Verification of Probabilistic Systems

Dave Parker

University of Oxford

MOVEP'08 - Orléans - June 2008

Motivation – Why probability?

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

Motivation – Why probability?

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

Motivation – Why probability?

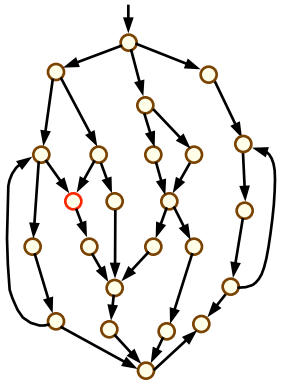
- Some systems are inherently probabilistic…
- Randomisation, e.g. in distributed coordination algorithms
  - as a symmetry breaker, in gossip routing to reduce flooding
- To model uncertainty and performance
  - to quantify rate of failures, express Quality of Service
- To model biological processes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

Verifying probabilistic systems

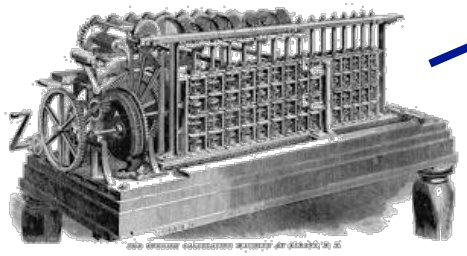
- We are not just interested in correctness
- We want to be able to quantify:
  - security, privacy, trust, anonymity, fairness
  - safety, reliability, performance, dependability
  - resource usage, e.g. battery life
  - and much more…
- Quantitative, as well as qualitative requirements:
  - how reliable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

Verification via model checking

Finite-state model



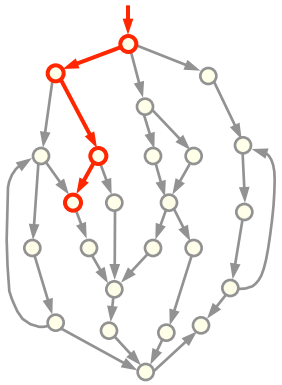
Model checker



Result

¬EF error

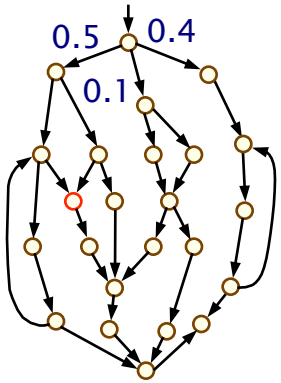
Temporal logic specification



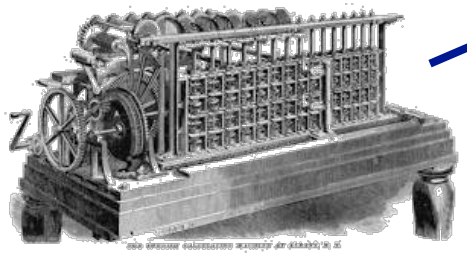
Error trace

Probabilistic model checking

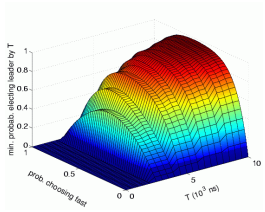
Probabilistic model
e.g. Markov chain



Probabilistic model checker
e.g. PRISM

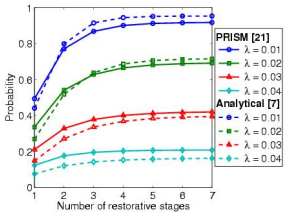


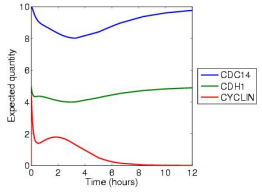
Result



$P_{<0.01}$ [F error]

