0.5}$ [true U a]
  - lower probability bound so minimum probabilities required

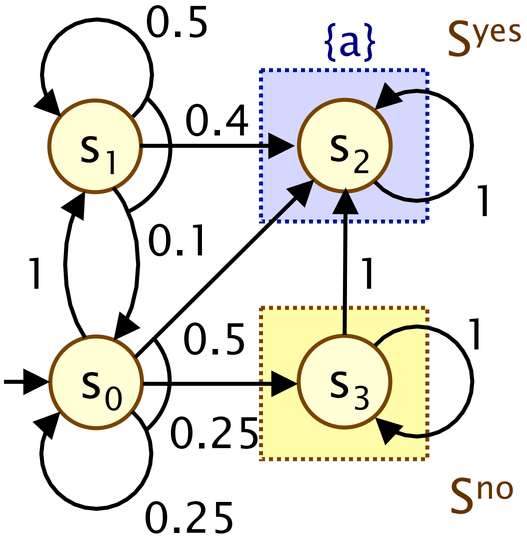


S^{yes} = Sat(a)

Prob0E

S^{no} = { s∈S | p_{min}(s, F a)=0 }

PCTL until – Example



Compute: $p_{min}(s_i, F\ a)$

$S^{yes} = \{s_2\}$, $S^{no} = \{s_3\}$, $S^{?} = \{s_0, s_1\}$

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

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

n=1: $[\ \min(1\cdot0,\ 0.25\cdot0+0.25\cdot0+0.5\cdot1),$

$0.1\cdot0+0.5\cdot0+0.4\cdot1,\ 1,\ 0\ ]$

$= [\ 0, 0.4, 1, 0\ ]$

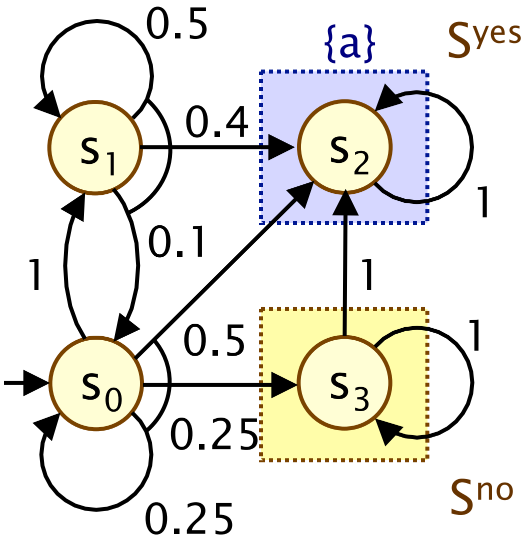
n=2: $[\ \min(1\cdot0.4, 0.25\cdot0+0.25\cdot0+0.5\cdot1),$

$0.1\cdot0+0.5\cdot0.4+0.4\cdot1,\ 1,\ 0\ ]$

$= [\ 0.4, 0.6, 1, 0\ ]$

n=3: …

PCTL until – Example



$\underline{p}_{min}(F\ a) =$
$[\ 2/3,\ 14/15,\ 1,\ 0\ ]$

$Sat(P_{>0.5}\ [F\ a]) = \{\ s_0,\ s_1,\ s_2\ \}$

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

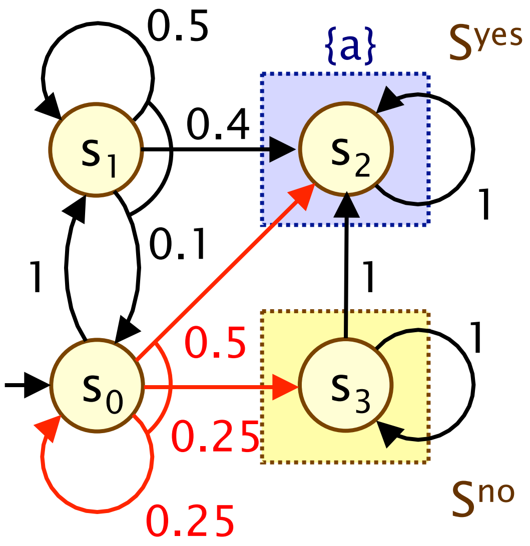
n=0: [0.000000, 0.000000, 1, 0]
n=1: [0.000000, 0.400000, 1, 0]
n=2: [0.400000, 0.600000, 1, 0]
n=3: [0.600000, 0.740000, 1, 0]
n=4: [0.650000, 0.830000, 1, 0]
n=5: [0.662500, 0.880000, 1, 0]
n=6: [0.665625, 0.906250, 1, 0]
n=7: [0.666406, 0.919688, 1, 0]
n=8: [0.666602, 0.926484, 1, 0]
…
n=20: [0.666667, 0.933332, 1, 0]
n=21: [0.666667, 0.933332, 1, 0]

$\approx [\ 2/3,\ 14/15,\ 1,\ 0\ ]$

Example – Optimal adversary

- Like for reachability, can generate an optimal memoryless adversary using min/max probability values
  - and thus also a DTMC

- Min adversary σ_{min}



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

…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

≈ [2/3, 14/15, 1, 0]

s_0 : min(1·14/15, 0.5·1+0.5·0+0.25·2/3)

=min(14/15, 2/3)

Method 2 – Linear optimisation problem

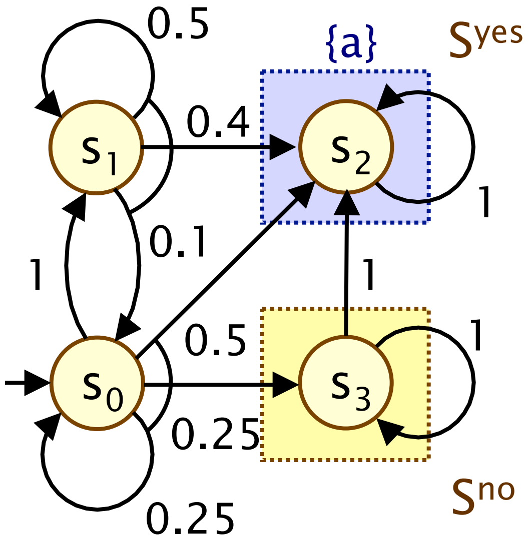
- Probabilities for states in $S^? = S \setminus (S^{yes} \cup S^{no})$ can also be obtained from a linear optimisation problem
- Minimum probabilities:

$$\boxed{\begin{array}{c} \text{maximize } \sum_{s \in S^?} x_s \text{ subject to the constraints:} \\ x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s') \\ \text{for all } s \in S^? \text{ and for all } (a, \mu) \in \mathbf{Steps}(s) \end{array}}$$

- Maximum probabilities:

$$\boxed{\begin{array}{c} \text{minimize } \sum_{s \in S^?} x_s \text{ subject to the constraints:} \\ x_s \geq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s') \\ \text{for all } s \in S^? \text{ and for all } (a, \mu) \in \mathbf{Steps}(s) \end{array}}$$

PCTL until – Example



Let $x_i = p_{min}(s_i, F\ a)$

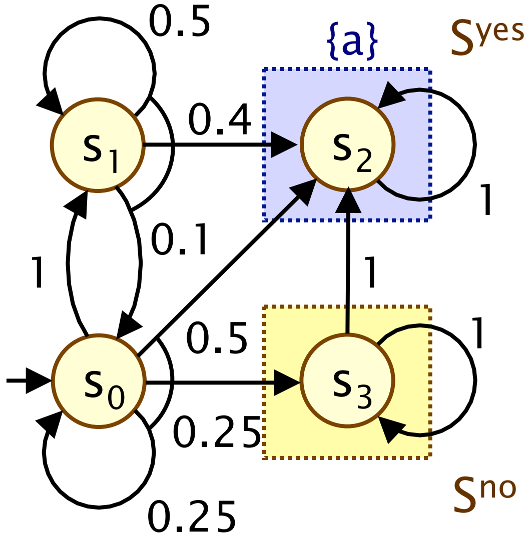
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

For $S^? = \{s_0, s_1\}$:

Maximise x_0+x_1 subject to constraints:

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

PCTL until – Example



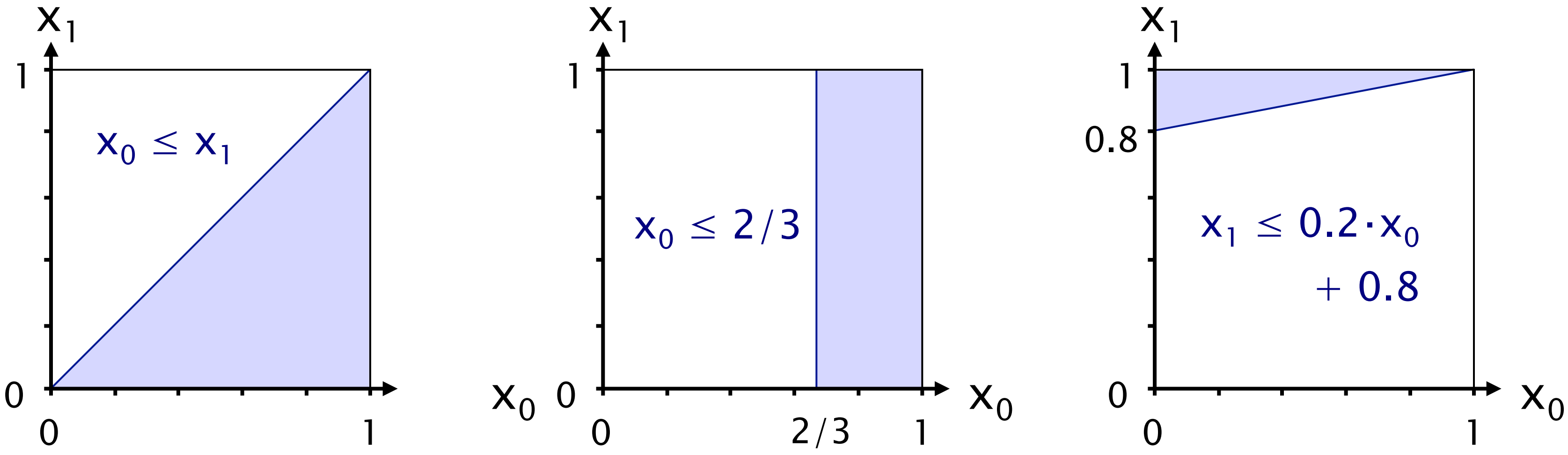
Let $x_i = p_{min}(s_i, F\ a)$

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

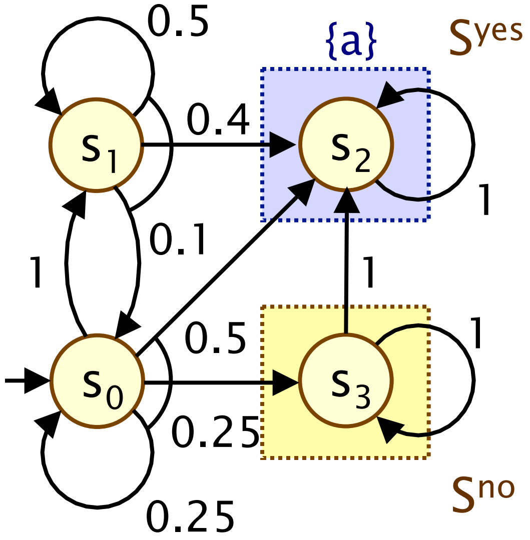
For $S^? = \{s_0, s_1\}$:

Maximise x_0+x_1 subject to constraints:

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



PCTL until – Example



Let $x_i = p_{min}(s_i, F\ a)$

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

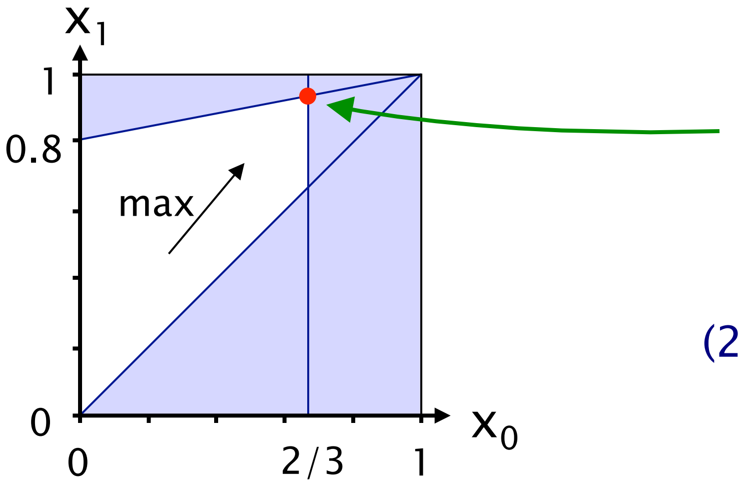
For $S^? = \{s_0, s_1\}$:

Maximise x_0+x_1 subject to constraints:

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

$\underline{p}_{min}(F\ a) =$
$[\ 2/3,\ 14/15,\ 1,\ 0\ ]$

$Sat(P_{>0.5}\ [F\ a]) =$
$\{\ s_0,\ s_1,\ s_2\ \}$



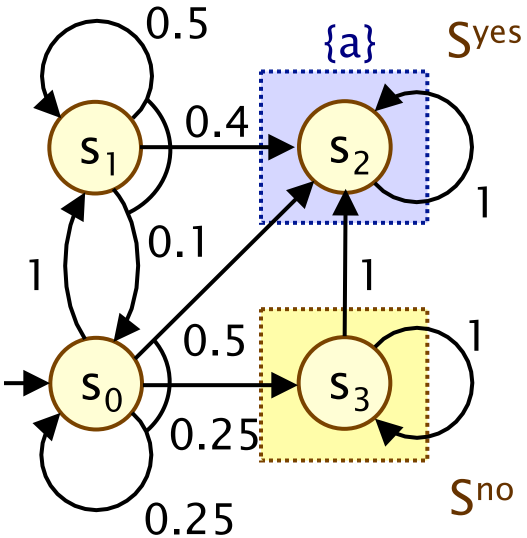
Solution:

(x_0, x_1)

$=$

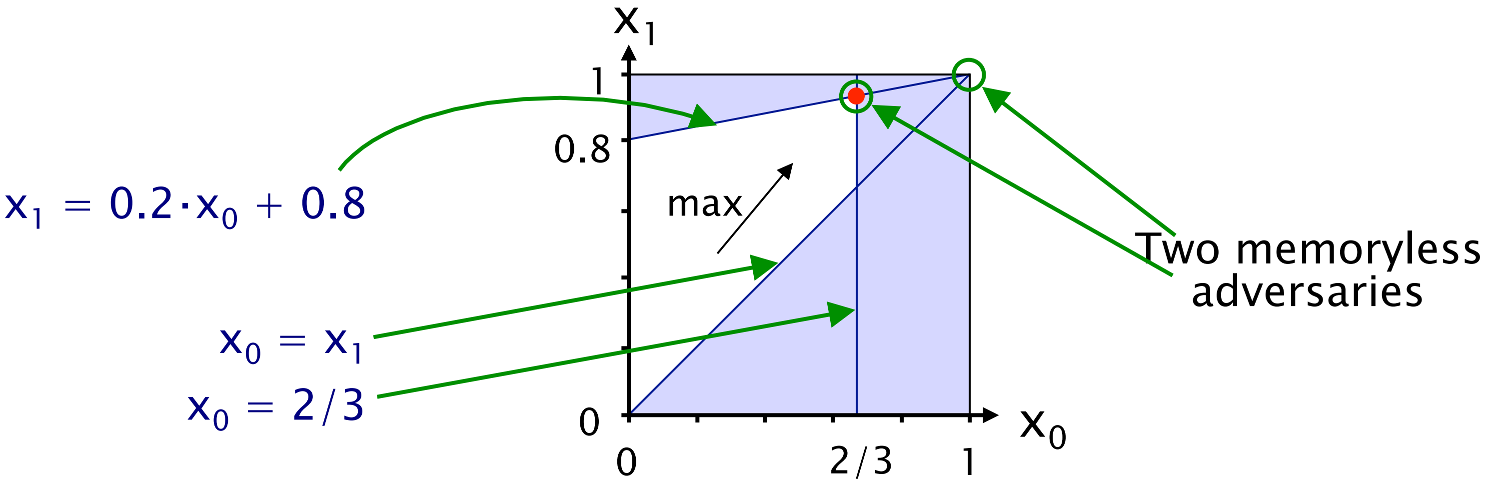
$(2/3, 14/15)$

Example – Optimal adversary



Get optimal adversary from constraints of optimisation problem that yield solution

Alternatively, use optimal probability values in value iteration function, as shown in value iteration example



$x_1 = 0.2 \cdot x_0 + 0.8$

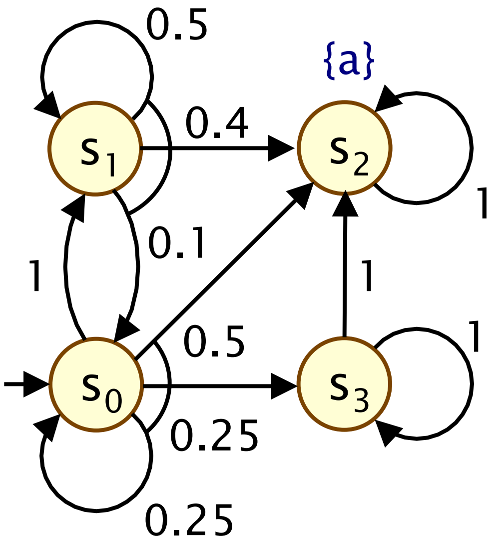
$x_0 = x_1$

$x_0 = 2/3$

Two memoryless adversaries

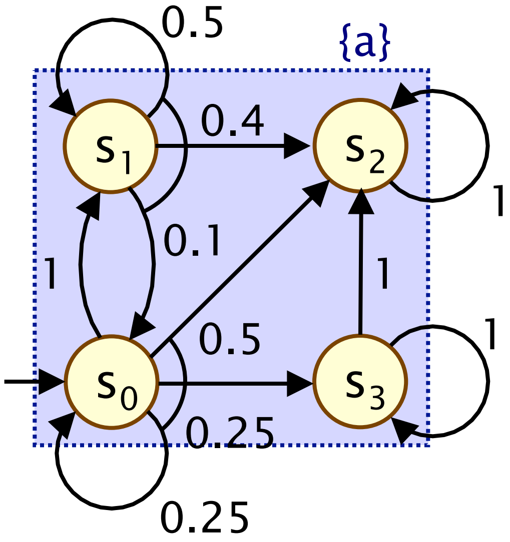
PCTL until – Example 2

- Model check: $P_{<0.1}$ [F a]
  - upper probability bound so maximum probabilities required



PCTL until – Example 2

- Model check: $P_{<0.1}$ [F a]
  - upper probability bound so maximum probabilities required



$S^{yes} = \{ s \in S \mid p_{min}(s, F\ a)=1 \} = S$

Prob1E

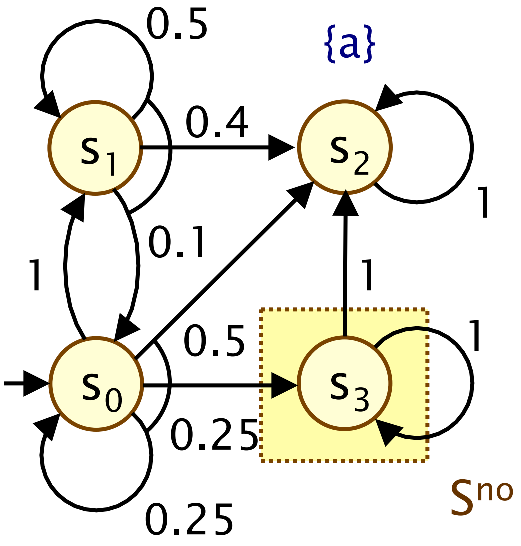
Prob0A

$S^{no} = \{ s \in S \mid p_{min}(s, F\ a)=0 \} = \varnothing$

- $\underline{p}_{max}(F\ a) = [\ 1, 1, 1, 1\ ]$ and $Sat(P_{<0.1}\ [\ F\ a\ ]) = \varnothing$

PCTL until – Example 3

- Model check: $P_{>0}$ [F a]
  - lower probability bound so minimum probabilities required
  - qualitative property so numerical computation can be avoided



$S^{no} = \{ s \in S \mid p_{min}(s, F\ a)=0 \}$

Prob0E yields $S^{no} = \{s_3\}$

- $\underline{p}_{min}(F\ a) = [\ ?, ?, ?, 0\ ]$ and $Sat(P_{>0}\ [\ F\ a\ ]) = \{s_0,s_1,s_2\}$

Costs and rewards

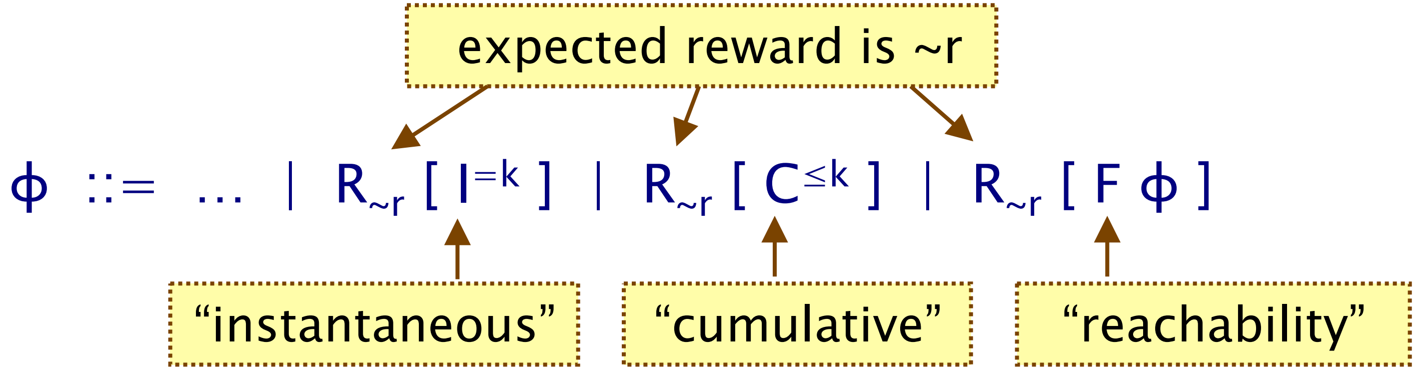
- We can augment MDPs with rewards (or costs)
  - real-valued quantities assigned to states and/or actions
  - different from the DTMC case where transition rewards assigned to individual transitions

- For a MDP (S,s_{init},**Steps**,L), a reward structure is a pair ($\underline{\rho}$,ι)
  - $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ is the state reward function
  - $\iota : S \times Act \rightarrow \mathbb{R}_{\geq 0}$ is transition reward function

- As for DTMCs these can be used to compute:
  - elapsed time, power consumption, size of message queue, number of messages successfully delivered, net profit, …

PCTL and rewards

- Augment PCTL with rewards based properties
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards



expected reward is ~r

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

"instantaneous"

"cumulative"

"reachability"

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

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

Types of reward formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - the expected value of the reward at time-step k is ~r for all adversaries
  - "the minimum 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 for all adversaries
  - "the maximum expected power consumption over one hour"

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

Reward formula semantics

- Formal semantics of the three reward operators:
  - for a state s in the MDP:
  - $s \models R_{\sim r} [I^{=k}] \Leftrightarrow Exp^{\sigma}(s, X_{I=k}) \sim r$ for all adversaries σ
  - $s \models R_{\sim r} [C^{\leq k}] \Leftrightarrow Exp^{\sigma}(s, X_{C\leq k}) \sim r$ for all adversaries σ
  - $s \models R_{\sim r} [F\ \Phi] \Leftrightarrow Exp^{\sigma}(s, X_{F\Phi}) \sim r$ for all adversaries σ

$Exp^{A}(s, X)$ denotes the expectation of the random variable $X : Path^{\sigma}(s) \to \mathbb{R}_{\geq 0}$ with respect to the probability measure Pr^{σ}_{s}

Reward formula semantics

- For an infinite path $\omega = s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$

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

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

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

where $k_\phi = \min\{\, i \mid s_i \vDash \phi \,\}$

Model checking reward formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - similar to the computation of bounded until probabilities
  - solution of recursive equations
  - k matrix-vector multiplications (+ min/max)

- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - extension of bounded until computation
  - solution of recursive equations
  - k iterations of matrix-vector multiplication + summation

- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to the case for until
  - solve a linear optimization problem (or value iteration)

Model checking complexity

- For model checking of an MDP $(S,s_{init},\mathbf{Steps},L)$ and PCTL formula ϕ (including reward operators)
  - complexity is linear in $|\Phi|$ and polynomial in $|S|$
- Size $|\phi|$ of ϕ is defined as number of logical connectives and temporal operators plus sizes of temporal operators
  - model checking is performed for each operator
- Worst-case operators are $P_{\sim p}$ [ϕ_1 U ϕ_2] and $R_{\sim r}$ [F ϕ]
  - main task: solution of linear optimization problem of size $|S|$
  - can be solved with ellipsoid method (polynomial in $|S|$)
  - and also precomputation algorithms (max $|S|$ steps)

Summing up…

- PCTL for MDPs
  - same as syntax as for PCTL
  - key difference in semantics: “for all adversaries”
  - requires computation of minimum/maximum probabilities
- PCTL model checking for MDPs
  - same basic algorithm as for DTMCs
  - next: matrix-vector multiplication + min/max
  - bounded until: k matrix-vector multiplications + min/max
  - until : precomputation algorithms + numerical computation
    - precomputation: Prob0A and Prob1E for max, Prob0E for min
    - numerical computation: value iteration, linear optimisation
  - complexity linear in $|\Phi|$ and polynomial in $|S|$
- Costs and rewards

Lecture 15

Long-run properties of DTMCs and MDPs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- LTL – Linear temporal logic
- Repeated reachability and persistence
- Long-run properties of DTMCs
  - bottom strongly connected components (BSCCs)
- Long-run properties of MDPs
  - end components (E.C.s)

Limitations of PCTL

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

- More expressive logics can be used, for example:
  - LTL [Pnu77] – the non-probabilistic linear-time temporal logic
  - PCTL* [ASB+95,BdA95] – which subsumes both PCTL and LTL
  - both allow path operators to be combined

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

Review – CTL and PCTL

- CTL:
  - $\phi ::= \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid A\,\psi \mid E\,\psi$
  - $\psi ::= X\,\phi \mid \phi\, U\, \phi$

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

- Notation for paths: $\omega = s_0 s_1 s_2 \ldots$
  - Path(s) = set of all (infinite) paths with $s_0 = s$
  - $\omega(i)$ denotes the (i+1)th state, i.e. $\omega(i) = s_i$
  - $\omega[i\ldots]$ is the suffix starting from s_i, i.e. $\omega[i\ldots] = s_i s_{i+1} s_{i+2} \ldots$

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

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

LTL – Linear temporal logic

- Derived operators like CTL, for example:
  - $F\ \psi \equiv \text{true}\ U\ \psi$
  - $G\ \psi \equiv \neg F(\neg\psi)$

- LTL semantics (non-probabilistic)
  - implicit universal quantification over paths
  - i.e. for an LTS $M = (S,s_{init},\rightarrow,L)$ and LTL formula ψ
  - $s \vDash \psi$ iff $\omega \vDash \psi$ for all paths $\omega \in Path(s)$
  - $M \vDash \psi$ iff $s_{init} \vDash \psi$
- e.g:
  - $A\ F\ (req \wedge X\ ack)$
  - "it is always possible that a request, followed immediately by an acknowledgement, can occur"

More LTL examples

- $(\mathrm{F}\ tmp\_fail_1) \wedge (\mathrm{F}\ tmp\_fail_2)$
    - "both servers suffer temporary failures at some point"
- GF ready
    - "the server always eventually returns to a ready-state"
- G (req → F ack)
    - "requests are always followed by an acknowledgement"
- FG stable
    - "the system reaches and stays in a 'stable' state"

Branching vs. Linear time

- LTL but not CTL:
  - FG stable
  - “the system reaches and stays in a ‘stable’ state”
  - e.g. A FG stable ≢ AF AG stable

- CTL but not LTL:
  - AG EF init
  - e.g. “for every computation, it is always possible to return to the initial state”

LTL + probabilities

- Same idea as PCTL: probabilities of sets of path formulae
  - for a state s of a DTMC and an LTL formula ψ:
  - $\mathrm{Prob}(s, \psi) = \mathrm{Pr}_s \{ \omega \in \mathrm{Path}(s) \mid \omega \vDash \psi \}$
  - all such path sets are measurable (see later lecture)

- For MDPs, we can again consider lower/upper bounds
  - $p_{\min}(s, \psi) = \inf_{\sigma \in Adv} \mathrm{Prob}^{\sigma}(s, \psi)$
  - $p_{\max}(s, \psi) = \sup_{\sigma \in Adv} \mathrm{Prob}^{\sigma}(s, \psi)$
  - (for LTL formula ψ)

- For DTMCs or MDPs, an LTL specification often comprises an LTL (path) formula and a probability bound
  - e.g. $P_{>0.99}$ [F (req $\wedge$ X ack)]

PCTL*

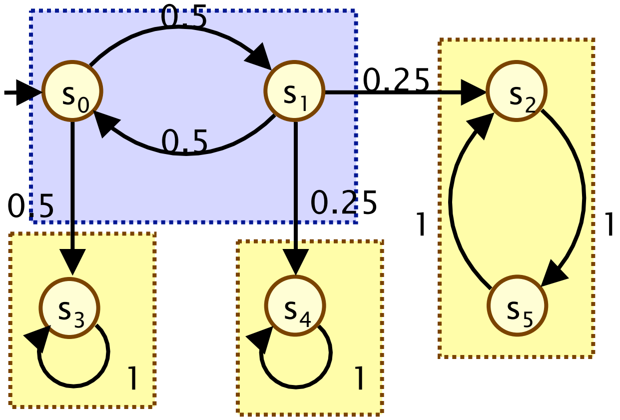
- PCTL* subsumes both (probabilistic) LTL and PCTL

- State formulae:
  - $\phi \ ::= \ \text{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid P_{\sim p} [\ \psi\ ]$
  - where $a \in AP$, $\sim \in \{<,>,\leq,\geq\}$, $p \in [0,1]$ and ψ a path formula
- Path formulae:
  - $\psi \ ::= \phi \mid \psi \wedge \psi \mid \neg\psi \mid X\ \psi \mid \psi\ U\ \psi$
  - where ϕ is a state formula

- A PCTL* formula is a state formula ϕ
  - e.g. $P_{>0.99} [\ GF\ crit_1\ ] \wedge P_{>0.99} [\ GF\ crit_2\ ]$
  - e.g. $P_{\geq 0.75} [\ GF\ P_{>0} [\ F\ init\ ]\ ]$

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
- With probability 1, a BSCC will be reached and all of its states visited infinitely often



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

Repeated reachability – DTMCs

- Repeated reachability:
  - "always eventually…" or "infinitely often…"

- e.g. "what is the probability that the protocol successfully sends a message infinitely often?"

- Using LTL notation:
  - $\omega \vDash \mathbf{GF}\ a$

    $\Leftrightarrow$
  - $\forall\ i \geq 0\ .\ \exists\ j \geq i\ .\ \omega(j) \in \mathrm{Sat}(a)$

- Prob(s, GF a)

  $= \mathrm{Pr}_s\{\ \omega \in \mathrm{Path}(s) \mid \forall\ i \geq 0\ .\ \exists\ j \geq i\ .\ \omega(j) \in \mathrm{Sat}(a)\ \}$

Qualitative repeated reachability

- $Pr_s\{ \omega \in Path(s) \mid \forall\, i \geq 0 \,.\, \exists\, j \geq i \,.\, \omega(j) \in Sat(a) \} = 1$
- $P_{\geq 1}$ [GF a]

PCTL*

if and only if

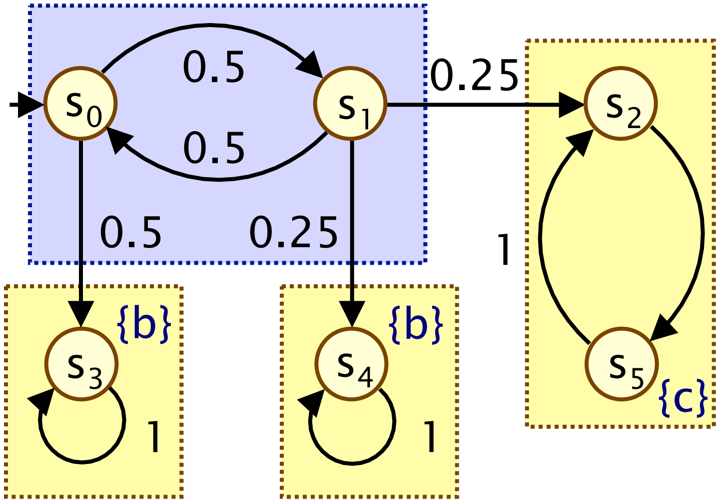
- $T \cap Sat(a) \neq \emptyset$ for all BSCCs T reachable from s

Examples:

$s_0 \vDash P_{\geq 1}$ [GF (b∨c)]

$s_0 \nvDash P_{\geq 1}$ [GF b]

$s_2 \vDash P_{\geq 1}$ [GF c]



Quantitative repeated reachability

- $\text{Prob}(s, \text{GF } a) = \text{Prob}(s, \text{F } T_{\text{GFa}})$
  - where T_{GFa} = union of all BSCCs T with $T \cap \text{Sat}(a) \neq \varnothing$

Example:

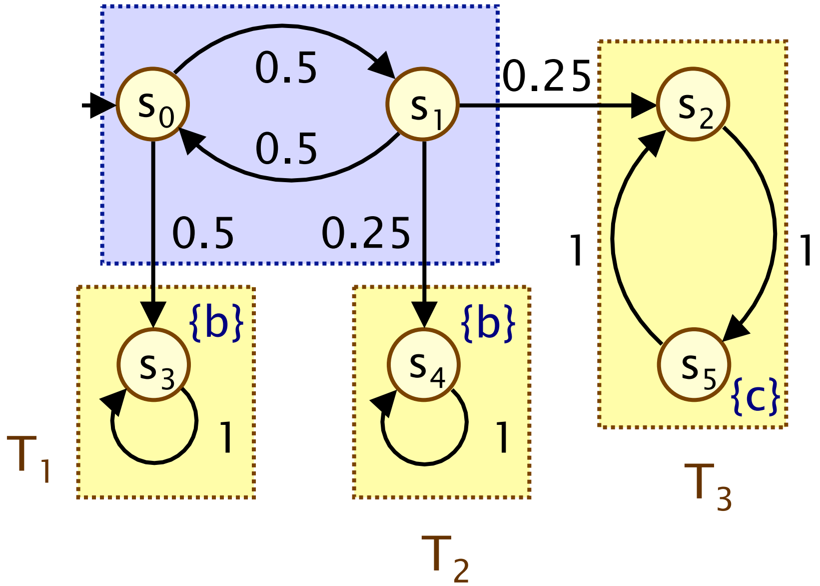
$\text{Prob}(s_0, \text{GF } b)$

$= \text{Prob}(s_0, \text{F } T_{\text{GFb}})$

$= \text{Prob}(s_0, \text{F } (T_1 \cup T_2))$

$= \text{Prob}(s_0, \text{F } \{s_3,s_4\})$

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



- From the above, we also have:
  - $P_{>0}$ [GF a] ⇔ $T \cap \text{Sat}(a) \neq \varnothing$ for some reachable BSCC T

Persistence – DTMCs

- Persistence properties: “eventually always…”
  - e.g. “what is the probability of the leader election algorithm reaching, and staying in, a stable state?”
  - e.g. “what is the probability that an irrecoverable error occurs?”

- Using LTL notation:
  - $\omega \vDash \mathbf{FG}\ a$

    $\Leftrightarrow$
  - $\exists\ i \geq 0\ .\ \forall\ j \geq i\ .\ \omega(j) \in Sat(a)$

- Prob(s, FG a)

  $= Pr_s\{\ \omega \in Path(s)\ |\ \exists\ i \geq 0\ .\ \forall\ j \geq i\ .\ \omega(j) \in Sat(a)\ \}$

Qualitative persistence

- $\mathrm{Pr}_s\{\, \omega \in \mathrm{Path}(s) \mid \exists\, i \geq 0 \,.\, \forall\, j \geq i \,.\, \omega(j) \in \mathrm{Sat}(a) \,\} = 1$
- $P_{\geq 1}$ [FG a]

if and only if

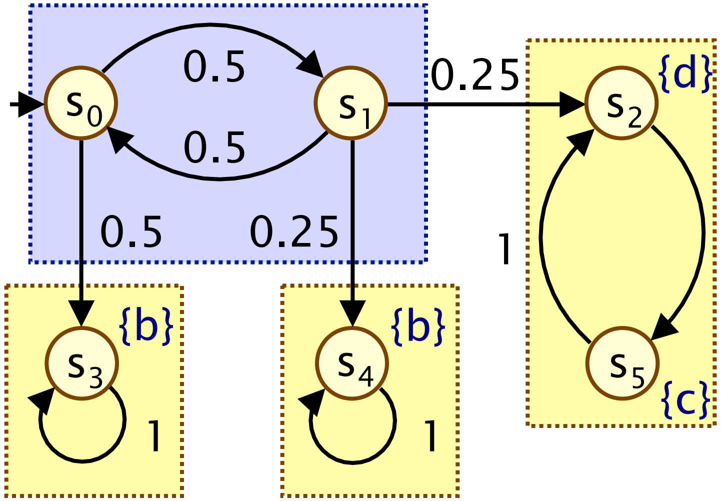
- T ⊆ Sat(a) for all BSCCs T reachable from s

Examples:

$s_0 \nvDash P_{\geq 1}$ [FG (b∨c)]

$s_0 \vDash P_{\geq 1}$ [FG (b∨c∨d)]

$s_2 \vDash P_{\geq 1}$ [FG (c∨d)]



Quantitative persistence

- Prob(s, FG a) = Prob(s, F T_{FGa})
  - where T_{FGa} = union of all BSCCs T with $T \subseteq Sat(a)$

Example:

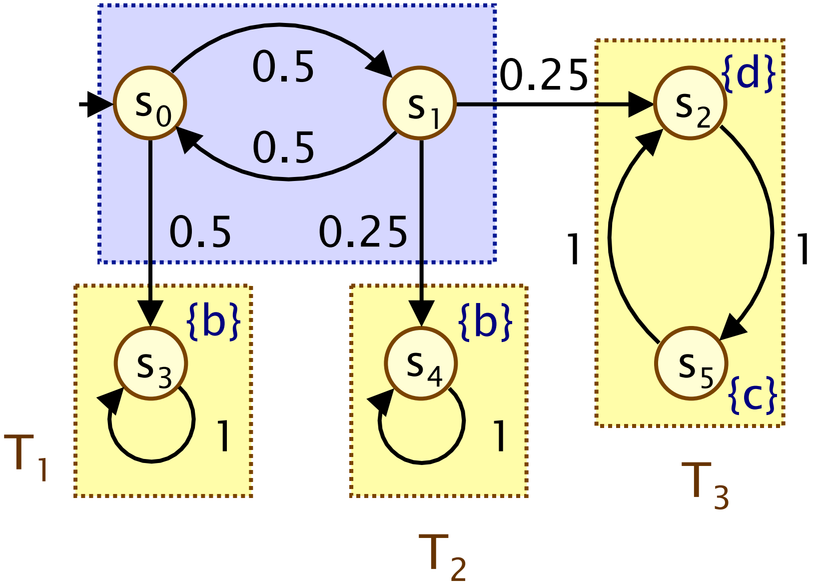
Prob(s_0, FG ($b \vee c$))

= Prob(s_0, F $T_{FG(b \vee c)}$)

= Prob(s_0, F ($T_1 \cup T_2$))

= Prob(s_0, F $\{s_3, s_4\}$)

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



Success sets

- The sets T_P for property P are called success sets
  - T_{GFa} = union of all BSCCs T with $T \cap Sat(a) \neq \emptyset$
  - T_{FGa} = union of all BSCCs T with $T \subseteq Sat(a)$
- Sometimes denoted U_P
  - e.g. U_{GFa}
  - we use T_p here (to avoid confusion with the until operator)

Repeated reachability + persistence

- Repeated reachability and persistence are dual properties
  - $GF\ a \equiv \neg(FG\ \neg a)$
  - $FG\ a \equiv \neg(GF\ \neg a)$
- Hence, for example:
  - $Prob(s, GF\ a) = 1 - Prob(s, FG\ \neg a)$

- Can show this through LTL equivalences, or…

- $Prob(s, GF\ a) + Prob(s, FG\ \neg a)$

$= Prob(s, F\ T_{GFa}) + Prob(s, F\ T_{FG\neg a})$

  - T_{GFa} = union of BSCCs T with $T \cap Sat(a) \neq \varnothing$ (T intersects Sat(a))
  - $T_{FG\neg a}$ = union of BSCCs T with $T \subseteq (S \setminus Sat(a))$ (no intersection)

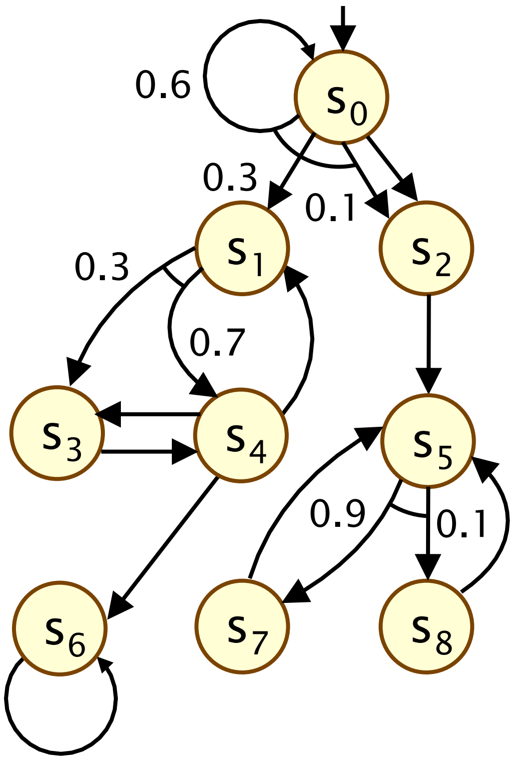
$= Prob(s, F\ (T_{GFa} \cup T_{FG\neg a})) = 1$ (fundamental DTMC property)

End components of MDPs

- Consider an MDP M = (S,s_{init},**Steps**,L)

- A sub-MDP of M is a pair (T,**Steps**') where:
  - T ⊆ S is a (non-empty) subset of M's states
  - **Steps**'(s) ⊆ **Steps**(s) for each s ∈ T
  - (T,**Steps**') is closed under probabilistic branching, i.e. the set of states { s' | μ(s')>0 for some (a,μ)∈**Steps**'(s) } is a subset of T

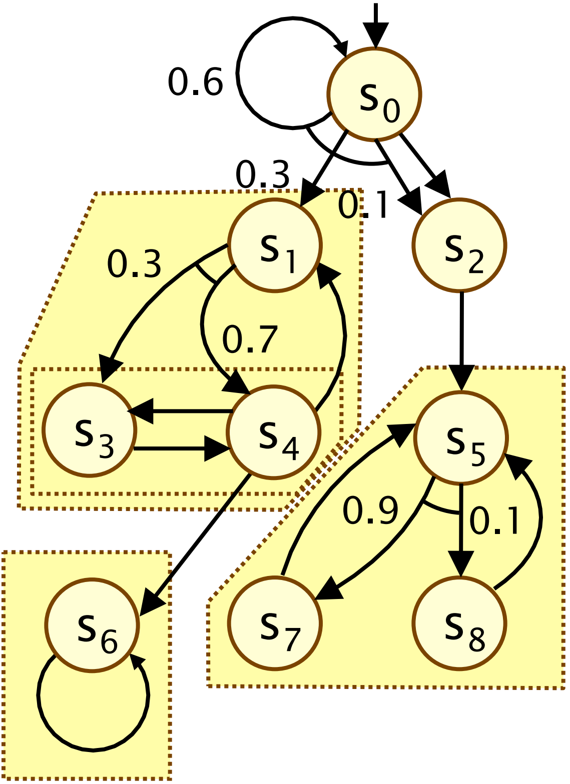
- An end component of M is a strongly connected sub-MDP



Note:
- action labels omitted
- probabilities omitted where =1

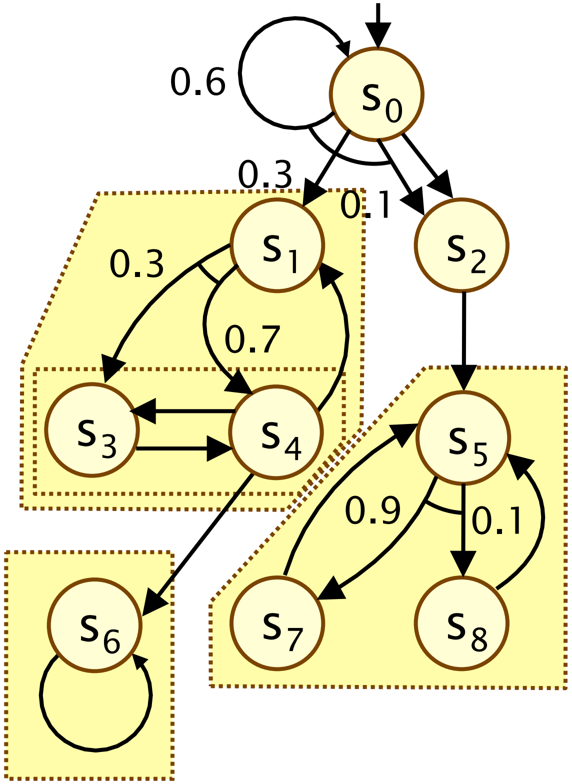
End components – Examples

- Sub-MDPs
  - can be formed from state sets such as:
  - $\{s_2,s_5,s_7,s_8\}$, $\{s_0,s_2,s_5,s_7,s_8\}$, $\{s_5,s_7,s_8\}$,
  - $\{s_1,s_3,s_4\}$, $\{s_1,s_3,s_4,s_6\}$, $\{s_3,s_4\}$, …
- End components
  - can be formed from state sets:
  - $\{s_3,s_4\}$, $\{s_1,s_3,s_4\}$, $\{s_6\}$, $\{s_5,s_7,s_8\}$
- Note that
  - state sets do not necessarily uniquely identify end components
  - e.g. $\{s_1,s_3,s_4\}$



End components of MDPs

- For finite MDPs…
  - (analogue of fundamental property of finite DTMCs)
- For every end component, there is an adversary which, with probability 1, forces the MDP to remain in the end component, and visit all its states infinitely often
- Under every adversary σ, with probability 1 an end component will be reached and all of its states visited infinitely often



Repeated reachability – MDPs (max)

- Repeated reachability (GF) for MDPs
  - consider first the case of maximum probabilities…
  - $p_{max}(s, GF\ a)$

- First, a simple qualitative property:
  - $Prob^{\sigma}(s, GF\ a) > 0$ for some adversary σ, i.e. $p_{max}(s, GF\ a) > 0$

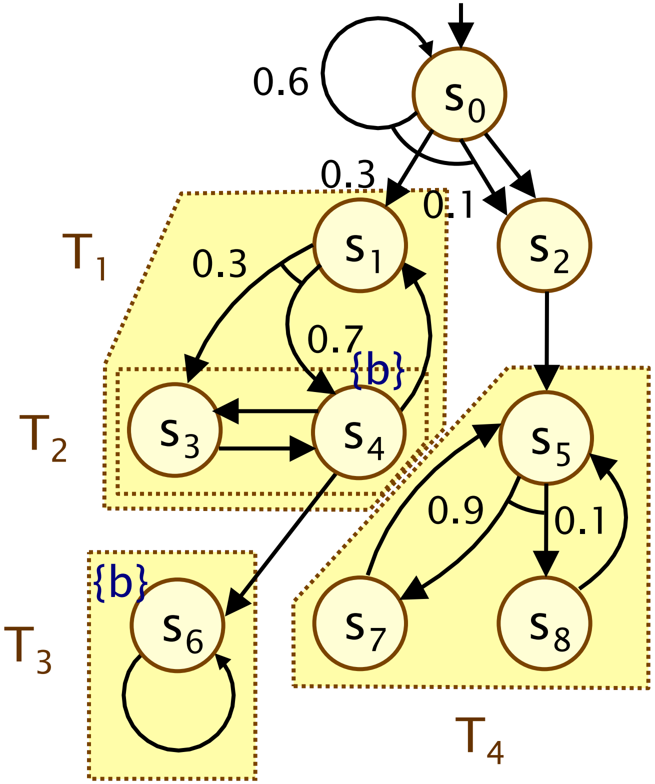
    $\Leftrightarrow$
  - $T \cap Sat(a) \neq \emptyset$ for some end component T reachable from s