Probabilistic temporal logic specification
e.g. PCTL





Quantitative results

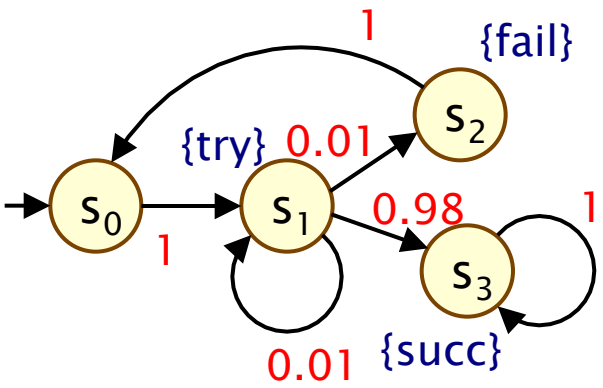
Overview

- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

Discrete-time Markov chains

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

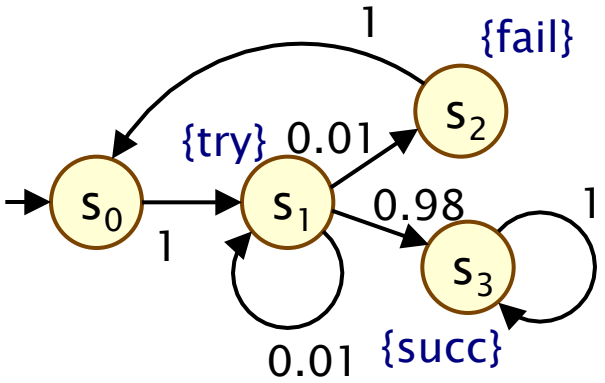
- States
  - discrete set of states representing possible configurations of the system being modelled
- Transitions
  - transitions between states occur in discrete time-steps
- Probabilities
  - probability of making transitions between states is given by discrete probability distributions



Discrete-time Markov chains

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

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



DTMCs: An alternative definition

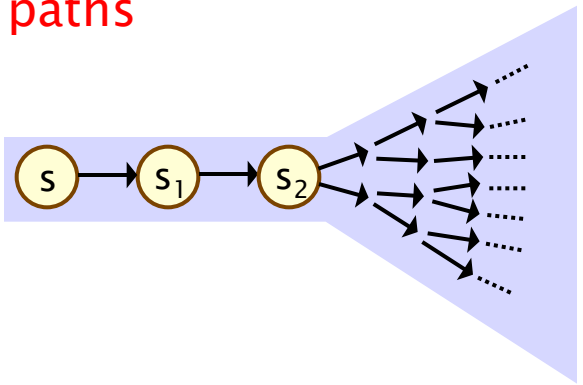
- Alternative definition: a DTMC is:
  - a family of random variables $\{ X(k) \mid k=0,1,2,\ldots \}$
  - $X(k)$ are observations at discrete time-steps
  - i.e. $X(k)$ is the state of the system at time-step k

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

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

Paths and probabilities

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



Probability spaces

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

Probability space over paths

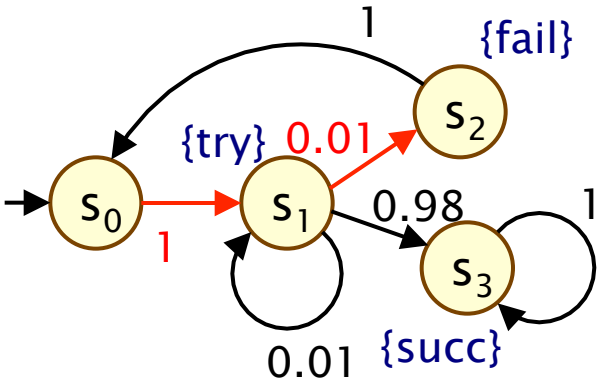
- Sample space Ω = Path(s)

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

- See [KSK76] for further details

Probability space – Example

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



- Paths which are eventually successful and with no failures
  - $C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots$
  - $Pr_{s0}(C(s_0 s_1 s_3) \cup C(s_0 s_1 s_1 s_3) \cup C(s_0 s_1 s_1 s_1 s_3) \cup \ldots)$
    $= P_{s0}(s_0 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_3) + P_{s0}(s_0 s_1 s_1 s_1 s_3) + \ldots$
    $= 1 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.98 + 1 \cdot 0.01 \cdot 0.01 \cdot 0.98 + \ldots$
    $= 0.9898989898\ldots$
    $= 98/99$

Overview

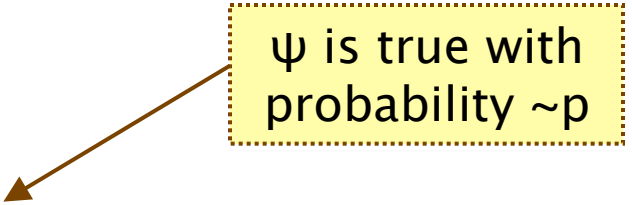
- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

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

PCTL syntax

- PCTL syntax:



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

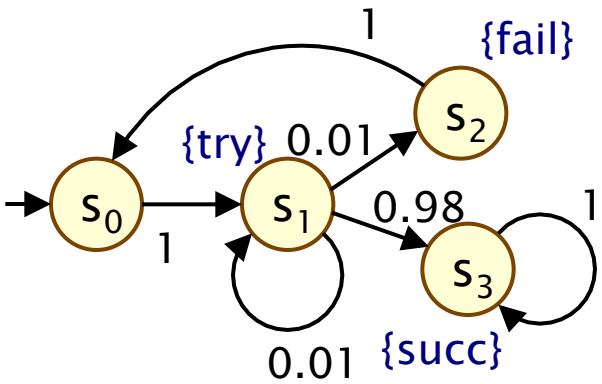
"next" — "bounded until" — "until"