- The quantitative case (for maximum probabilities):
  - $p_{max}(s, GF\ a) = p_{max}(s, F\ T_{GFa})$
  - where T_{GFa} is the union of sets T for all end components (T,**Steps'**) with $T \cap Sat(a) \neq \emptyset$ (i.e. at least one a-state in T)

Example

- Check: $P_{<0.8}$ [GF b] for s_0

- Compute p_{max}(GF b)
  - p_{max}(GF b) = p_{max}(s, F T_{GFb})
  - T_{GFb} is the union of sets T for all end components with $T \cap Sat(b) \neq \varnothing$
  - Sat(b) = { s_4, s_6 }
  - $T_{GFb} = T_1 \cup T_2 \cup T_3 = \{ s_1, s_3\ s_4, s_6 \}$
  - p_{max}(s, F T_{GFb}) = 0.75
  - p_{max}(GF b) = 0.75



- Result: $s_0 \vDash P_{<0.8}$ [GF b]

Repeated reachability – MDPs (max)

- Quantitative case:
  - $p_{max}(s, GF\ a) = p_{max}(s, F\ T_{GFa})$
- This yields the qualitative property given earlier:
  - $Prob^{\sigma}(s, GF\ a) > 0$ for some adversary σ

    $\Leftrightarrow\ p_{max}(s, GF\ a) > 0$

    $\Leftrightarrow\ p_{max}(s, F\ T_{GFa}) > 0$

    $\Leftrightarrow\ Prob^{\sigma}(s, F\ T_{GFa}) > 0$ for some adversary σ

    $\Leftrightarrow\ s \vDash EF\ T_{GFa}$

    $\Leftrightarrow\ T \cap Sat(a) \neq \varnothing$ for some E.C. T reachable from s
- Another qualitative property:
  - $Prob^{\sigma}(s, GF\ a) = 1$ for some adversary σ

    $\Leftrightarrow\ p_{max}(s, GF\ a) = 1$

    $\Leftrightarrow\ p_{max}(s, F\ T_{GFa}) = 1$

Compute with Prob1E

Repeated reachability – MDPs (min)

- Repeated reachability for MDPs – minimum probabilities
  - p_{min}(s, GF a)

- First, a useful qualitative property:
  - $Prob^{\sigma}(s, GF\ a) = 1$ for all adversaries σ

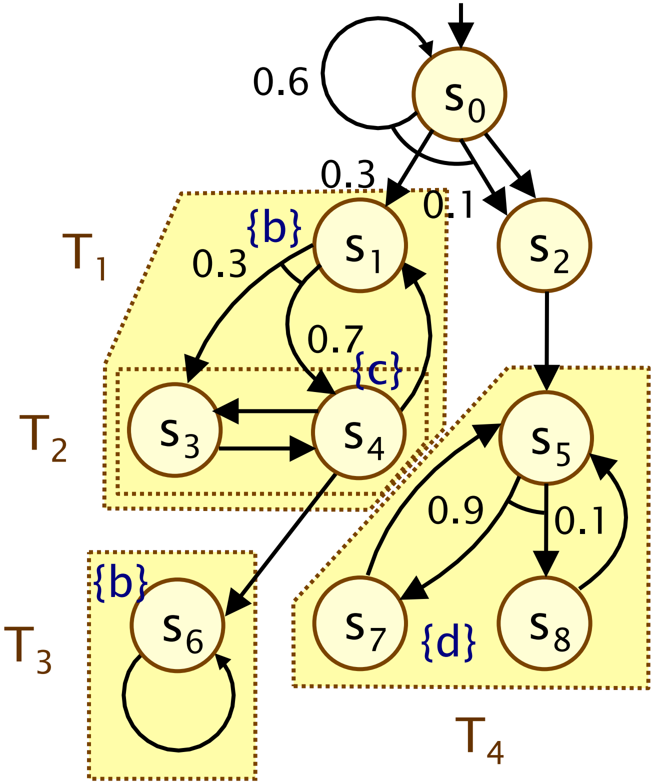
    $\Leftrightarrow$
  - $s \vDash P_{\geq 1}$ [GF a]

    PCTL*

    $\Leftrightarrow$
  - $T \cap Sat(a) \neq \varnothing$ for all end components T reachable from s

Examples

- $s_0 \vDash P_{\geq 1}$ [GF (b∨c∨d)] ?



- $s_0 \vDash P_{\geq 1}$ [GF (b∨d)] ?

Repeated reachability – MDPs (min)

- Repeated reachability for MDPs – minimum probabilities
  - $p_{min}(s, GF\ a)$

- Quantitative case
  - use duality of min/max probabilities for MDPs
  - $p_{min}(s, \psi) = 1 - p_{max}(s, \neg\psi)$
  - e.g. $p_{min}(s, GF\ a) = 1 - p_{max}(s, FG\neg a)$

- So min probabilities for repeated reachability (GF)
  - can be computed as max probabilities for persistence (FG)

Persistence – MDPs

- Persistence for MDPs
  - p_{min}(s, FG a) or p_{max}(s, FG a)

- Quantitative case – maximum probabilities
  - p_{max}(s, FG a) = p_{max}(s, F T_{FGa})
  - where T_{FGa} is the union of sets T for all end components (T,**Steps'**) with $T \subseteq Sat(a)$ (i.e. all states in T satisfy a)

Repeated reachability (again)

- We now have way a of computing minimum probabilities for repeated reachability (GF)
  - $p_{min}(s, \text{GF } a) = 1 - p_{max}(s, \text{FG}\neg a)$
    $= 1 - p_{max}(s, \text{F } T_{\text{FG}\neg a})$
  - where $T_{\text{FG}\neg a}$ is the union of sets T for all end components (T,**Steps'**) with $T \subseteq S \backslash Sat(a)$
  - ie. $T_{\text{FG}\neg a}$ is the union of sets T for all end components (T,**Steps'**) with $T \cap Sat(a) = \varnothing$

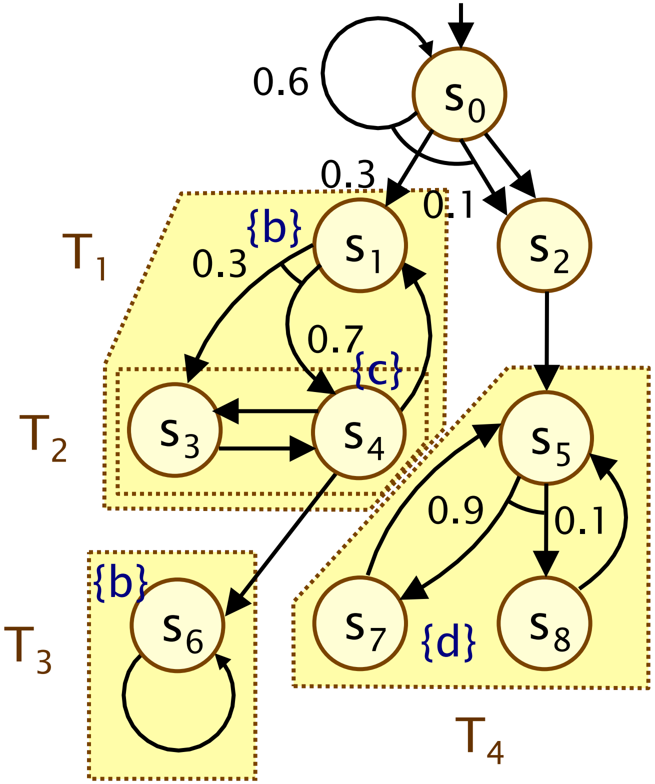
Opposite of condition for GFa

- Can also now show why:
  - $s \vDash P_{\geq 1}$ [GF a]
    $\Leftrightarrow$
  - $T \cap Sat(a) \neq \varnothing$ for all end components T reachable from s

Examples

- $s_0 \vDash P_{>0}$ [GF d] ?

- $s_0 \vDash P_{>0.3}$ [GF d] ?



Summing up… I

- LTL: path-based, path operators can be combined
- PCTL*: subsumes PCTL and LTL

CTL	Φ	non-probabilistic (LTSs)
LTL	ψ	
PCTL	Φ	probabilistic (DTMCs, MDPs)
LTL + prob.	Prob(s, ψ)	
PCTL*	Φ	

Summing up… II

- 2 useful instances of LTL formulae:
  - repeated reachability: GF a
  - persistence: FG a
- DTMCs
  - qualitative: properties of reachable BSCCs
  - quantitative: probability of reaching success set (BSCC set)
- MDPs
  - end components: MDP analogue of BSCCs
  - $p_{max}(s, GF\ a)$ – max. reachability of success set ($T \cap Sat(a) \neq \emptyset$)
  - $P_{\geq 1}$ [GF a] – reachability of end components
  - $p_{min}(s, GF\ a)$ – one minus max. prob. for dual property
  - $p_{max}(s, FG\ a)$ – max. reachability of success set ($T \subseteq Sat(a)$)
  - $p_{min}(s, FG\ a)$ – again, via dual property

Lecture 16
Automata-based properties

Dr. Dave Parker

Department of Computer Science
University of Oxford

Property specifications

- 1. Reachability properties, e.g. in PCTL
  - F a or $F^{\leq t}$ a (reachability)
  - a U b or a $U^{\leq t}$ b (until – constrained reachability)
  - G a (invariance) (dual of reachability)
  - probability computation: graph analysis + solution of linear equation system (or linear optimisation problem)

- 2. Long-run properties, e.g. in LTL
  - GF a (repeated reachability)
  - FG a (persistence)
  - probability computation: BSCCs + probabilistic reachability

- This lecture: more expressive class for type 1

Overview

- Nondeterministic finite automata (NFA)
- Regular expressions and regular languages
- Deterministic finite automata (DFA)
- Regular safety properties
- DFAs and DTMCs

Some notation

- Let Σ be a finite alphabet

- A (finite or infinite) word w over Σ is
  - a sequence of $\alpha_1\alpha_2\ldots$ where $\alpha_i \in \Sigma$ for all i

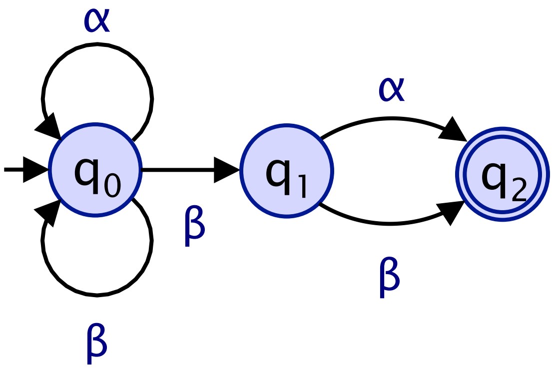
- A prefix w' of word w = $\alpha_1\alpha_2\ldots$ is
  - a finite word $\beta_1\,\beta_2\ldots\,\beta_n$ with $\beta_i=\alpha_i$ for all $1\leq i\leq n$

- Σ^* denotes the set of finite words over Σ

- Σ^ω denotes the set of infinite words over Σ

Finite automata

- A nondeterministic finite automaton (NFA) is…
  - a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where:
  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
  - $Q_0 \subseteq Q$ is a set of initial states
  - $F \subseteq Q$ is a set of "accept" states



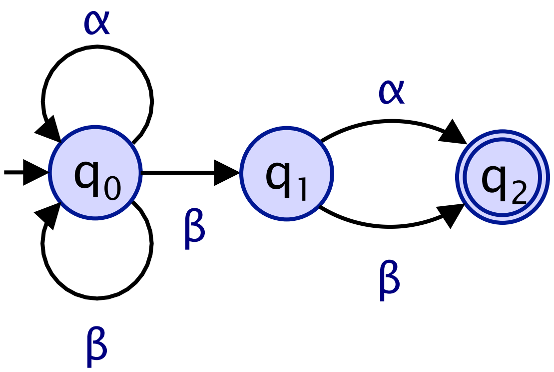
Language of an NFA

- Consider an NFA $A = (Q, \Sigma, \delta, Q_0, F)$

- A run of A on a finite word $w=\alpha_1\alpha_2\ldots\alpha_n$ is:
  - a sequence of automata states $q_0q_1\ldots q_n$ such that:
  - $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \alpha_{i+1})$ for all $0 \leq i < n$
- An accepting run is a run with $q_n \in F$

- Word w is accepted by A iff:
  - there exists an accepting run of A on w
- The language of A, denoted L(A) is:
  - the set of all words accepted by A
- Automata A and A' are equivalent if L(A)=L(A')

Example – NFA



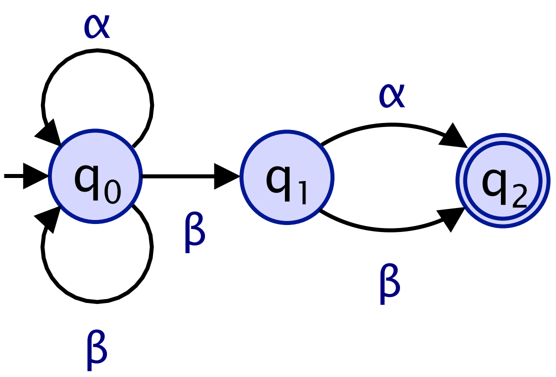
Regular expressions

- Regular expressions E over a finite alphabet Σ
  - are given by the following grammar:
  - $E ::= \emptyset \mid \varepsilon \mid \alpha \mid E + E \mid E.E \mid E^*$
  - where $\alpha \in \Sigma$

- Language $L(E) \subseteq \Sigma^*$ of a regular expression:
  - $L(\emptyset) = \emptyset$ (empty language)
  - $L(\varepsilon) = \{ \varepsilon \}$ (empty word)
  - $L(\alpha) = \{ \alpha \}$ (symbol)
  - $L(E_1 + E_2) = L(E_1) \cup L(E_2)$ (union)
  - $L(E_1.E_2) = \{ w_1.w_2 \mid w_1 \in L(E_1) \text{ and } w_2 \in L(E_2) \}$ (concatenation)
  - $L(E^*) = \{ w^i \mid w \in L(E) \text{ and } i \in \mathbb{N} \}$ (finite repetition)

Regular languages

- A set of finite words L is a regular language…
  - iff L = L(E) for some regular expression E
  - iff L = L(A) for some finite automaton A

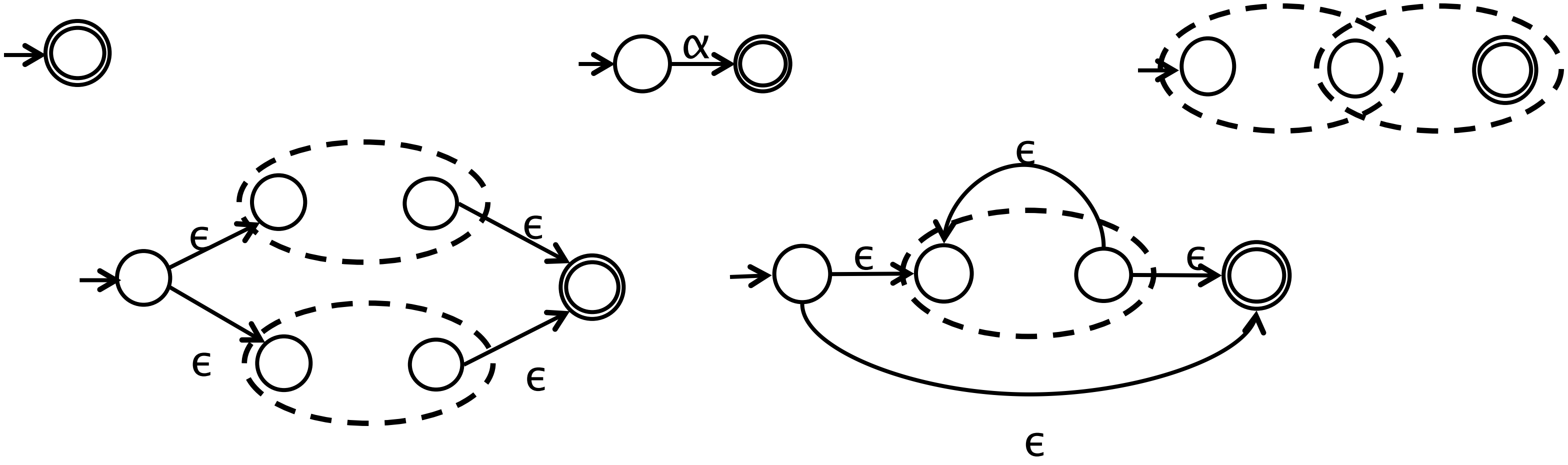


$(\alpha+\beta)^*\beta(\alpha+\beta)$

(i.e. penultimate symbol is β)

Operations on NFA

- Can construct NFA from regular expression inductively
  - includes addition (and then removal) of ε–transitions



- Can construct the intersection of two NFA
  - build (synchronised) product automaton
  - cross product of $A_1 \otimes A_2$ accepts $L(A_1) \cap L(A_2)$

Deterministic finite automata

- A finite automaton is deterministic if:
  - $|Q_0|=1$
  - $|\delta(q, \alpha)| \leq 1$ for all $q \in Q$ and $\alpha \in \Sigma$
  - i.e. one initial state and no nondeterministic successors

- A deterministic finite automaton (DFA) is total if:
  - $|\delta(q, \alpha)| = 1$ for all $q \in Q$ and $\alpha \in \Sigma$
  - i.e. unique successor states

- A total DFA
  - can always be constructed from a DFA
  - has a unique run for any word $w \in \Sigma^*$

Determinisation: NFA → DFA

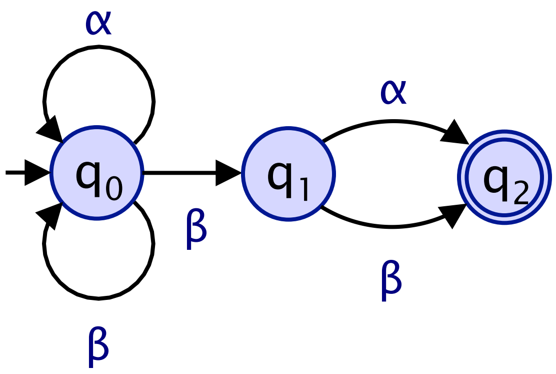
- Determinisation of an NFA $A = (Q, \Sigma, \delta, Q_0, F)$
  - i.e. removal of choice in each automata state

- Equivalent DFA is $A_{det} = (2^Q, \Sigma, \delta_{det}, q_0, F_{det})$ where:
  - $\delta_{det}(Q', \alpha) = \bigcup_{q \in Q'} \delta(q, \alpha)$
  - $F_{det} = \{ Q' \subseteq Q \mid Q' \cap F \neq \emptyset \}$

- Note exponential blow-up in size…

Example

NFA A

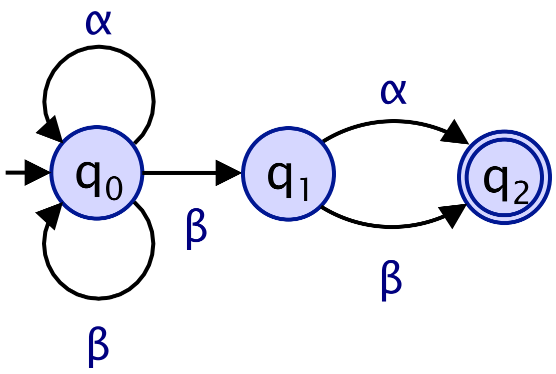


regexp:

$(\alpha+\beta)^*\beta(\alpha+\beta)$

Example

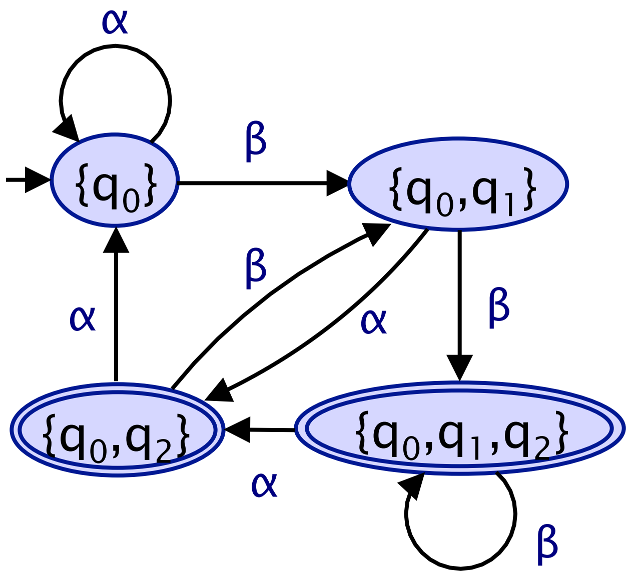
NFA A



regexp:

$(\alpha+\beta)^*\beta(\alpha+\beta)$

DFA A_{det}



Other properties of NFA/DFA

- NFA/DFA have the same expressive power
  - but NFA can be more efficient (up to exponentially smaller)

- NFA/DFA are closed under complementation
  - build total DFA, swap accept/non-accept states

- For any regular language L, there is a unique minimal DFA that accepts L (up to isomorphism)
  - efficient algorithm to minimise DFA into equivalent DFA
  - partition refinement algorithm (like for bisimulation)

- Language emptiness of an NFA reduces to reachability
  - $L(A) \neq \varnothing$ iff can reach a state in F from an initial state in Q_0

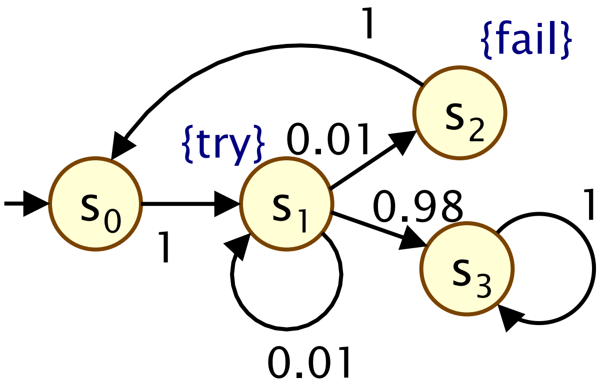
Languages as properties

- Consider a model, i.e. an LTS/DTMC/MDP/…
  - e.g. DTMC $D = (S, s_{init}, \mathbf{P}, Lab)$
  - where labelling Lab uses atomic propositions from set AP
  - let $\omega \in Path(s)$ be some infinite path

- Temporal logic properties
  - for some temporal logic (path) formula ψ, does $\omega \vDash \psi$?

- Traces and languages
  - $trace(\omega) \in (2^{AP})^\omega$ denotes the projection of state labels of ω
  - i.e. $trace(s_0 s_1 s_2 s_3 \ldots) = Lab(s_0)Lab(s_1)Lab(s_2)Lab(s_3)\ldots$
  - for some language $L \subseteq (2^{AP})^\omega$, is $trace(\omega) \in L$?

Example



- Atomic propositions
  - AP = { fail, try }
  - 2^{AP} = { ∅, {fail}, {try}, {fail,try} }
- Paths and traces
  - e.g. $\omega = s_0 s_1 s_1 s_2 s_0 s_1 s_2 s_0 s_1 s_3 s_3 s_3 \ldots$
  - trace(ω) = ∅ {try} {try} {fail} ∅ {try} {fail} ∅ {try} ∅ ∅ ∅ …
- Languages
  - e.g. "no failures"
  - $L = \{ \alpha_1 \alpha_2 \ldots \in (2^{AP})^\omega \mid \alpha_i$ is ∅ or {try} for all i }

Regular safety properties

- A safety property P is a language over 2^{AP} such that
  - for any word w that violates P (i.e. is not in the language), w has a prefix w', all extensions of which, also violate P

- A regular safety property is
  - safety property for which the set of "bad prefixes" (finite violations) forms a regular language

- Formally…
  - $P \subseteq (2^{AP})^\omega$ is a safety property if:
    - $\forall\ w \in ((2^{AP})^\omega \backslash P)\ .\ \exists$ finite prefix w' of w such that:
    - $P \cap \{\ w'' \in (2^{AP})^\omega \mid w' \text{ is a prefix of } w''\ \} = \varnothing$
  - P is a regular safety property if:
    - $\{\ w' \in (2^{AP})^* \mid \forall\ w'' \in (2^{AP})^\omega\ .\ w'.w'' \notin P\ \}$ is regular

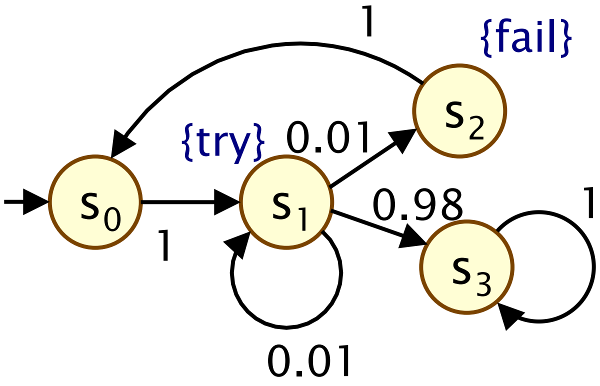
Regular safety properties

- A safety property P is a language over 2^{AP} such that
  - for any word w that violates P (i.e. is not in the language), w has a prefix w', all extensions of which, also violate P
- A regular safety property is
  - safety property for which the set of "bad prefixes" (finite violations) forms a regular language
- Examples:
  - "at least one traffic light is always on"
  - "two traffic lights are never on simultaneously"
  - "a red light is always preceded immediately by an amber light"

Example

- Regular safety property:
  - "at most 2 failures occur"
  - language over:

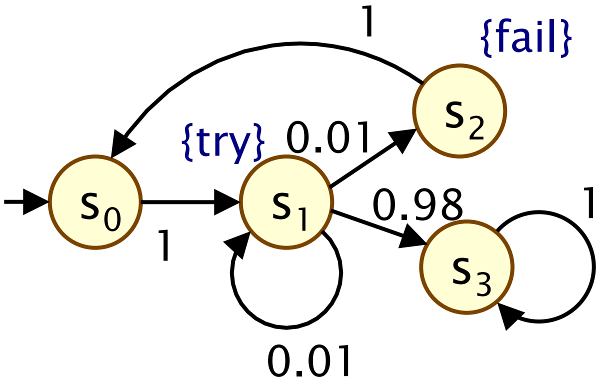
    $2^{AP} = \{ \varnothing, \{fail\}, \{try\}, \{fail,try\} \}$



Example

- Regular safety property:
  - "at most 2 failures occur"
  - language over:

    $2^{AP} = \{ \varnothing, \{fail\}, \{try\}, \{fail,try\} \}$

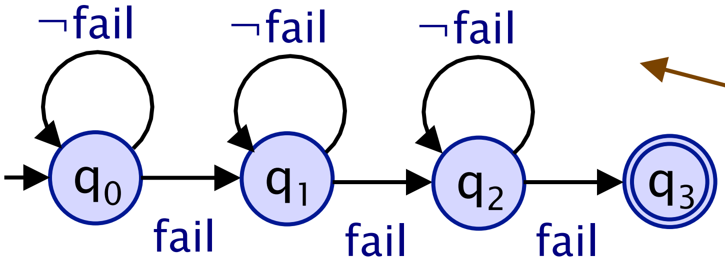


- Bad prefixes (regexp):

  (¬fail)*.fail.(¬fail)*.fail.(¬fail)*.fail

fail denotes:
({fail} + {fail,try})
¬fail denotes:
(∅ + {try})

- Bad prefixes (DFA):



fail denotes:
{fail}, {fail,try}
¬fail denotes:
∅, {try}

Regular safety properties + DTMCs

- Consider a DTMC D (with atomic propositions from AP) and a regular safety property $P \subseteq (2^{AP})^\omega$

- Let $Prob^D(s, P)$ denote the probability of P being satisfied
  - i.e. $Prob^D(s, P) = Pr^D_s\{ \omega \in Path(s) \mid trace(\omega) \in P \}$
  - where Pr^D_s is the probability measure over Path(s) for D
  - this set is always measurable (see later)

- Example (safety) specifications
  - "the probability that at most 2 failures occur is $\geq$0.999"
  - "what is the probability that at most 2 failures occur?"

- How to compute $Prob^D(s, P)$?

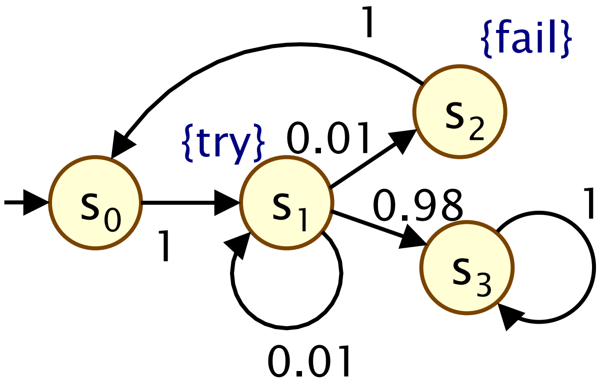
Product DTMC

- We construct the product of
  - a DTMC $D = (S, s_{init}, P, L)$
  - and a (total) DFA $A = (Q, \Sigma, \delta, q_0, F)$
  - intuitively: records state of A for path fragments of D

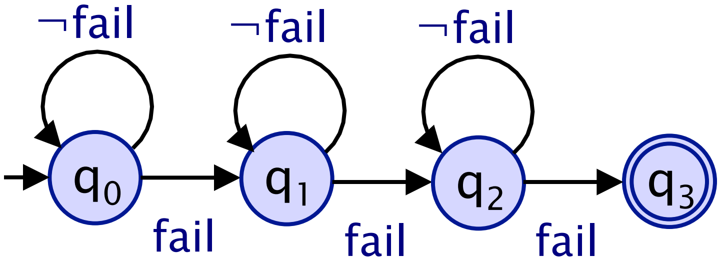
- The product DTMC $D \otimes A$ is:
  - the DTMC $(S \times Q, (s_{init}, q_{init}), P', L')$ where:
  - $q_{init} = \delta(q_0, L(s_{init}))$
  - $\mathbf{P}'((s_1,q_1),(s_2,q_2)) = \begin{cases} \mathbf{P}(s_1,s_2) & \text{if } q_2 = \delta(q_1, L(s_2)) \\ 0 & \text{otherwise} \end{cases}$
  - $L'(s,q) = \{ \text{accept} \}$ if $q \in F$ and $L'(s,q) = \varnothing$ otherwise

Example

DTMC D



DFA A

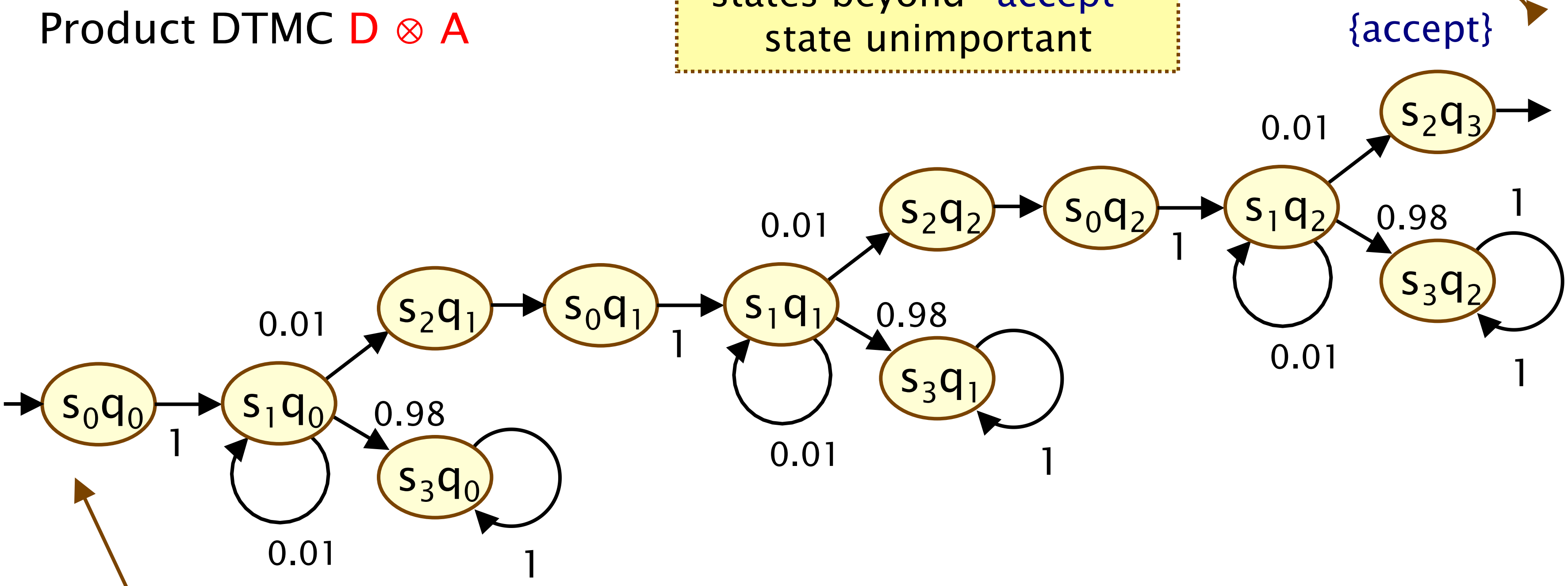


fail denotes:
{fail}, {fail,try}
¬fail denotes:
∅, {try}

Example

Product DTMC $D \otimes A$

states beyond "accept" state unimportant



$s_0,\delta(q_0,L(s_0))$

Product DTMC

- One interpretation of $D \otimes A$:
  - unfolding of D where q for each state (s,q) records state of automata A for path fragment so far

- In fact, since A is deterministic…
  - for any $\omega \in Path(s)$ of the DTMC D:
    - there is a unique run in A for trace(ω)
    - and a corresponding (unique) path through $D \otimes A$
  - for any path $\omega' \in Path^{D\otimes A}(s,q_{init})$ where $q_{init} = \delta(q_0,L(s))$
    - there is a corresponding path in D and a run in A

- DFA has no effect on probabilities
  - i.e. probabilities preserved in product DTMC

Regular safety properties + DTMCs

- Regular safety property $P \subseteq (2^{AP})^{\omega}$
  - "bad prefixes" (finite violations) represented by DFA A

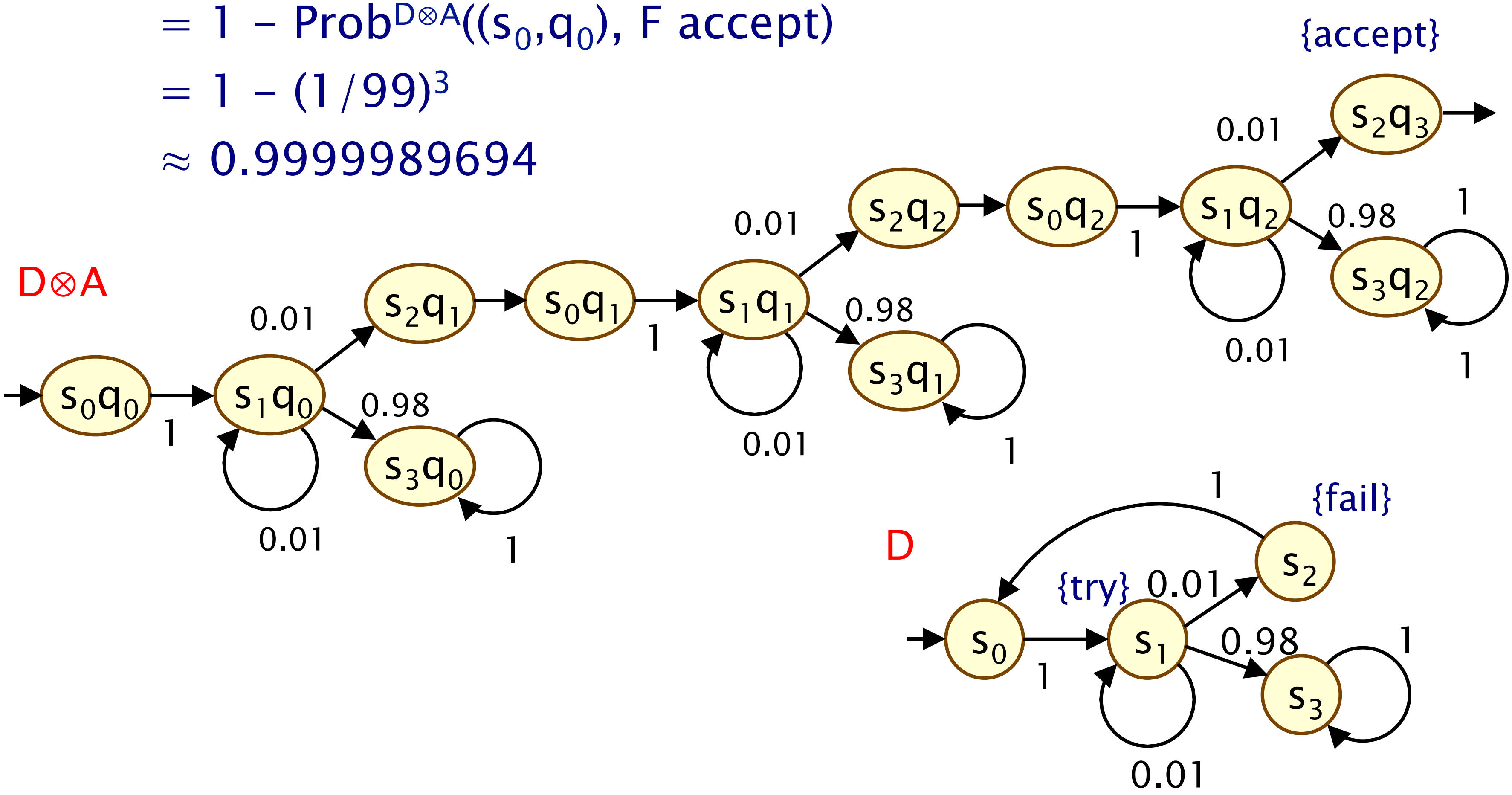
- Probability of P being satisfied in state s of D
  - $\text{Prob}^D(s, P) = \text{Pr}^D_s\{ \omega \in \text{Path}(s) \mid \text{trace}(\omega) \in P \}$
    $= 1 - \text{Pr}^D_s\{ \omega \in \text{Path}(s) \mid \text{trace}(\omega) \notin P \}$
    $= 1 - \text{Pr}^D_s\{ \omega \in \text{Path}(s) \mid \text{pref}(\text{trace}(\omega)) \cap L(A) \neq \varnothing \}$
  - where pref(w) = set of all finite prefixes of infinite word w

$$\boxed{\text{Prob}^D(s, P) = 1 - \text{Prob}^{D \otimes A}((s,q_s), \text{F accept})}$$

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

Example

- $\text{Prob}^{D}(s_0, \text{“at most 2 failures occur”})$
  $= 1 - \text{Prob}^{D\otimes A}((s_0,q_0), \text{F accept})$
  $= 1 - (1/99)^3$
  ≈ 0.9999989694



Summing up…

- Nondeterministic finite automata (NFA)
  - can represent any regular language, regular expression
  - closed under complementation, intersection, …
  - (non-)emptiness reduces to reachability
- Deterministic finite automata (DFA)
  - can be constructed from NFA through determinisation
  - equally expressive as NFA, but may be larger
- Regular safety properties
  - language representing set of possible traces
  - bad (violating) prefixes form a regular language
- Probability of a regular safety property on a DTMC
  - construct product DTMC
  - reduces to probabilistic reachability

Probabilistic Model Checking Michaelmas Term 2011

Lecture 17
ω-regular properties

Dr. Dave Parker

Department of Computer Science
University of Oxford

Long-run properties

- Last lecture: regular safety properties
  - e.g. “a message failure never occurs”
  - e.g. “an alarm is only ever triggered by an error”
  - bad prefixes represented by a regular language
  - property always refuted by a finite trace/path
- Liveness properties
  - e.g. "for every request, an acknowledge eventually follows”
  - no finite prefix refutes the property
  - any finite prefix can be extended to a satisfying trace
- Fairness assumptions
  - e.g. “every process that is enabled infinitely often is scheduled infinitely often”
- Need properties of infinite paths

Overview

- ω–regular expressions and ω–regular languages
- Nondeterministic Büchi automata (NBA)
- Deterministic Büchi automata (DBA)
- Deterministic Rabin automata (DRA)
- Deterministic ω–automata and DTMCs

ω−regular expressions

- Regular expressions E over alphabet Σ are given by:
  - $E ::= \varnothing \mid \varepsilon \mid \alpha \mid E + E \mid E.E \mid E^*$ (where $\alpha \in \Sigma$)

- An ω−regular expression takes the form:
  - $G = E_1.(F_1)^\omega + E_2.(F_2)^\omega + \ldots + E_n.(F_n)^\omega$
  - where E_i and F_i are regular expressions with $\varepsilon \notin L(F_i)$

- The language $L(G) \subseteq \Sigma^\omega$ of an ω−regular expression G
  - is $L(E_1).L(F_1)^\omega \cup L(E_2).L(F_2)^\omega + \ldots + L(E_n).L(F_n)^\omega$
  - where L(E) is the language of regular expression E
  - and $L(E)^\omega = \{ w^\omega \mid w \in L(E) \}$

- Example: $(\alpha+\beta+\gamma)^*(\beta+\gamma)^\omega$ for $\Sigma = \{ \alpha, \beta, \gamma \}$

ω-regular languages/properties

- A language $L \subseteq \Sigma^\omega$ over alphabet Σ is an ω-regular language if and only if:
  - $L = L(G)$ for some ω-regular expression G

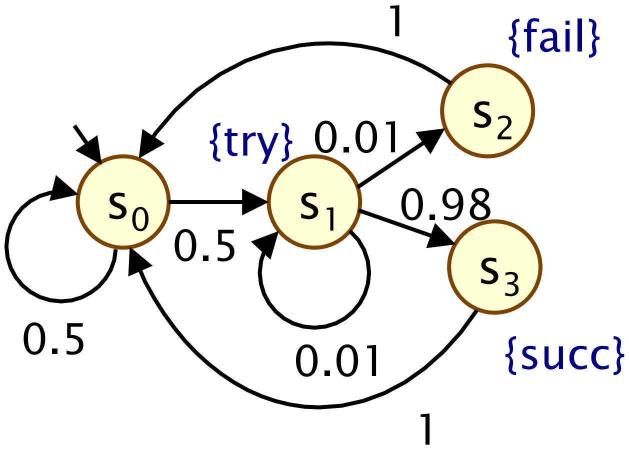
- ω-regular languages are:
  - closed under intersection
  - closed under complementation

- $P \subseteq (2^{AP})^\omega$ is an ω-regular property
  - if P is an ω-regular language over 2^{AP}
  - (where AP is the set of atomic propositions for some model)
  - path ω satisfies P if $trace(\omega) \in P$
  - NB: any regular safety property is an ω-regular property

Examples

- A message is sent successfully infinitely often
  - $((\neg\text{succ})^*.\text{succ})^\omega$

- Every time the process tries to send a message, it eventually succeeds in sending it
  - $((\neg\text{try})^* + \text{try}.(\neg\text{succ})^*.\text{succ})^\omega$



Büchi automata

- A nondeterministic Büchi automaton (NBA) is…

  - a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where:

  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
  - $Q_0 \subseteq Q$ is a set of initial states
  - $F \subseteq Q$ is a set of “accept” states

  - i.e. just like a nondeterministic finite automaton (NFA)

- The difference is the accepting condition…

Language of an NBA

- Consider a Büchi automaton $A = (Q, \Sigma, \delta, Q_0, F)$

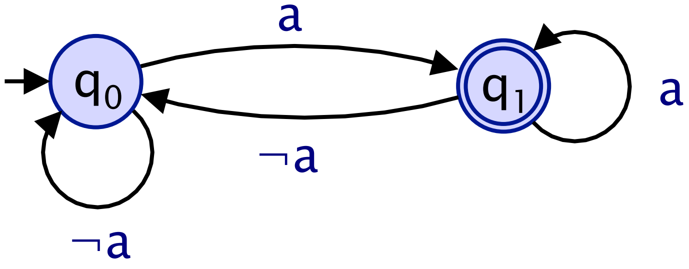
- A run of A on an infinite word $\alpha_1\alpha_2\ldots$ is:
  - an infinite sequence of automata states $q_0q_1\ldots$ such that:
  - $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \alpha_{i+1})$ for all $i \geq 0$

- An accepting run is a run with $q_i \in F$ for infinitely many i

- The language L(A) of A is the set of all infinite words on which there exists an accepting run of A

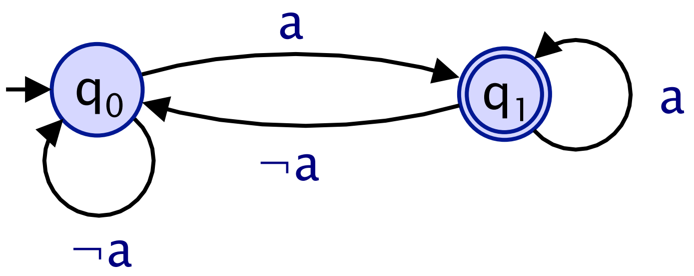
Example

- Infinitely often a

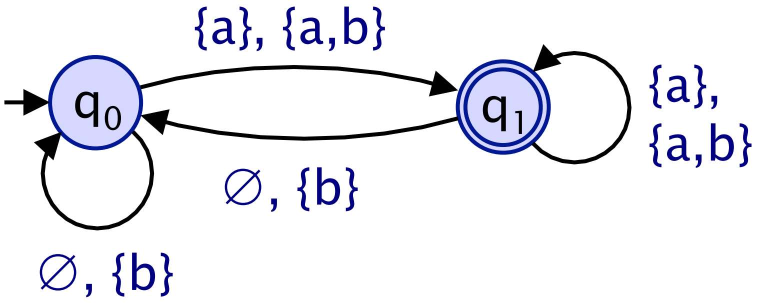


Example…

- As in the last lecture, we use automata to represent languages of the form $L \subseteq (2^{AP})^\omega$
- So, if AP = {a,b}, then:



- …is actually:



Properties of Büchi automata

- ω-regular languages
  - L(A) is an ω-regular language for any NBA A
  - any ω-regular language can be represented by an NBA

- ω-regular expressions
  - like for finite automata, can construct an NBA from an arbitrary ω-regular expression $E_1.(F_1)^\omega + \ldots + E_n.(F_n)^\omega$
  - i.e. there are operations on NBAs to:
    - construct NBA accepting L^ω for regular language L
    - construct NBA from NFA for (regular) E and NBA for (ω-regular) F
    - construct NBA accepting union $L(A_1) \cup L(A_2)$ for NBA A_1 and A_2

Büchi automata and LTL

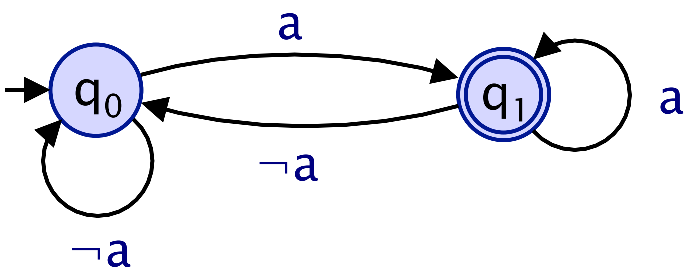
- LTL formulae
  - $\psi ::= \text{true} \mid a \mid \psi \wedge \psi \mid \neg\psi \mid X\,\psi \mid \psi\, U\, \psi$
  - where $a \in AP$ is an atomic proposition

- Can convert any LTL formula ψ into an NBA A over 2^{AP}
  - i.e. $\omega \vDash \psi \Leftrightarrow \text{trace}(\omega) \in L(A)$ for any path ω