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

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

PCTL semantics for DTMCs

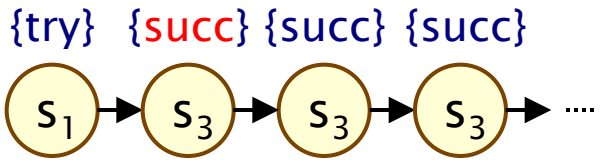
- PCTL formulas interpreted over states of a DTMC
  - $s \models \phi$ denotes ϕ is "true in state s" or "satisfied in state s"
- Semantics of (non-probabilistic) state formulas:
  - for a state s of the DTMC $(S, s_{init}, \mathbf{P}, L)$:
  - $s \models a \quad \Leftrightarrow \quad a \in L(s)$
  - $s \models \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad s \models \phi_1$ and $s \models \phi_2$
  - $s \models \neg\phi \quad \Leftrightarrow \quad s \models \phi$ is false
- Examples
  - $s_3 \models$ succ
  - $s_1 \models$ try $\wedge$ $\neg$fail



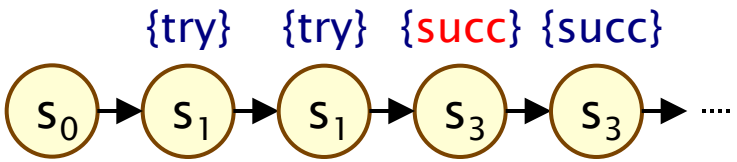
PCTL semantics for DTMCs

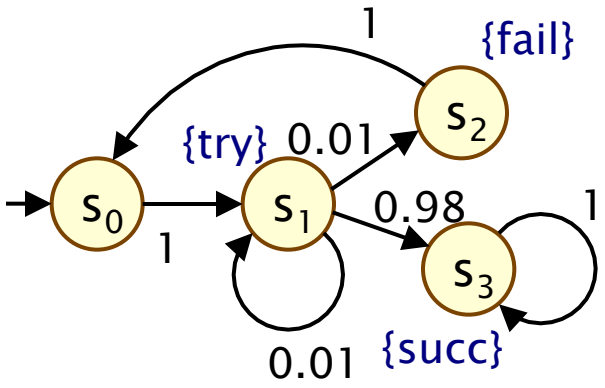
- Semantics of path formulas:
  - for a path $\omega = s_0 s_1 s_2 \ldots$ in the DTMC:
  - $\omega \models X\, \phi \quad \Leftrightarrow \quad s_1 \models \phi$
  - $\omega \models \phi_1\, U^{\leq k}\, \phi_2 \quad \Leftrightarrow \quad \exists i \leq k$ such that $s_i \models \phi_2$ and $\forall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

- Some examples of satisfying paths:
  - X succ



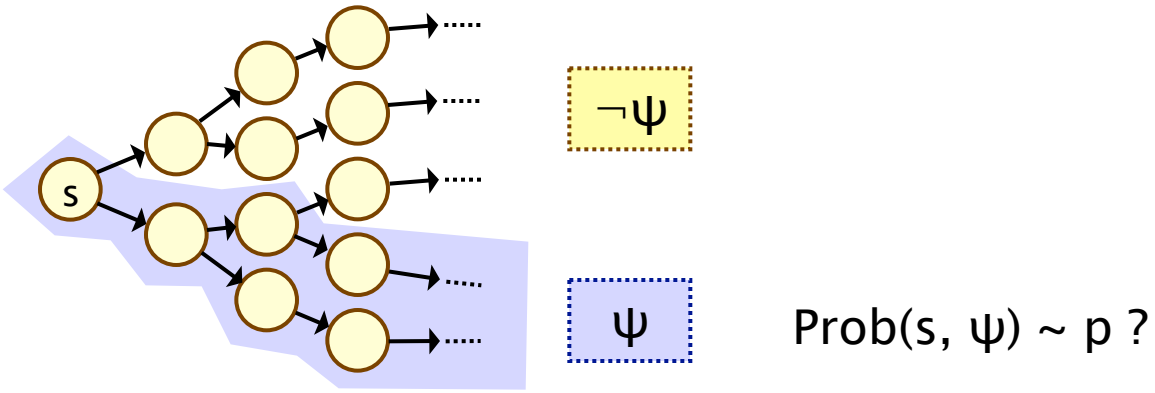
  - ¬fail U succ





PCTL semantics for DTMCs

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



More PCTL…

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

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

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

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)