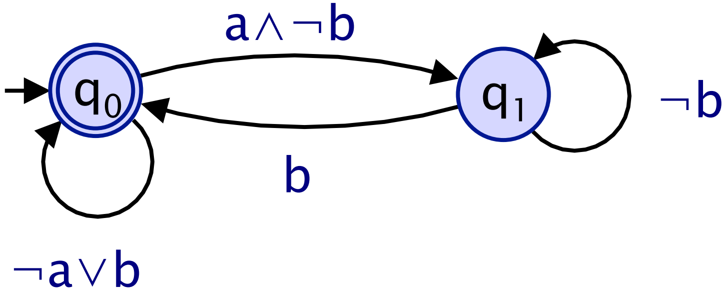
- LTL-to-NBA translation (see e.g. [VW94], [DGV99])
  - construct a generalized NBA (multiple sets of accept states)
  - based on decomposition of LTL formula into subformulae
  - can convert GNBA into an equivalent NBA
  - various optimisations to the basic techniques developed
  - not covered here; see e.g. section 5.2 of [BK08]

Büchi automata and LTL

- GF a (“infinitely often a”)



- G(a → F b) (“b always eventually follows a”)



Deterministic Büchi automata

- Like for finite automata…

- A NBA is deterministic if:
  - $|Q_0|=1$
  - $|\delta(q, \alpha)| \leq 1$ for all $q \in Q$ and $\alpha \in \Sigma$
  - i.e. one initial state and no nondeterministic successors

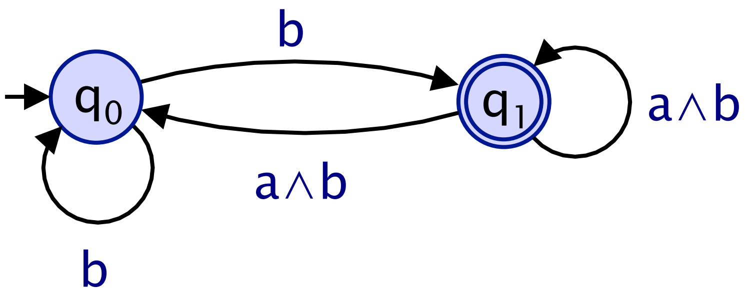
- A deterministic Büchi automaton (DBA) is total if:
  - $|\delta(q, \alpha)| = 1$ for all $q \in Q$ and $\alpha \in \Sigma$
  - i.e. unique successor states

- But, NBA can not always be determinised…
  - i.e. NBA are strictly more expressive than DBA

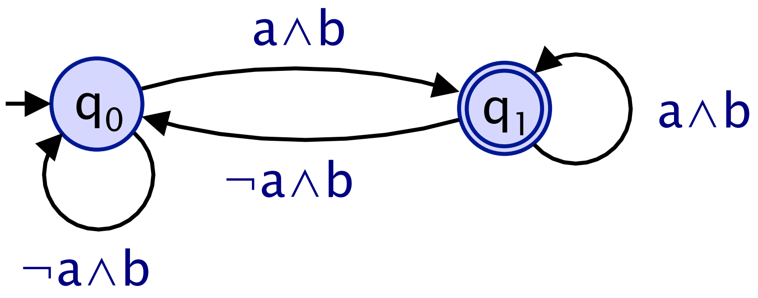
NBA and DBA

- NBA and DBA for the LTL formula G b ∧ GF a

NBA:



DBA:

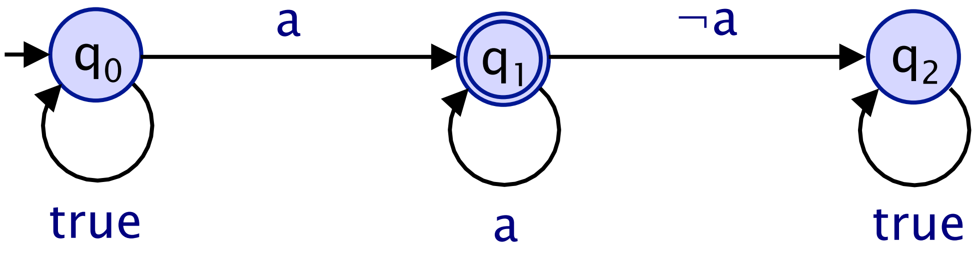


No DBA possible

- Consider the ω-regular expression $(\alpha+\beta)^*\alpha^\omega$ over $\Sigma=\{\alpha,\beta\}$
  - i.e. words containing only finitely many instances of β
  - there is no deterministic Büchi automata accepting this

- In particular, take $\alpha = \{a\}$ and $\beta = \varnothing$, i.e. $\Sigma=2^{AP}$, $AP=\{a\}$
  - $(\alpha+\beta)^*\alpha^\omega$ represents the LTL formula FG a

- FG a is represented by the following NBA:



- But there is no DBA for FG a

Deterministic Rabin automata

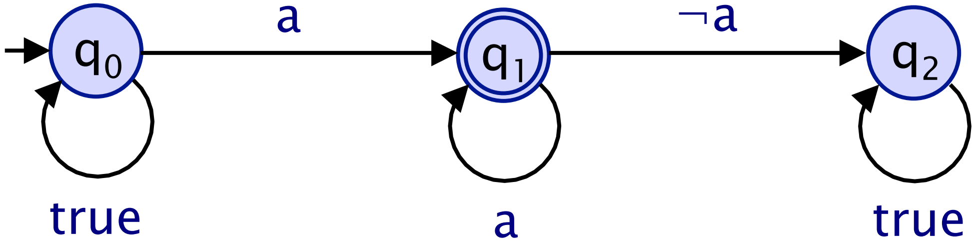
- A deterministic Rabin automaton (DRA) is…
  - a tuple $A = (Q, \Sigma, \delta, q_0, Acc)$ where:
  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \rightarrow Q$ is a transition function
  - $q_0 \in Q$ is an initial state
  - $Acc \subseteq 2^Q \times 2^Q$ is an acceptance condition
- The acceptance condition is a set of pairs of state sets
  - $Acc = \{ (L_i, K_i) \mid 1 \leq i \leq k \}$

Deterministic Rabin automata

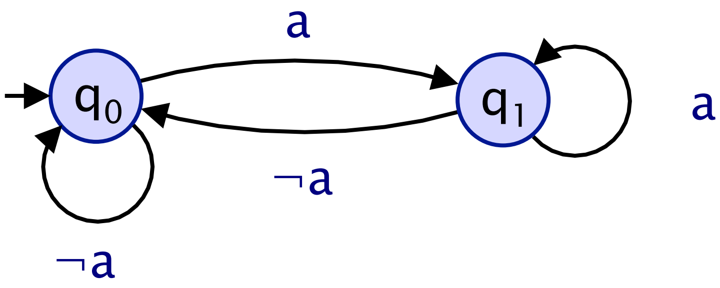
- A run of a word on a DRA is accepting iff:
  - for some pair (L_i, K_i), the states in L_i are visited finitely often and (some of) the states in K_i are visited infinitely often
  - or in LTL: $\bigvee_{1 \le i \le k} (FG\, \neg L_i \wedge GF\, K_i)$
- Hence:
  - a deterministic Büchi automaton is a special case of a deterministic Rabin automaton where $Acc = \{ (\emptyset, \{F\}) \}$

FG a

- NBA for FG a (no DBA exists)



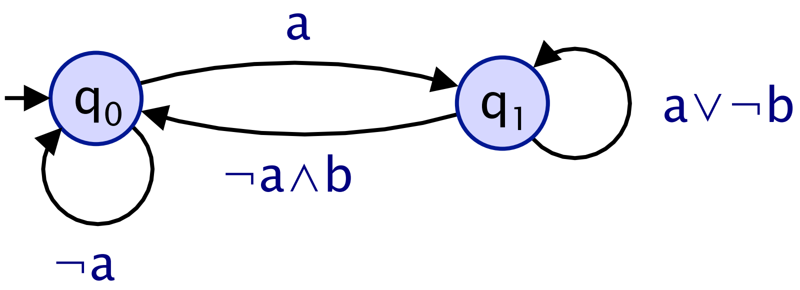
- DRA for FG a



  - where acceptance condition is Acc = { ({q_0},{q_1}) }

Example - DRA

- Another example of a DRA (over alphabet $2^{\{a,b\}}$)



  - where acceptance condition is Acc = { ({q_1},{q_0}) }

- In LTL: G(a → F(¬a∧b)) ∧ FG ¬a

Properties of DRA

- Any ω-regular language can represented by a DRA
  - (and L(A) is an ω-regular language for any DRA A)

- i.e. DRA and NBA are equally expressive
  - (but NBA may be more compact)
  - and DRA are strictly more expressive than DBA

- Any NBA can be converted to an equivalent DRA [Saf88]
  - size of the resulting DRA is $2^{O(n\log n)}$

Deterministic ω-automata and DTMCs

- Let A be a DBA or DRA over the alphabet 2^{AP}
  - i.e. $L(A) \subseteq (2^{AP})^\omega$ identifies a set of paths in a DTMC

- Let $Prob^D(s, A)$ denote the corresponding probability
  - from state s in a discrete-time Markov chain D
  - i.e. $Prob^D(s, A) = Pr^D_s\{ \omega \in Path(s) \mid trace(\omega) \in L(A) \}$

- Like for finite automata (i.e. DFA), we can evaluate $Prob^D(s, A)$ by constructing a product of D and A
  - which records the state of both the DTMC and the automaton

Product DTMC for a DBA

- For a DTMC $D = (S, s_{init}, \mathbf{P}, L)$
- and a (total) DBA $A = (Q, \Sigma, \delta, q_0, F)$

- The product DTMC $D \otimes A$ is:
  - the DTMC $(S \times Q, (s_{init}, q_{init}), \mathbf{P'}, L')$ where:

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

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

$$L'(s,q) = \{ \text{accept} \} \text{ if } q \in F \text{ and } L'(s,q) = \varnothing \text{ otherwise}$$

- Since A is deterministic
  - unique mappings between paths of D, A and $D \otimes A$
  - probabilities of paths are preserved

Product DTMC for a DBA

- For DTMC D and DBA A

$$\boxed{\mathrm{Prob}^{D}(s,\ A) = \mathrm{Prob}^{D\otimes A}((s,q_s),\ \mathrm{GF\ accept})}$$

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

- Hence:

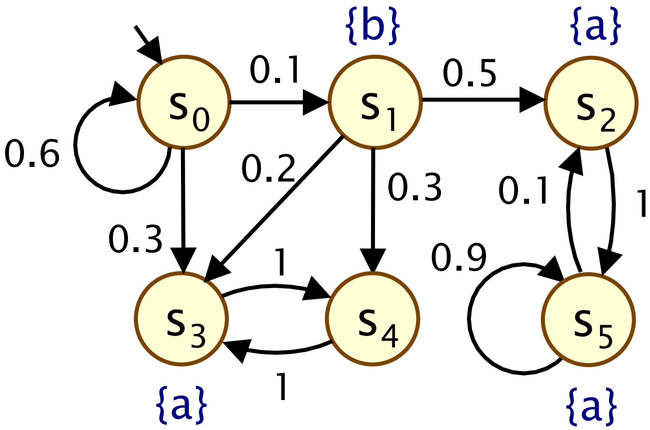
$$\boxed{\mathrm{Prob}^{D}(s,\ A) = \mathrm{Prob}^{D\otimes A}((s,q_s),\ \mathrm{F}\ T_{\mathrm{GFaccept}})}$$

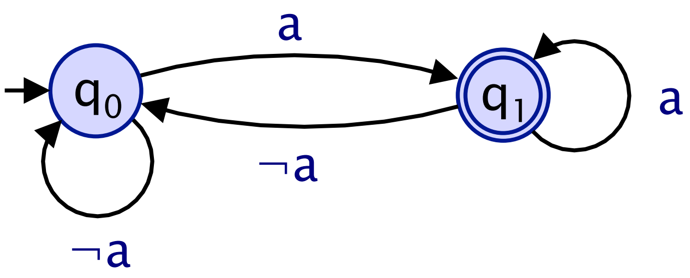
  - where T_{GFaccept} = union of $D\otimes A$ BSCCs T with $T \cap \mathrm{Sat}(accept) \neq \varnothing$

- Reduces to computing BSCCs and reachability probabilities

Example

- Compute Prob(s_0, GF a)
  - property can be represented as a DBA

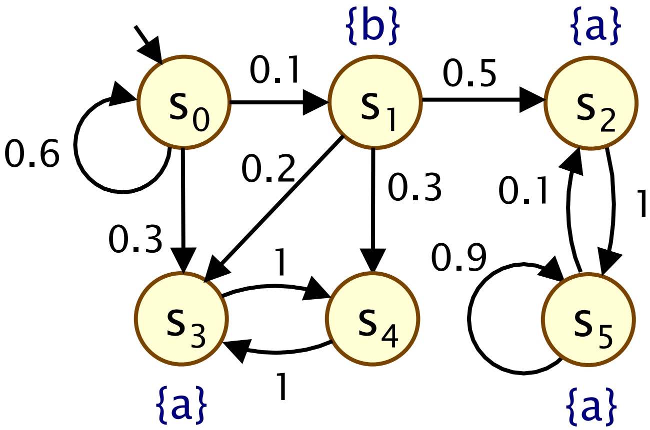


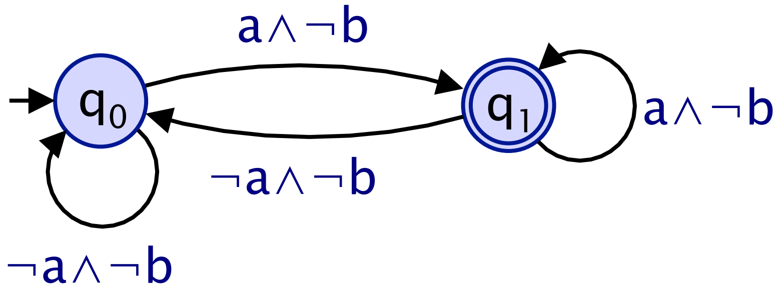


- Result: 1

Example 2

- Compute Prob(s_0, G ¬b ∧ GF a)
  - property can be represented as a DBA





- Result: 0.75

Product DTMC for a DRA

- For a DTMC $D = (S, s_{init}, \mathbf{P}, L)$
- and a (total) DRA $A = (Q, \Sigma, \delta, q_0, Acc)$
  - where $Acc = \{ (L_i, K_i) \mid 1 \leq i \leq k \}$

- The product DTMC $D \otimes A$ is:
  - the DTMC $(S \times Q, (s_{init}, q_{init}), \mathbf{P}', L')$ where:

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

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

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

    (i.e. state sets of acceptance condition used as labels)

- (same product as for DBA, except for state labelling)

Product DTMC for a DRA

- For DTMC D and DRA A

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

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

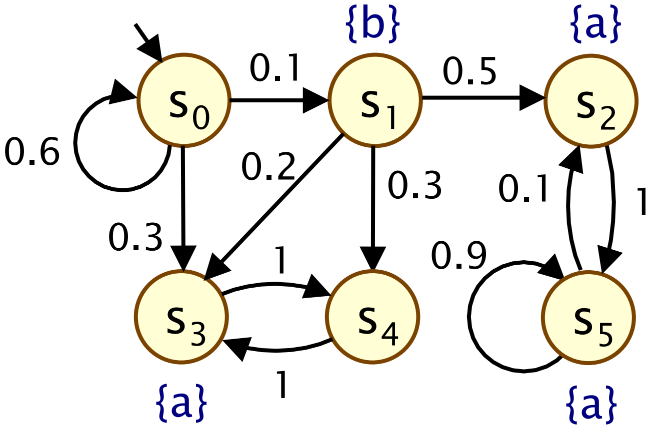
$$\boxed{\text{Prob}^{D}(s, A) = \text{Prob}^{D\otimes A}((s,q_s), F\ T_{Acc})}$$

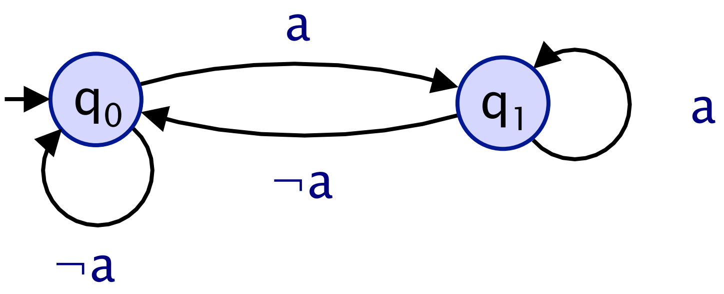
  - where T_{Acc} is the union of all accepting BSCCs in $D\otimes A$
  - an accepting BSCC T of $D\otimes A$ is such that, for some $1\leq i\leq k$:
    - $q \vDash \neg l_i$ for all $(s,q) \in T$ and $q \vDash k_i$ for some $(s,q) \in T$
    - i.e. $T \cap (S \times L_i) = \varnothing$ and $T \cap (S \times K_i) \neq \varnothing$

- Reduces to computing BSCCs and reachability probabilities

Example 3

- Compute Prob(s_0, FG a)
  - property can be represented as a DRA



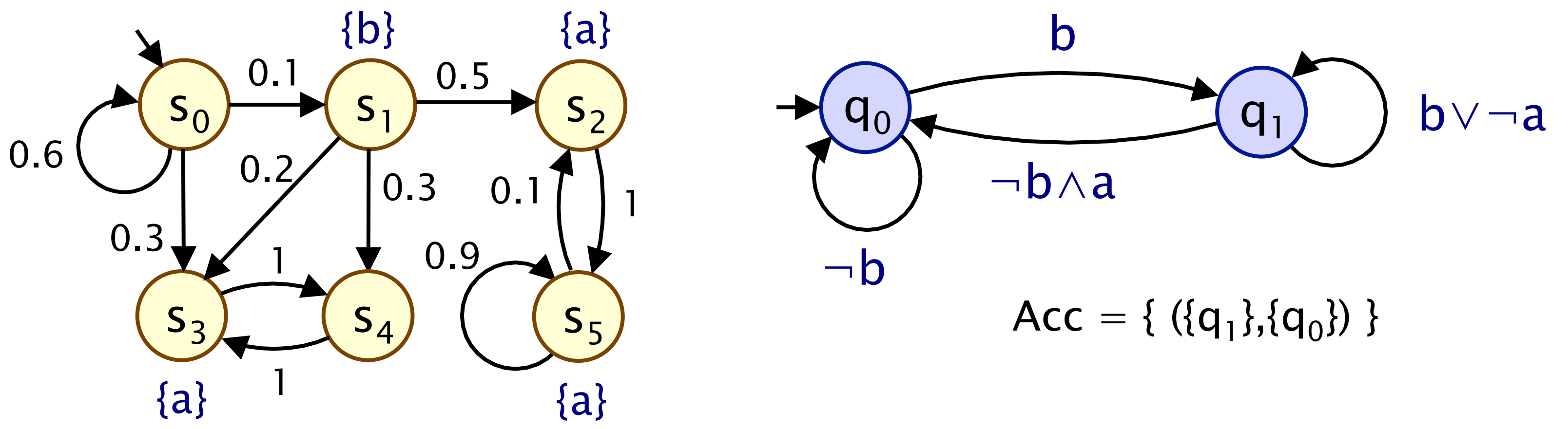


Acc = { ({q_0},{q_1}) }

- Result: 0.125

Example 4

- Compute Prob(s_0, G(b → F(¬b∧a)) ∧ FG ¬b)
  - property can be represented as a DRA



- Result: 1

Summing up…

- ω-regular expressions and ω-regular languages
  - languages of infinite words: $E_1.(F_1)^\omega + E_2.(F_2)^\omega + \ldots + E_n.(F_n)^\omega$
- Nondeterministic Büchi automata (NBA)
  - accepting runs visit a state in F infinitely often
  - can represent any ω-regular language by an NBA
  - can translate any LTL formula into equivalent NBA
- Deterministic Büchi automata (DBA)
  - strictly less expressive than NBA (e.g. no NBA for FG a)
- Deterministic Rabin automata (DRA)
  - generalised acceptance condition: $\{ (L_i, K_i) \mid 1 \leq i \leq k \}$
  - as expressive as NBA; can convert any NBA to a DRA
- Deterministic ω-automata and DTMCs
  - product DTMC + BSCC computation + reachability

Lecture 18
LTL model checking for DTMCs and MDPs

Dr. Dave Parker

Department of Computer Science
University of Oxford

Overview

- Recall
  - deterministic ω-automata (DBA or DRA) and DTMCs

- LTL model checking for DTMCs
  - measurability
  - complexity
  - PCTL* model checking for DTMCs

- LTL model checking for MDPs

Recall – DBA and DRA

- Deterministic Büchi automata (DBA)
  - $(Q, \Sigma, \delta, q_0, F)$
  - accepting run must visit some state in F infinitely often
  - less expressive than nondeterministic Büchi automata (NBA)

- Deterministic Rabin automata (DRA)
  - $(Q, \Sigma, \delta, q_0, Acc)$
  - $Acc = \{ (L_i, K_i) \mid 1 \leq i \leq k \}$
  - for some pair (L_i, K_i), the states in L_i must be visited finitely often and (some of) the states in K_i visited infinitely often
  - equally expressive as NBA
  - (i.e. all ω-regular properties; and hence all LTL formulae)

Product DTMC for a DBA

- For DTMC D and DBA A

$$\boxed{\mathrm{Prob}^{D}(s, A) = \mathrm{Prob}^{D\otimes A}((s,q_s), \mathrm{GF}\ \mathit{accept})}$$

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

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

  - where $T_{GFaccept}$ is the union of all BSCCs T in $D\otimes A$ with $T\cap Sat(accept)\neq\varnothing$

- Reduces to computing BSCCs and reachability probabilities

Product DTMC for a DRA

- For DTMC D and DRA A

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

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

- Hence:

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

  - where T_{Acc} is the union of all accepting BSCCs in $D\otimes A$
  - an accepting BSCC T of $D\otimes A$ is such that, for some $1\leq i\leq k$:
    - $q \models \neg l_i$ for all $(s,q) \in T$ and $q \models k_i$ for some $(s,q) \in T$
    - i.e. $T \cap (S \times L_i) = \varnothing$ and $T \cap (S \times K_i) \neq \varnothing$

- Reduces to computing BSCCs and reachability probabilities

LTL model checking for DTMCs

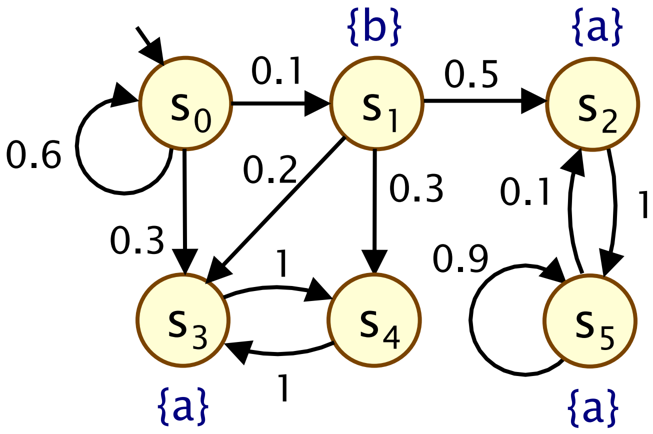
- Model check LTL specification $P_{\sim p}$ [ψ] against DTMC D

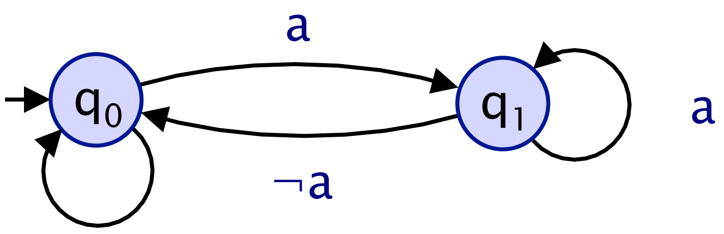
- 1. Generate a deterministic Rabin automaton (DRA) for ψ
  - build nondeterministic Büchi automaton (NBA) for ψ [VW94]
  - convert the NBA to a DRA [Saf88]
- 2. Construct product DTMC D⊗A
- 3. Identify accepting BSCCs of D⊗A
- 4. Compute probability of reaching accepting BSCCs
  - from all states of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

- Qualitative LTL model checking – no probabilities needed

Example 3 (Lec 17) revisited

- Model check $P_{>0.2}$ [FG a]





Acc = $\{ (\{q_0\},\{q_1\}) \}$

- Result:
  - Prob(FG a) = [0.125, 0.5, 1, 0, 0, 1]
  - Sat($P_{>0.2}$ [FG a]) = $\{ s_1, s_2, s_5 \}$

Measurability of ω-regular properties

- For any ω-regular property ψ
  - the set of ψ-satisfying paths in any DTMC D is measurable
- Hence, the same applies to
  - any regular safety property
  - any LTL formula

- Proof sketch
  - any ω-regular property can be represented by a DRA A
  - we can construct D⊗A, in which there is a direct mapping from any path ω in D to a path ω' in D⊗A
  - $\omega \vDash \psi$ iff $\omega' \vDash \bigvee_{1 \leq i \leq k} (\mathrm{FG}\ \neg l_i \ \wedge\ \mathrm{GF}\ k_i)$
  - GF Φ and FG Φ are measurable (see lecture 3)
  - ∧ and ∨ = intersection/union (which preserve measurability)

Complexity

- Complexity of model checking LTL formula ψ on DTMC D
  - is doubly exponential in |ψ| and polynomial in |D|
  - (for the algorithm presented in these lectures)
- Converting LTL formula ψ to DRA A
  - for some LTL formulae of size n, size of smallest DRA is 2^{2^n}
- BSCC computation
  - Tarjan algorithm – linear in model size (states/transitions)
- Probabilistic reachability
  - linear equations – cubic in (product) model size
- In total: O(poly(|D|,|A|))
- In practice: |ψ| is small and |D| is large
- Complexity can be reduced to single exponential in |ψ|
  - see e.g. [CY88,CY95]

PCTL* model checking

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

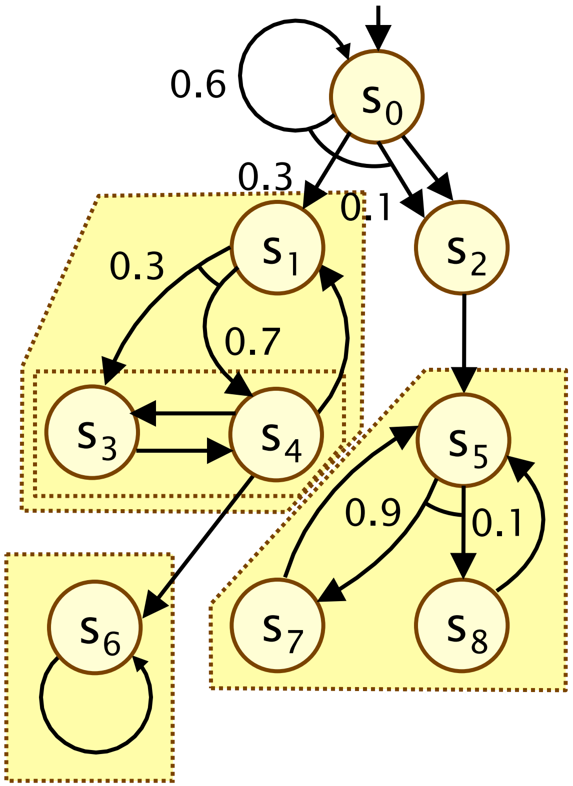
- PCTL* model checking algorithm
  - bottom-up traversal of parse tree for formula (like PCTL)
  - to model check $P_{\sim p}$ [ψ]:
    - replace maximal state subformulae with atomic propositions
    - (state subformulae already model checked recursively)
    - modified formula ψ is now an LTL formula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

Recall – end components in MDPs

- End components of MDPs are the analogue of BSCCs in DTMCs

- An end component is a strongly connected sub-MDP

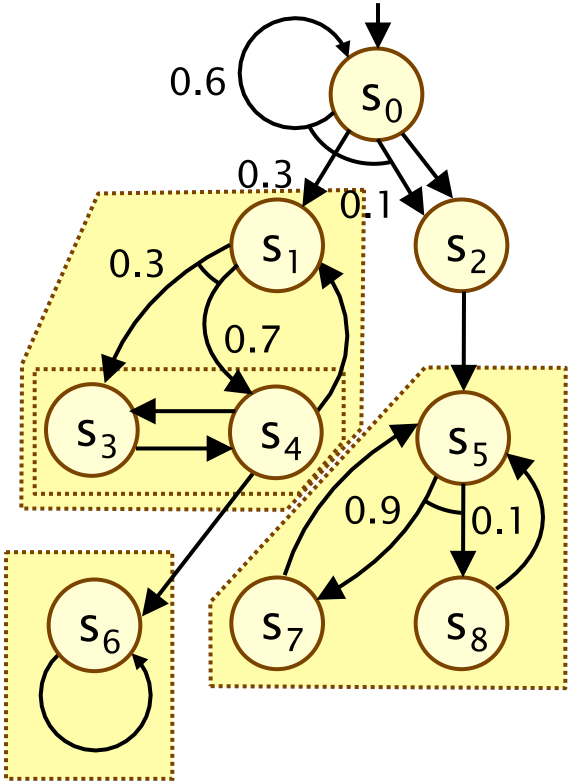
- A sub-MDP comprises a subset of states and a subset of the actions/distributions available in those states, which is closed under probabilistic branching



Note:
- action labels omitted
- probabilities omitted where =1

Recall – end components in MDPs

- End components of MDPs are the analogue of BSCCs in DTMCs
- For every end component, there is an adversary which, with probability 1, forces the MDP to remain in the end component, and visit all its states infinitely often
- Under every adversary σ, with probability 1, the set of states visited infinitely often forms an end component



Recall – long-run properties of MDPs

- Maximum probabilities
  - $p_{max}(s, GF\ a) = p_{max}(s, F\ T_{GFa})$
    - where T_{GFa} is the union of sets T for all end components (T,**Steps'**) with $T \cap Sat(a) \neq \varnothing$
  - $p_{max}(s, FG\ a) = p_{max}(s, F\ T_{FGa})$
    - where T_{FGa} is the union of sets T for all end components (T,**Steps'**) with $T \subseteq Sat(a)$
- Minimum probabilities
  - need to compute from maximum probabilities…
  - $p_{min}(s, GF\ a) = 1 - p_{max}(s, FG\neg a)$
  - $p_{min}(s, FG\ a) = 1 - p_{max}(s, GF\neg a)$

Automata-based properties for MDPs

- For an MDP M and automaton A over alphabet 2^{AP}
  - consider probability of "satisfying" language $L(A) \subseteq (2^{AP})^\omega$
  - $Prob^{M,\sigma}(s, A) = Pr_s^{M,\sigma}\{ \omega \in Path^{M,\sigma}(s) \mid trace(\omega) \in L(A) \}$
  - $p_{max}^M(s, A) = \sup_{\sigma\in Adv} Prob^{M,\sigma}(s, A)$
  - $p_{min}^M(s, A) = \inf_{\sigma\in Adv} Prob^{M,\sigma}(s, A)$

- Might need minimum or maximum probabilities
  - e.g. $s \vDash P_{\geq 0.99} [\ \psi_{good}\ ] \Leftrightarrow p_{min}^M (s, \psi_{good}) \geq 0.99$
  - e.g. $s \vDash P_{\leq 0.05} [\ \psi_{bad}\ ] \Leftrightarrow p_{max}^M (s, \psi_{bad}) \leq 0.05$
- But, ψ-regular properties are closed under negation
  - as are the automata that represent them
  - so can always consider maximum probabilities…
  - $p_{max}^M(s, \psi_{bad})$ or $1 - p_{max}^M(s, \neg\psi_{good})$

LTL model checking for MDPs

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

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

Product MDP for a DRA

- For a MDP $M = (S, s_{init}, \mathbf{Steps}, L)$
- and a (total) DRA $A = (Q, \Sigma, \delta, q_0, Acc)$
  - where $Acc = \{ (L_i, K_i) \mid 1 \leq i \leq k \}$

- The product MDP $M \otimes A$ is:
  - the MDP $(S \times Q, (s_{init}, q_{init}), \mathbf{Steps}', L')$ where:

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

$\mathbf{Steps}'(s,q) = \{ \mu^q \mid \mu \in Step(s) \}$

$$\mu^q(s', q') = \begin{cases} \mu(s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases}$$

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

(i.e. state sets of acceptance condition used as labels)

Product MDP for a DRA

- For MDP M and DRA A

$$\boxed{p_{max}{}^{M}(s,\ A) = p_{max}{}^{M \otimes A}((s,q_s),\ \vee_{1 \leq i \leq k}\ (FG\ \neg l_i \wedge GF\ k_i)}$$

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

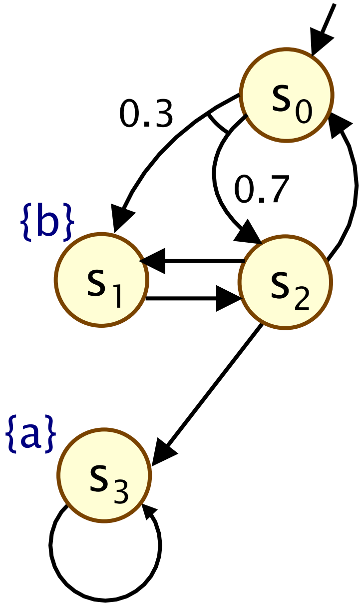
- Hence:

$$\boxed{p_{max}{}^{M}(s,\ A) = p_{max}{}^{M \otimes A}((s,q_s),\ F\ T_{Acc})}$$

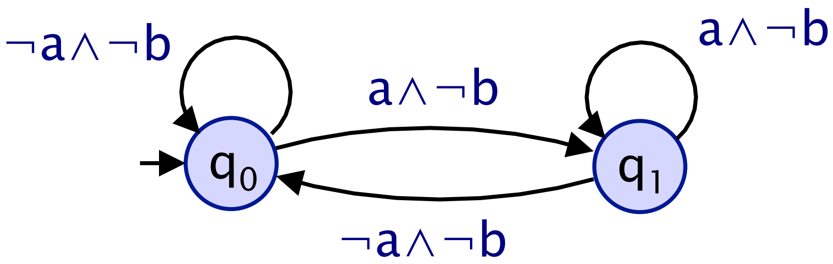
  - where T_{Acc} is the union of all sets T for accepting end components (T,**Steps'**) in $D \otimes A$
  - an accepting end components is such that, for some $1 \leq i \leq k$:
    - $(s,q) \vDash \neg l_i$ for all $(s,q) \in T$ and $(s,q) \vDash k_i$ for some $(s,q) \in T$
    - i.e. $T \cap (S \times L_i) = \varnothing$ and $T \cap (S \times K_i) \neq \varnothing$

MDPs – Example 1

- Model check $P_{<0.8}$ [G ¬b ∧ GF a]



DRA (in fact DBA):

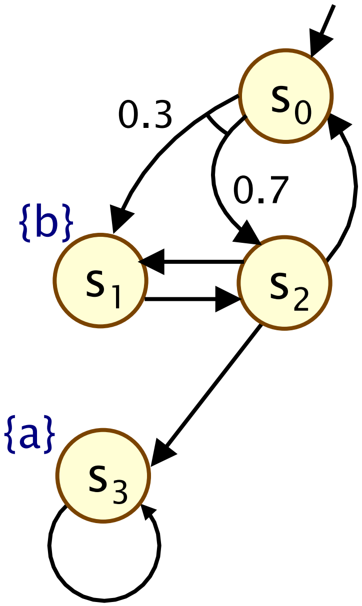


Acc = { (∅, {q_1}) }

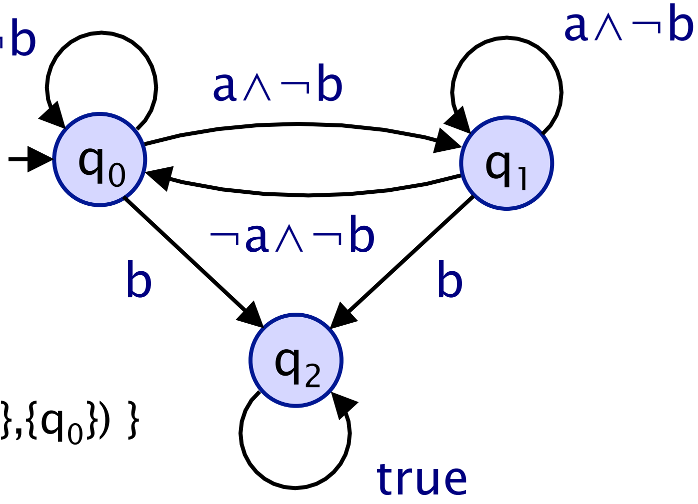
- Result:
  - $\underline{p}_{max}$(G ¬b ∧ GF a) = [0.7, 0, 1, 1]
  - Sat($P_{<0.8}$ [G ¬b ∧ GF a]) = { s_0, s_1 }

MDPs – Example 2

- Model check $P_{>0}$ [G ¬b ∧ GF a]
  - $p_{min}(s, G\ \neg b \wedge GF\ a) = 1 - p_{max}(s, \neg(G\ \neg b \wedge GF\ a))$
    $= 1 - p_{max}(s, F\ b \vee FG\ \neg a))$



DRA:



Acc = { (∅,{q_2}),
({q_1,q_2},{q_0}) }

- Result: $\underline{p}_{min}(G\ \neg b \wedge GF\ a) = [\ 0, 0, 0, 1\ ]$
  - Sat($P_{>0}$ [G ¬b ∧ GF a]) = {s_3}

LTL model checking for MDPs

- Maximal end components
  - can optimise LTL model checking using maximal end components (there may be exponentially many ECs)
- Qualitative LTL model checking
  - no numerical computation: use Prob1E, Prob0A algorithms
- Complexity of model checking LTL formula ψ on MDP M
  - is doubly exponential in |ψ| and polynomial in |M|
  - unlike DTMCs, this cannot be improved upon
- PCTL* model checking
  - LTL model checking can be adapted to PCTL*, as for DTMCs
- Optimal adversaries for LTL formulae
  - memoryless adversary always exists for p_{max}(s, GF a) and for p_{max}(s, FG a) but not for arbitrary LTL formulae

Summing up…

- Deterministic ω-automata (DBA or DRA) and DTMCs
  - probability of language acceptance reduces to probabilistic reachability of set of accepting BSCCs in product DTMC
- LTL model checking for DTMCs
  - via construction of DRA for LTL formula
  - complexity: (doubly) exponential in the size of the LTL formula and polynomial in the size of the DTMC
  - measurability of any ω-regular property on a DTMC
- PCTL* model checking for DTMCs
  - combination of PCTL and LTL model checking algorithms
- LTL model checking for MDPs
  - max. probabilities of reaching accepting end components
  - min. probabilities through negation and max. probabilities

Lecture 19
Probabilistic symbolic model checking

Dr. Dave Parker

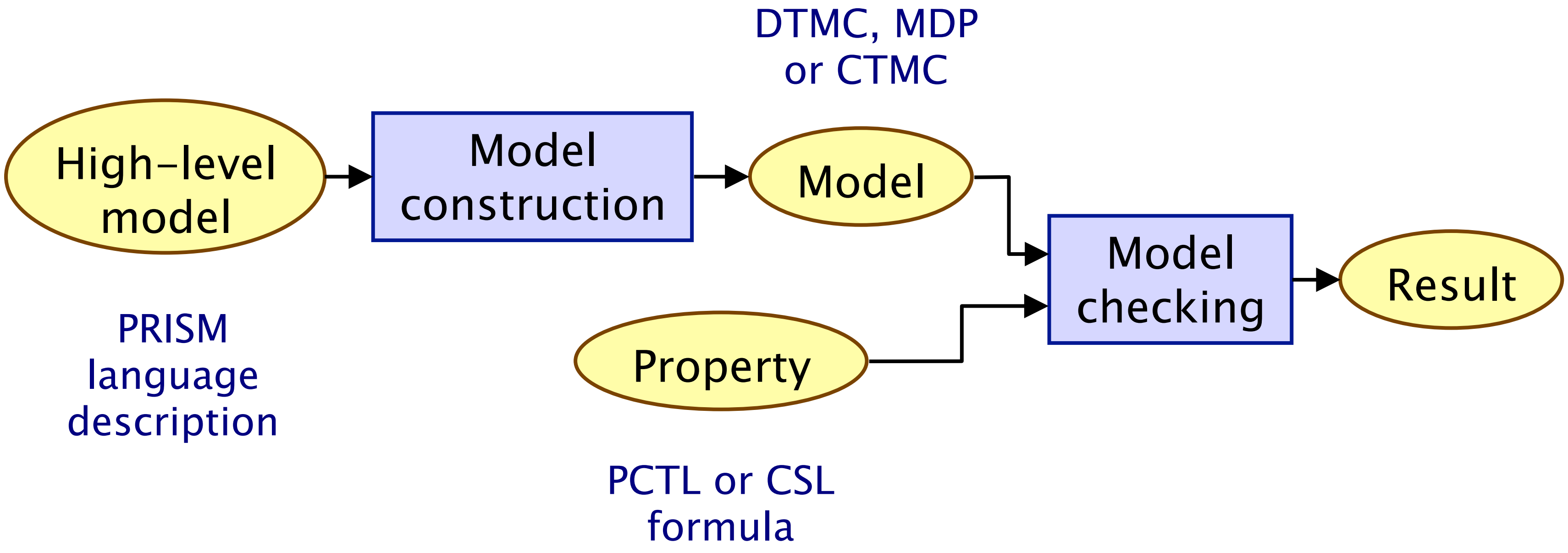
Department of Computer Science
University of Oxford

Overview

- Implementation of probabilistic model checking
  - overview, key operations, symbolic vs. explicit
- Binary decision diagrams (BDDs)
  - introduction, sets, transition relations, …
- Multi-terminal BDDs (MTBDDs)
  - introduction, vectors, matrices, …
- Operations on/with BDDs and MTBDDs

Implementation overview

- Overview of the probabilistic model checking process
  - two distinct phases: model construction, model checking
  - three different models, several different logics, various different solution/analysis methods
  - but… all these processes have much in common



DTMC, MDP or CTMC

High-level model

Model construction

Model

Model checking

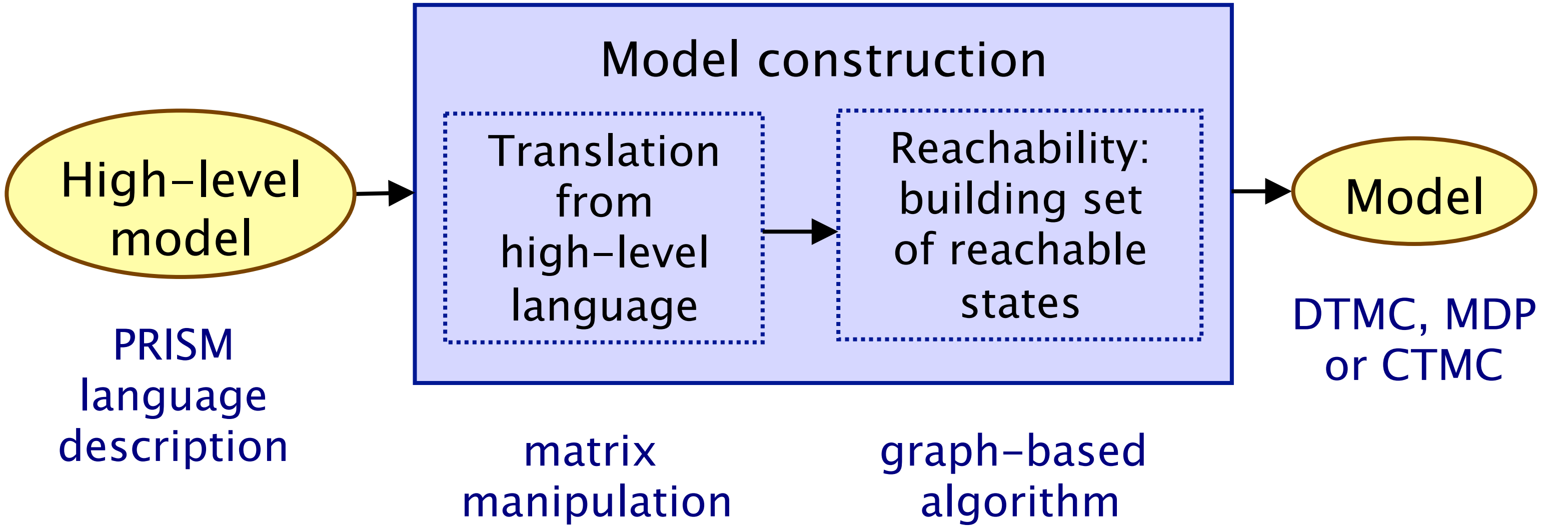
Result

Property

PRISM language description

PCTL or CSL formula

Model construction



High-level model

PRISM language description

Model construction

Translation from high-level language

matrix manipulation

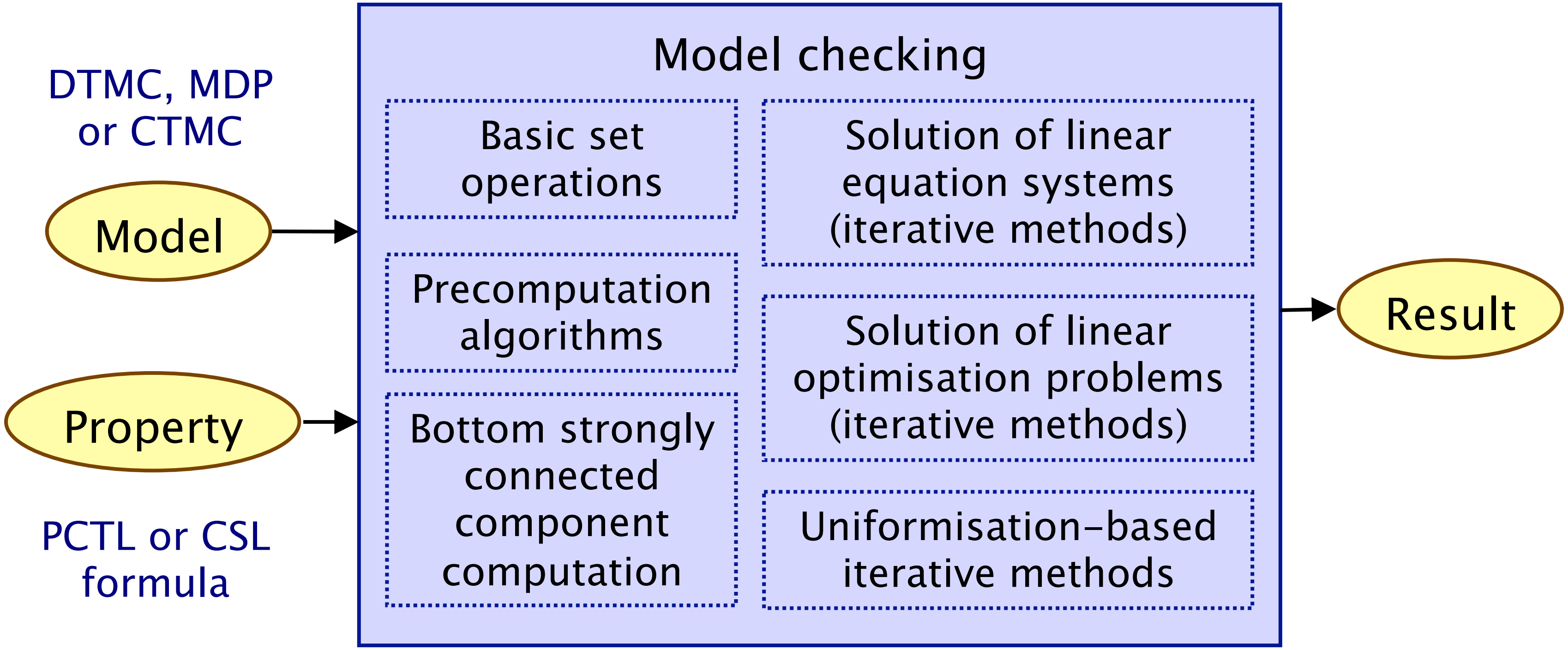
Reachability: building set of reachable states

graph-based algorithm

Model

DTMC, MDP or CTMC

Model checking



DTMC, MDP
or CTMC

Model

Property

PCTL or CSL
formula

Model checking

Basic set operations

Precomputation algorithms

Bottom strongly connected component computation

Solution of linear equation systems (iterative methods)

Solution of linear optimisation problems (iterative methods)

Uniformisation-based iterative methods

Result

Two distinct classes of techniques:

graph-based algorithms
iterative numerical computation

Underlying operations

- Key objects/operations for probabilistic model checking

- Graph-based algorithms
  - underlying transition relation of DTMC/MDP/CTMC
  - manipulation of transition relation and state sets

- Iterative numerical computation
  - transition matrix of DTMC/MDP/CTMC, real-valued vectors
  - manipulation of real-valued matrices and vectors
  - in particular: matrix-vector multiplication

State-space explosion

- Models of real-life systems are typically huge
  - familiar problem for verification/model checking techniques

- State-space explosion problem
  - linear increase in size of system can result in an exponential increase in the size of the model
  - e.g. n parallel components of size m, can give up to m^n states

- Need efficient ways of storing models, sets of states, etc.
  - and efficient ways of constructing, manipulating them

- Here, we will focus on symbolic approaches

Explicit vs. symbolic data structures

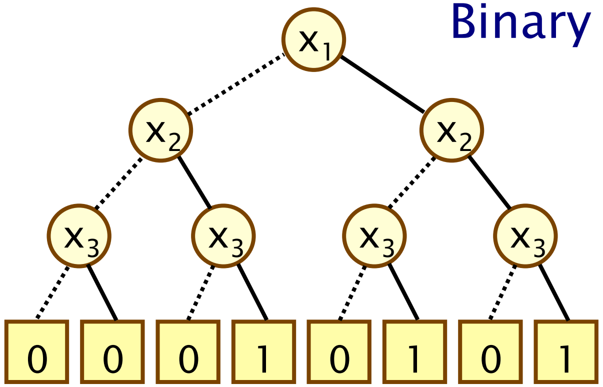
- Symbolic data structures
  - usually based on binary decision diagrams (BDDs) or variants
  - avoid explicit enumeration of data by exploiting regularity
  - potentially very compact storage (but not always)
- Sets of states:
  - explicit: bit vectors
  - symbolic: BDDs
- Real-valued vectors:
  - explicit: arrays of reals (in practice, doubles/floats)
  - symbolic: multi-terminal BDDs (MTBDDs)
- Real-valued matrices:
  - explicit: sparse matrices
  - symbolic: MTBDDs

Representations of Boolean formulas

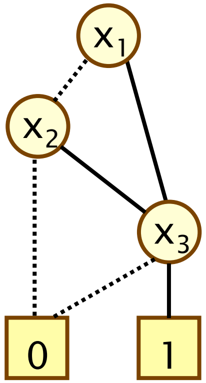
- Propositional formula: $f = (x_1 \vee x_2) \wedge x_3$

Truth table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

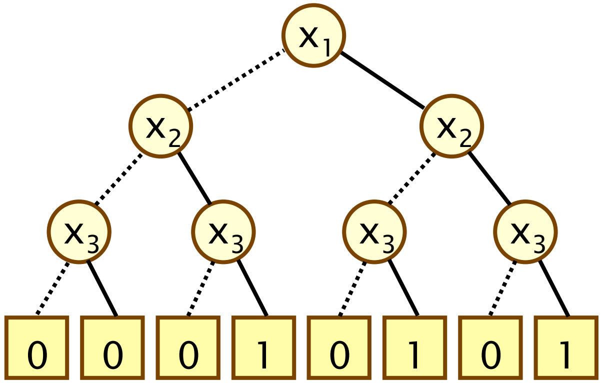

Binary decision tree

Binary decision diagram



Binary decision trees

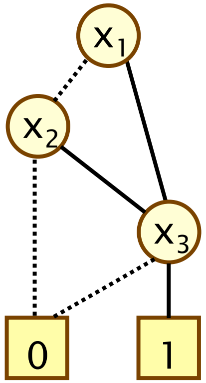
- Graphical representation of Boolean functions
  - $f(x_1,\ldots,x_n) : \{0,1\}^n \rightarrow \{0,1\}$
- Binary tree with two types of nodes
- Non-terminal nodes
  - labelled with a Boolean variable x_i
  - two children: 1 (“then”, solid line) and 0 (“else”, dotted line)
- Terminal nodes (or “leaf” nodes)
  - labelled with 0 or 1
- To read the value of $f(x_1,\ldots,x_n)$
  - start at root (top) node
  - take “then” edge if $x_i=1$
  - take “else” edge if $x_i=0$
  - result given by leaf node



Binary decision diagrams

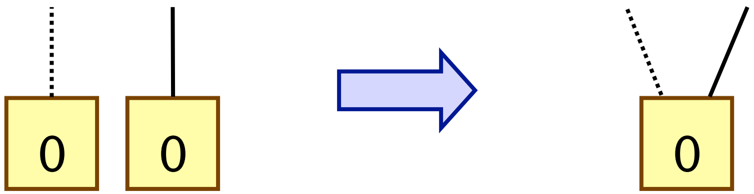
- Binary decision diagrams (BDDs) [Bry86]
  - based on binary decision trees, but reduced and ordered
  - sometimes called reduced ordered BDDs (ROBDDs)
  - actually directed acyclic graphs (DAGs), not trees
  - compact, canonical representation for Boolean functions

- Variable ordering
  - a BDD assumes a fixed total ordering over its set of Boolean variables
  - e.g. $x_1 < x_2 < x_3$
  - along any path through the BDD, variables appear at most once each and always in the correct order

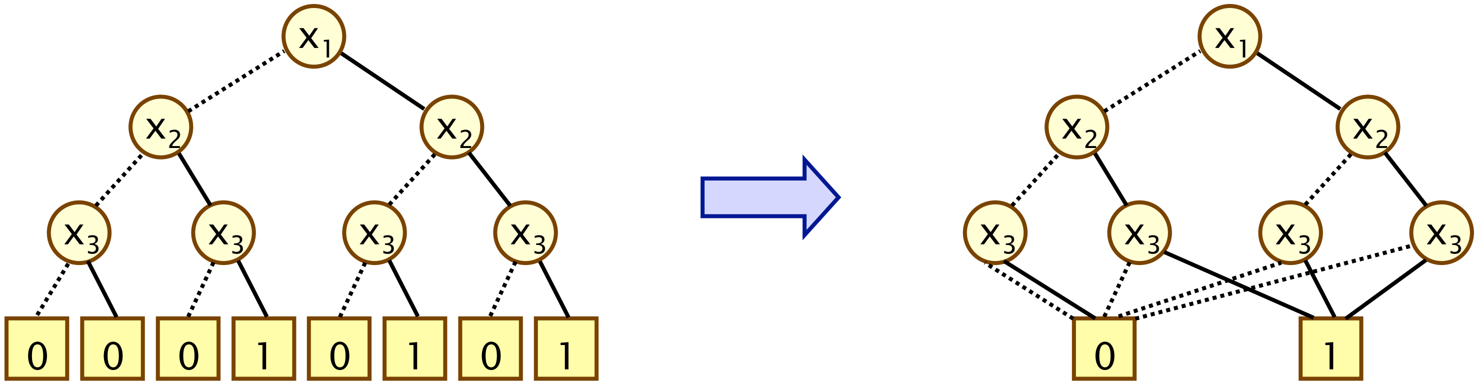


BDD reduction rule 1

- Rule 1: Merge identical terminal nodes

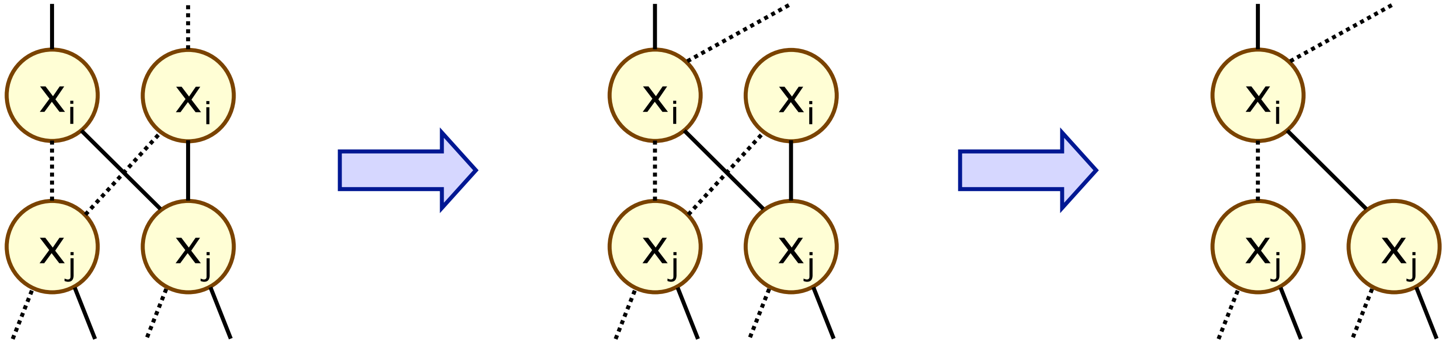


- Example:

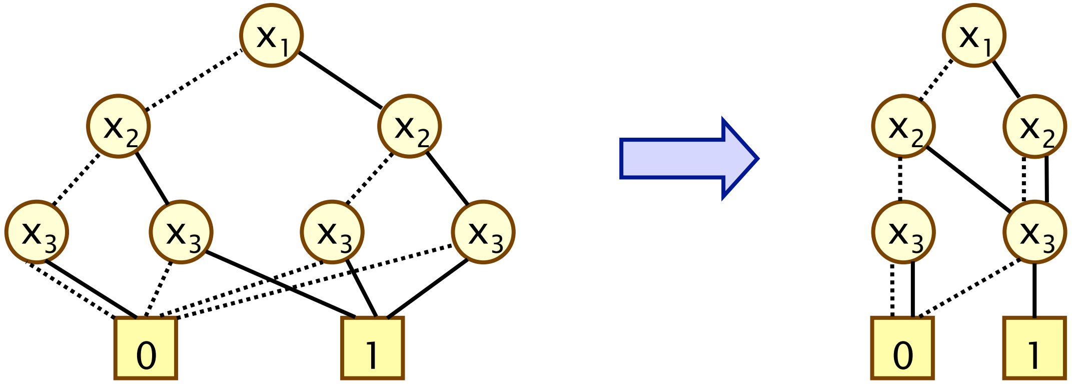


BDD reduction rule 2

- Rule 2: Merge isomorphic nodes, redirect incoming nodes

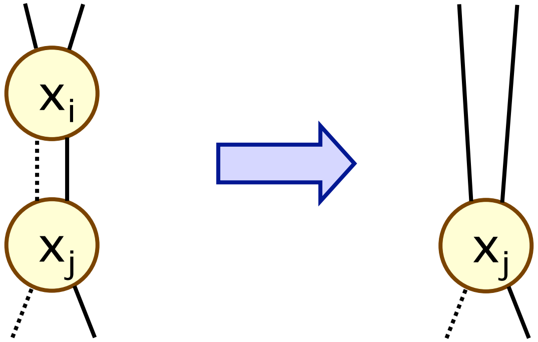


- Example:

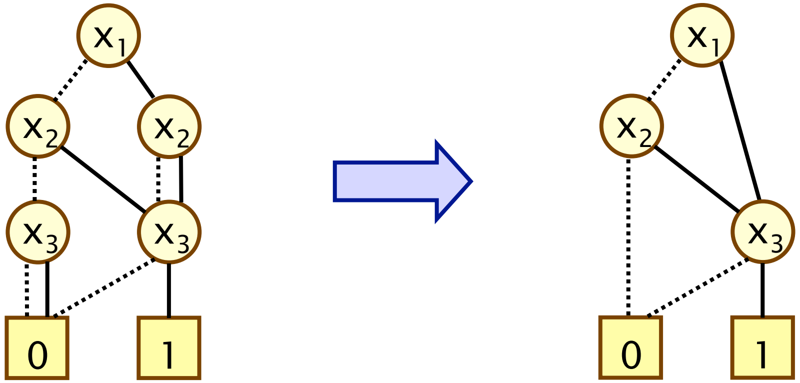


BDD reduction rule 3

- Rule 3: Remove redundant nodes (with identical children)

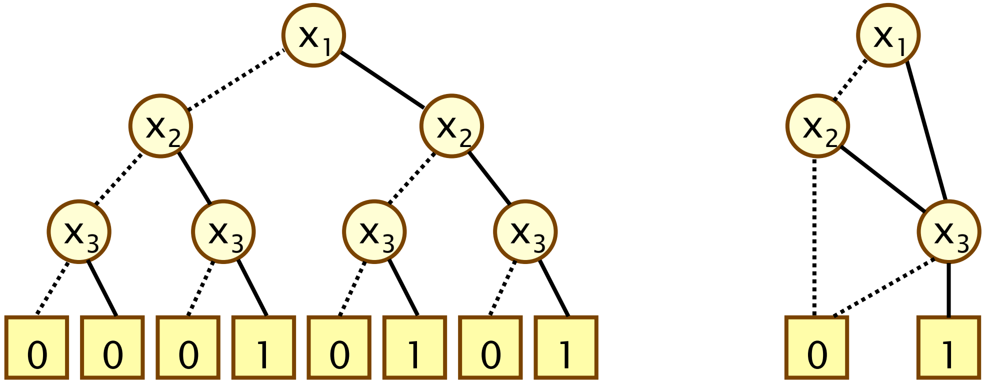


- Example:



Canonicity

- BDDs are a canonical representation for Boolean functions
  - two Boolean functions are equivalent if and only if the BDDs which represent them are isomorphic
  - uniqueness relies on: reduced BDDs, fixed variable ordered



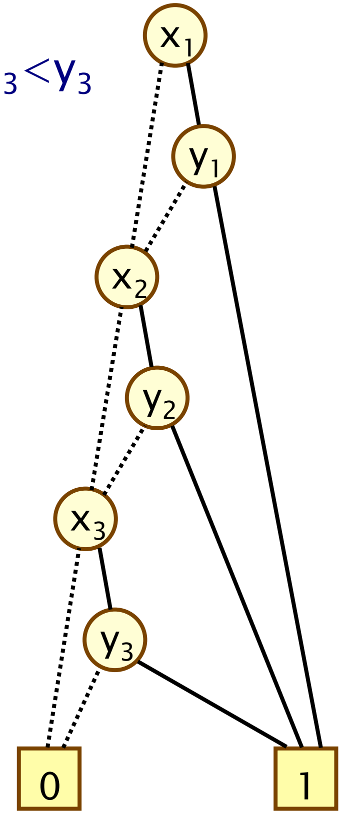
- Important implications for implementation efficiency
  - can be tested in linear (or even constant) time

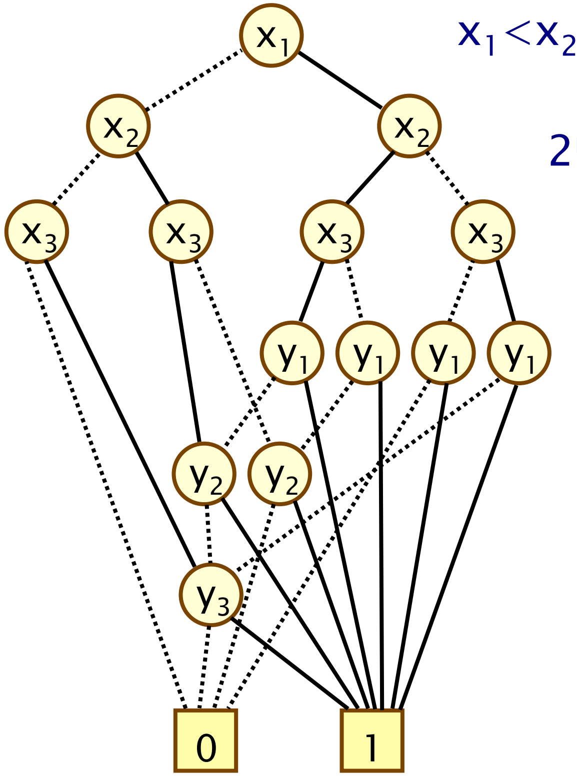
BDD variable ordering

- BDD size can be very sensitive to the variable ordering
  - example: $f = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$

$x_1 < y_1 < x_2 < y_2 < x_3 < y_3$

$2n+2$ nodes





$x_1 < x_2 < x_3 < y_1 < y_2 < y_3$

2^{n+1} nodes

BDDs to represent sets of states

- Consider a state space S and some subset $S' \subseteq S$
- We can represent S' by its characteristic function $\chi_{S'}$
  - $\chi_{S'} : S \rightarrow \{0,1\}$ where $\chi_{S'}(s) = 1$ if and only if $s \in S'$
- Assume we have an encoding of S into n Boolean variables
  - this is always possible for a finite set S
  - e.g. enumerate the elements of S and use a binary encoding
  - (note: there may be more efficient encodings though)
- So $\chi_{S'}$ can be seen as a function $\chi_{S'}(x_1,\ldots x_n) : \{0,1\}^n \rightarrow \{0,1\}$
  - which is simply a Boolean function
  - which can therefore be represented as a BDD

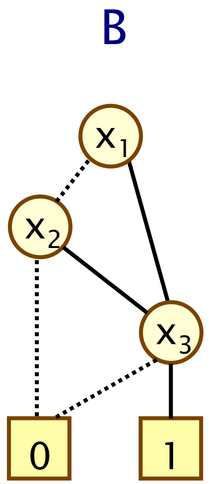
BDD and sets of states – Example

- State space S: {0, 1, 2, 3, 4, 5, 6, 7}
- Encoding of S: {000, 001, 010, 011, 100, 101, 110, 111}
- Subset S' ⊆ S: {3, 5, 7} → {011, 101, 111}

Truth table:

x_1	x_2	x_3	f_B
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

BDD:



BDDs and transition relations

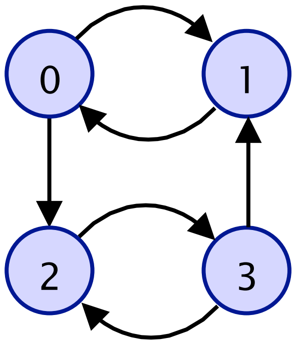
- Transition relations can also be represented by their characteristic function, but over pairs of states
  - relation: $R \subseteq S \times S$
  - characteristic function: $\chi_R : S \times S \rightarrow \{0,1\}$

- For an encoding of state space S into n Boolean variables
  - we have Boolean function $f_R(x_1,\ldots,x_n,y_1,\ldots,y_n) : \{0,1\}^{2n} \rightarrow \{0,1\}$
  - which can be represented by a BDD

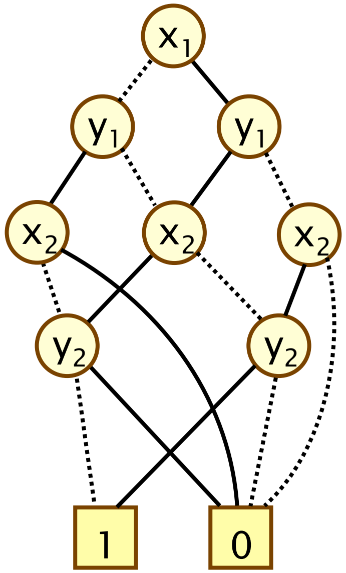
- Row and column variables
  - for efficiency reasons, we interleave the row variables $x_1,..,x_n$ and column variables $y_1,\ldots,y_n$
  - i.e. we use function $f_R(x_1,y_1,\ldots,x_n,y_n) : \{0,1\}^{2n} \rightarrow \{0,1\}$

BDDs and transition relations

- Example:
  - 4 states: 0, 1, 2, 3
  - Encoding: 0↦00, 1↦01, 2↦10, 3↦11



Transition	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$
(0,1)	0	0	0	1	0001
(0,2)	0	0	1	0	0100
(1,0)	0	1	0	0	0010
(2,3)	1	0	1	1	1101
(3,1)	1	1	0	1	1011
(3,2)	1	1	1	0	1110

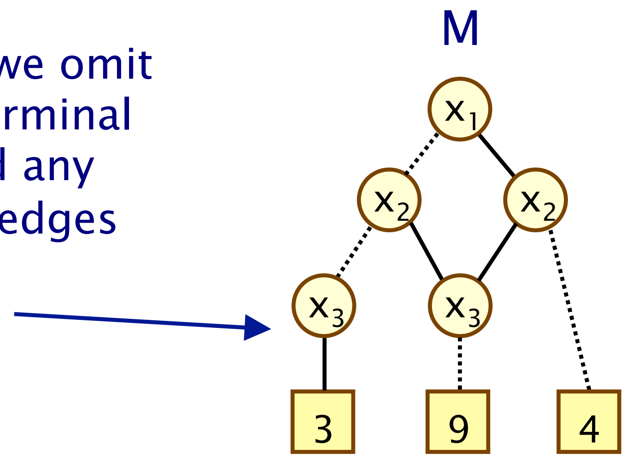


Multi-terminal binary decision diagrams

- Multi-terminal BDDs (MTBDDs), sometimes called ADDs
  - extension of BDDs to represent real-valued functions
  - like BDDs, an MTBDD M is associated with n Boolean variables
  - MTBDD M represents a function $f_M(x_1,\ldots,x_n) : \{0,1\}^n \to \mathbb{R}$

For clarity, we omit the zero terminal node and any incoming edges

e.g.



x_1	x_2	x_3	f_M
0	0	0	0
0	0	1	3
0	1	0	9
0	1	1	0
1	0	0	4
1	0	1	4
1	1	0	9
1	1	1	0

MTBDDs to represent vectors

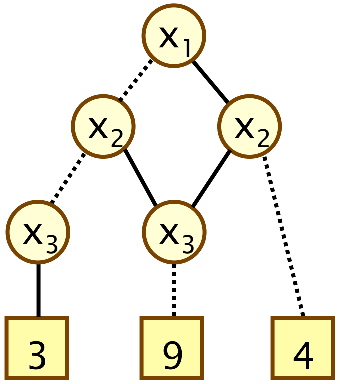
- In the same way that BDDs can represent sets of states…
  - MTBDDs can represent real-valued vectors over states S
  - e.g. a vector of probabilities Prob(s, ψ) for each state s ∈ S
  - assume we have an encoding of S into n Boolean variables
  - then vector $\underline{v} : S \rightarrow \mathbb{R}$ is a function $f_v(x_1,\ldots,x_n) : \{0,1\}^n \rightarrow \mathbb{R}$

Vector $\underline{v}$

[0,3,9,0,4,4,9,0]

x_1	x_2	x_3	i	f_v
0	0	0	0	0
0	0	1	1	3
0	1	0	2	9
0	1	1	3	0
1	0	0	4	4
1	0	1	5	4
1	1	0	6	9
1	1	1	7	0

MTBDD v



MTBDDs to represent matrices

- MTBDDs can be used to represent real-valued matrices indexed over a set of states S
  - e.g. the transition probability/rate matrix of a DTMC/CTMC

- For an encoding of state space S into n Boolean variables
  - a matrix **M** maps pairs of states to reals i.e. **M** : S × S→ℝ
  - this becomes: $f_M(x_1,\ldots,x_n,y_1,\ldots,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$

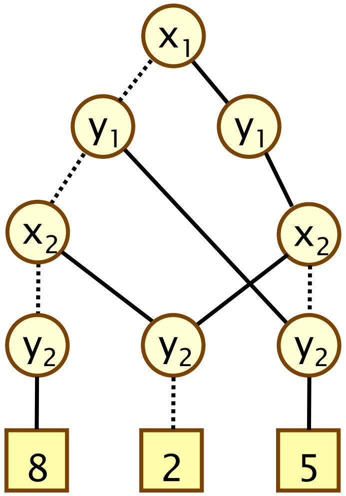
- Row and column variables
  - for efficiency reasons, we interleave the row variables $x_1,..,x_n$ and column variables $y_1,\ldots,y_n$
  - i.e. we use function $f_M(x_1,y_1,\ldots,x_n,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$

Matrices and MTBDDs – Example

Matrix **M** $\begin{bmatrix} 0 & 8 & 0 & 5 \\ 2 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \end{bmatrix}$

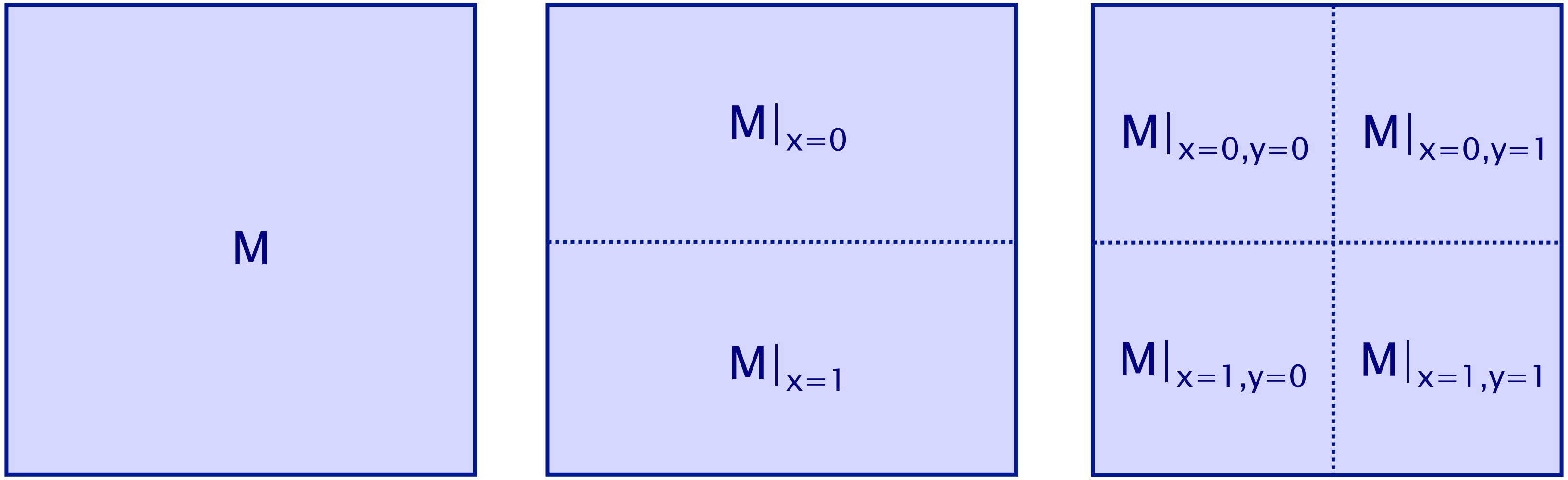
Entry in M	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$	f_M
(0,1) = 8	0	0	0	1	0001	8
(1,0) = 2	0	1	0	0	0010	2
(0,3) = 5	0	0	1	1	0101	5
(1,3) = 5	0	1	1	1	0111	5
(2,3) = 5	1	0	1	1	1101	5
(3,2) = 2	1	1	1	0	1110	2

MTBDD M



Matrices and MTBDDs – Recursion

- Descending one level in the MTBDD (i.e. setting $x_i=b$)
  - splits the matrix represented by the MTBDD in half
  - row variables (x_i) give horizontal split
  - column variables (y_i) give vertical split



M

$M|_{x=0}$

$M|_{x=1}$

$M|_{x=0,y=0}$ $M|_{x=0,y=1}$

$M|_{x=1,y=0}$ $M|_{x=1,y=1}$

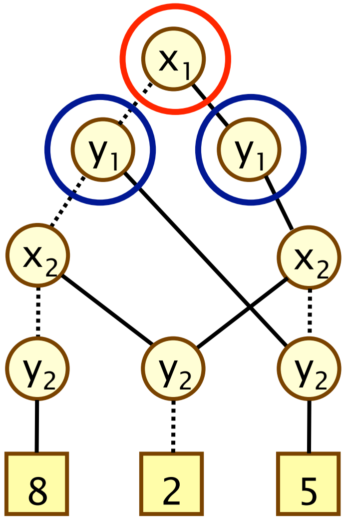
Matrices and MTBDDs – Recursion

Matrix **M**

$$\begin{bmatrix} \begin{array}{cc|cc} 0 & 8 & 0 & 5 \\ 2 & 0 & 0 & 5 \\ \hline 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \end{array} \end{bmatrix}$$

Entry in M	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$	f_M
(0,1) = 8	0	0	0	1	0001	8
(1,0) = 2	0	1	0	0	0010	2
(0,3) = 5	0	0	1	1	0101	5
(1,3) = 5	0	1	1	1	0111	5
(2,3) = 5	1	0	1	1	1101	5
(3,2) = 2	1	1	1	0	1110	2

MTBDD M



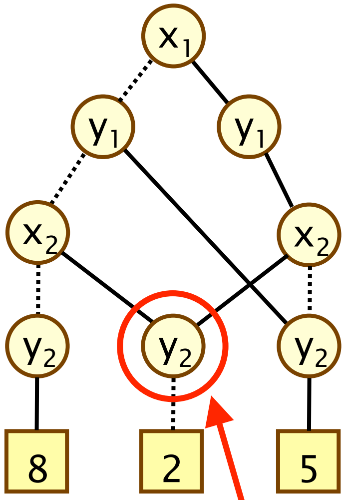
Matrices and MTBDDs – Regularity

Matrix **M**

$$\begin{bmatrix} 0 & 8 & 0 & 5 \\ 2 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

Repeated submatrices

MTBDD M



Shared MTBDD node

Entry in M	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$	f_M
(0,1) = 8	0	0	0	1	0001	8
(1,0) = 2	0	1	0	0	0010	2
(0,3) = 5	0	0	1	1	0101	5
(1,3) = 5	0	1	1	1	0111	5
(2,3) = 5	1	0	1	1	1101	5
(3,2) = 2	1	1	1	0	1110	2

Matrices and MTBDDs – Regularity

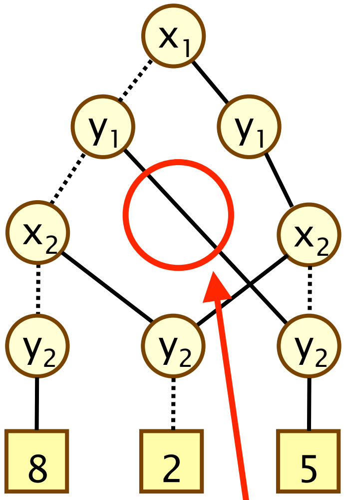
Matrix **M**

$$\begin{bmatrix} 0 & 8 & 0 & 5 \\ 2 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

Identical adjacent submatrices

Entry in M	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$	f_M
(0,1) = 8	0	0	0	1	0001	8
(1,0) = 2	0	1	0	0	0010	2
(0,3) = 5	0	0	1	1	0101	5
(1,3) = 5	0	1	1	1	0111	5
(2,3) = 5	1	0	1	1	1101	5
(3,2) = 2	1	1	1	0	1110	2

MTBDD M



MTBDD node removed

Matrices and MTBDDs – Sparseness

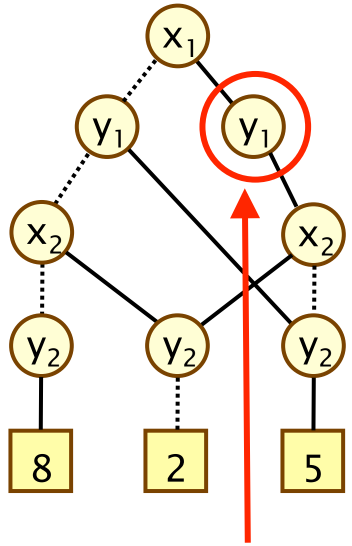
Matrix **M**

$$\begin{bmatrix} 0 & 8 & 0 & 5 \\ 2 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

Blocks of zeros

Entry in M	x_1	x_2	y_1	y_2	$x_1y_1x_2y_2$	f_M
(0,1) = 8	0	0	0	1	0001	8
(1,0) = 2	0	1	0	0	0010	2
(0,3) = 5	0	0	1	1	0101	5
(1,3) = 5	0	1	1	1	0111	5
(2,3) = 5	1	0	1	1	1101	5
(3,2) = 2	1	1	1	0	1110	2

MTBDD M

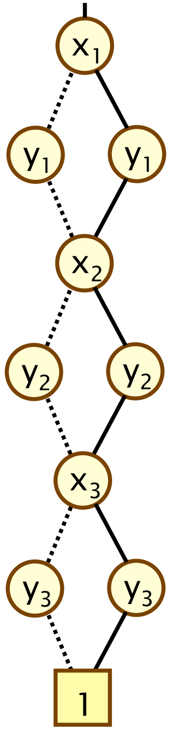


Edge goes straight to zero node

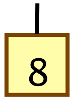
Matrices and MTBDDs – Compactness

- Some simple matrices have extremely compact representations as MTBDDs
  - e.g. the identify matrix or a constant matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 8 & 8 & 8 & 8 & \cdots \\ 8 & 8 & 8 & 8 & \cdots \\ 8 & 8 & 8 & 8 & \cdots \\ 8 & 8 & 8 & 8 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$



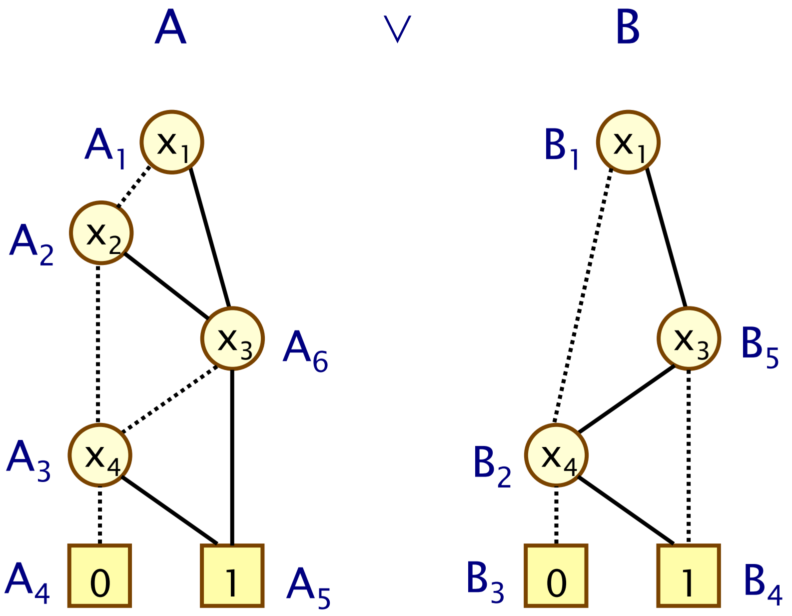
Manipulating BDDs

- Need efficient ways to manipulate Boolean functions
  - while they are represented as BDDs
  - i.e. algorithms which are applied directly to the BDDs
- Basic operations on Boolean functions:
  - negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), etc.
  - can all be applied directly to BDDs
- Key operation on BDDs: Apply(op, A, B)
  - where A and B are BDDs and op is a binary operator over Boolean values, e.g. $\wedge$, $\vee$, etc.
  - Apply(op, A, B) returns the BDD representing function f_A op f_B
  - often just use infix notation, e.g. Apply($\wedge$, A, B) = A $\wedge$ B
  - efficient algorithm: recursive depth-first traversal of A and B
  - complexity (and size of result) is $O(|A|\cdot|B|)$
    - where $|C|$ denotes size of BDD C

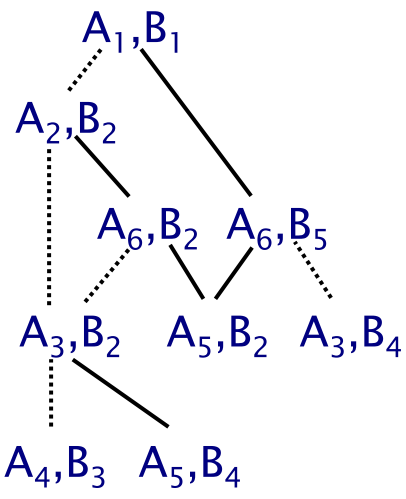
Apply – Example

- Example: Apply($\vee$, A, B)

Argument BDDs, with node labels:

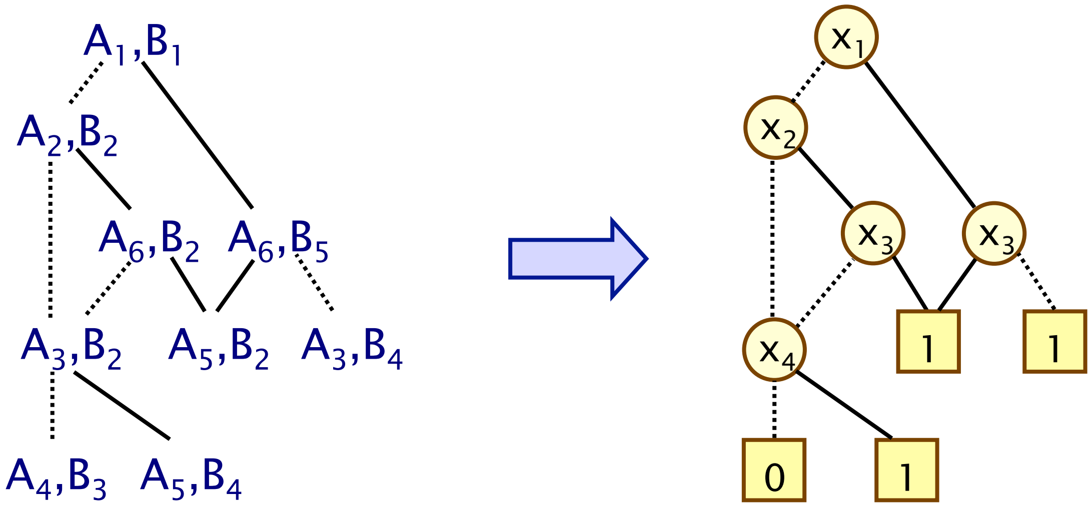


Recursive calls to Apply:



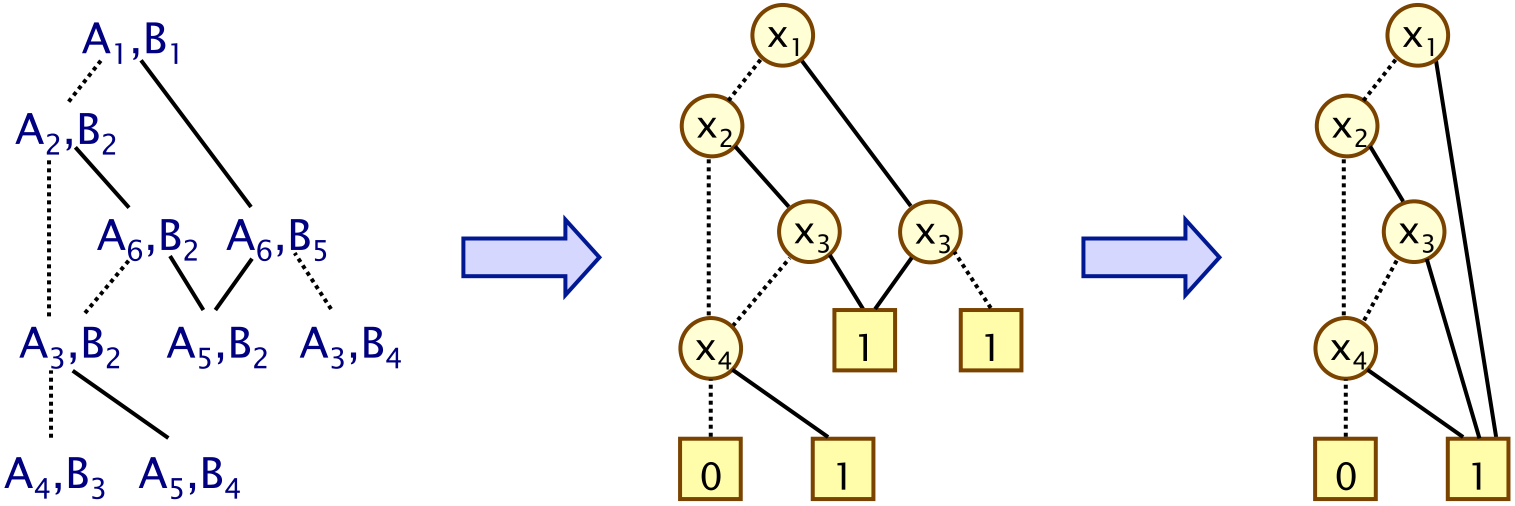
Apply – Example

- Example: Apply(∨, A, B)
  - recursive call structure implicitly defines resulting BDD



Apply – Example

- Example: Apply(∨, A, B)
  - but the resulting BDD needs to be reduced
  - in fact, we can do this as part of the recursive Apply operation, implementing reduction rules bottom-up



Implementation of BDDs

- Store all BDDs currently in use as one multi-rooted BDD
  - no duplicate BDD subtrees, even across multiple BDDs
  - every time a new node is created, check for existence first
  - sometimes called the "unique table"
  - implemented as set of hash tables, one per Boolean variable
  - need: node referencing/dereferencing, garbage collection
- Efficiency implications
  - very significant memory savings
  - trivial checking of BDD equality (pointer comparison)
- Caching of BDD operation results for reuse
  - store result of every BDD operation (memory dependent)
  - applied at every step of recursive BDD operations
  - relies on fast check for BDD equality

Operations with BDDs

- Operations on sets of states easy with BDDs
  - set union: $A \cup B$, in BDDs: $A \vee B$
  - set intersection: $A \cap B$, in BDDs: $A \wedge B$
  - set complement: $S \setminus A$, in BDDs: $\neg A$

- Graph-based algorithms (e.g. reachability)
  - need forwards or backwards image operator
    - i.e. computation of all successors/predecessors of a state
    - again, easy with BDD operations (conjunction, quantification)
  - other ingredients
    - set operations (see above)
    - equality of state sets (fixpoint termination) – equality of BDDs

Operations on MTBDDs

- The BDD operation Apply extends easily to MTBDDs

- For MTBDDs A, B and binary operation op over the reals:
  - Apply(op, A, B) returns the MTBDD representing f_A op f_B
  - examples for op: $+$, $-$, $\times$, min, max, …
  - often just use infix notation, e.g. Apply($+$, A, B) $=$ A $+$ B

- BDDs are just an instance of MTBDDs
  - in this case, can use Boolean ops too, e.g. Apply($\vee$, A, B)

- The recursive algorithm for implementing Apply on BDDs
  - can be reused for Apply on MTBDDs

Some other MTBDD operations

- Threshold(A, ~, c)
  - for MTBDD A, relational operator op and bound $c \in \mathbb{R}$
  - converts MTBDD to BDD based on threshold ~c
  - i.e. builds BDD representing function $f_A \sim c$
  - e.g. computing the underlying transition relation from the probability matrix of a DTMC: R = Threshold(P, >, 0)

- Abstract(op, $\{x_1,\ldots,x_n\}$, A)
  - for MTBDD A, variables $\{x_1,\ldots,x_n\}$ and commutative/associative binary operator over reals op
  - analogue of existential/universal quantification for BDDs
  - e.g. Abstract(+, {x}, A) constructs the MTBDD representing the function $f_{A|x=0} + f_{A|x=1}$
  - e.g. for BDD A: $\exists(x_1,..,x_n).A \equiv$ Abstract($\vee$, $\{x_1,\ldots,x_n\}$, A)

MTBDD matrix/vector operations

- Pointwise addition/multiplication and scalar multiplication
  - can be implemented with the Apply operator
  - Matrices: **A** + **B**, MTBDDs: Apply(+, A, B)

- Matrix-matrix multiplication **A·B**
  - can be expressed recursively based on 4-way matrix splits

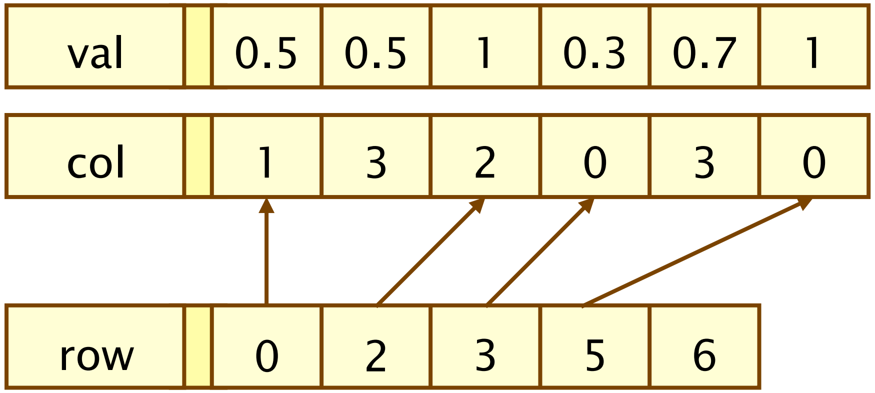
$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} \cdot \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} \qquad \mathbf{A}_1 = \mathbf{B}_1 \cdot \mathbf{C}_1 + \mathbf{B}_2 \cdot \mathbf{C}_3, \text{ etc.}$$

  - which forms the basis of an MTBDD implementation
  - various optimisations are possible

- Matrix-matrix multiplication **A**·$\underline{\mathbf{v}}$ is done in similar fashion

Sparse matrices

- Explicit data structure for matrices with many zero entries
  - assume a matrix **P** of size n × n with nnz non-zero elements
  - store three arrays: val and col (of size nnz) and row (of size n)
  - for each matrix entry (r,c)=v, c and v are stored in col/val
  - entries are grouped by row, with pointers stored in row
  - also possible to group by column



val	0.5	0.5	1	0.3	0.7	1
col	1	3	2	0	3	0
row	0	2	3	5	6	

$$P = \begin{bmatrix} \cdot & 0.5 & \cdot & 0.5 \\ \cdot & \cdot & 1 & \cdot \\ 0.3 & \cdot & \cdot & 0.7 \\ 1 & \cdot & \cdot & \cdot \end{bmatrix}$$

Sparse matrices

- Advantages
  - compact storage (proportional to number of non-zero entries)
  - fast access to matrix entries
  - especially if usually need an entire row at once
  - (which is the case for e.g. matrix-vector multiplication)

- Disadvantage
  - less efficient to manipulate (i.e. add/delete matrix entries)

- Storage requirements
  - for a matrix of size $n \times n$ with nnz non-zero elements
  - assume reals are 8 byte doubles, indices are 4 byte integers
  - we need $8 \cdot nnz + 4 \cdot nnz + 4 \cdot n = 12 \cdot nnz + 4 \cdot n$ bytes

Sparse matrices vs. MTBDDs

- Storage requirements
  - MTBDDs: each node is 20 bytes
  - sparse matrices: 12·nnz+4·n bytes (n states, nnz transitions)
- Case study: Kanban manufacturing system, N jobs
  - store transition rate matrix **R** of the corresponding CTMCs

N	States (n)	Transitions (nnz)	MTBDD (KB)	Sparse matrix (KB)
3	58,400	446,400	48	5,459
4	454,475	3,979,850	96	48,414
5	2,546,432	24,460,016	123	296,588
6	11,261,376	115,708,992	154	1,399,955
7	41,644,800	450,455,040	186	5,441,445
8	133,865,325	1,507,898,700	287	13,193,599

Implementation in PRISM

- PRISM is a symbolic probabilistic model checker
  - the key underlying data structures are MTBDDs (and BDDs)

- In fact, has multiple numerical computation engines
  - MTBDDs: storage/analysis of very large models (given structure/regularity), numerical computation can blow up
  - Sparse matrices: fastest solution for smaller models (<10^6 states), prohibitive memory consumption for larger models
  - Hybrid: combine MTBDD storage with explicit storage, ten-fold increase in analysable model size (~10^7 states)

Summing up…

- Implementation of probabilistic model checking
  - graph-based algorithms, e.g. reachability, precomputation
  - manipulation of sets of states, transition relations
  - iterative numerical computation
  - key operation: matrix-vector multiplication
- Binary decision diagrams (BDDs)
  - representation for Boolean functions
  - efficient storage/manipulation of sets, transition relations
- Multi-terminal BDDs (MTBDDs)
  - extension of BDDs to real-valued functions
  - efficient storage/manipulation of real-valued vectors, matrices (assuming structure and regularity)
  - can be much more compact than (explicit) sparse matrices