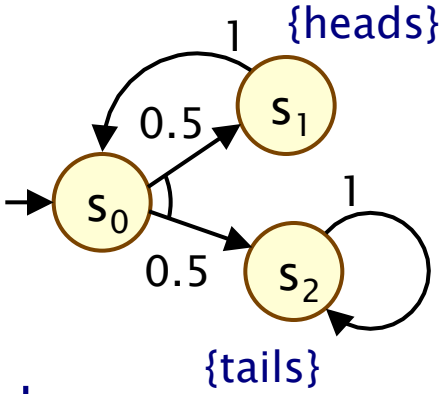
- Recall: probability space $(Path(s), \Sigma_{Path(s)}, Pr_s)$
  - $\Sigma_{Path(s)}$ contains cylinder sets $C(\omega)$ for all finite paths ω starting in s and is closed under complementation, countable union

- 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)

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

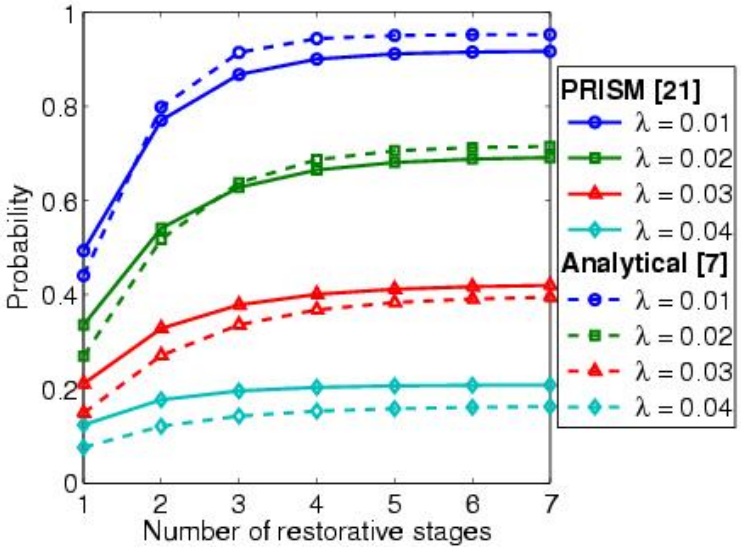


- $P_{>0}$ [F ϕ] is identical to EF ϕ
  - there exists a finite path to a ϕ-state

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

Quantitative properties

- Consider a PCTL formula $P_{\sim p}$ [ψ]
  - if the probability is unknown, how to choose the bound p?
- When the outermost operator of a PTCL formula is P
  - we allow the form $P_{=?}$ [ψ]
  - "what is the probability that path formula ψ is true?"
- Model checking is no harder: compute the values anyway
- Useful to spot patterns, trends
- Example
  - $P_{=?}$ [F err/total>0.1]
  - "what is the probability that 10% of the NAND gate outputs are erroneous?"



Some real PCTL examples

reliability

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

performance

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

fairness

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

Overview

- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

PCTL model checking for DTMCs

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

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

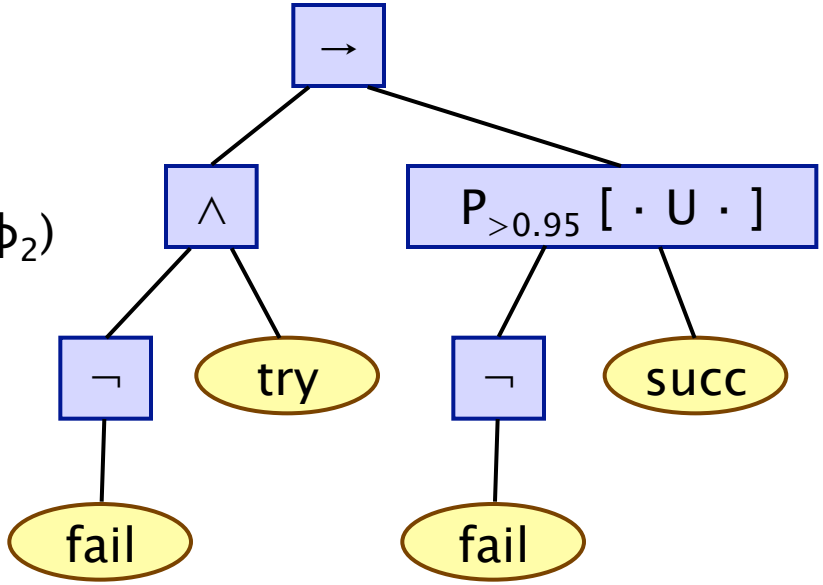
- Sometimes, focus on quantitative results
  - e.g. compute result of P=? [F error]
  - e.g. compute result of P=? [$F^{\leq k}$ error] for $0 \leq k \leq 100$

PCTL model checking for DTMCs

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

- For the non-probabilistic operators:
  - Sat(true) = S
  - Sat(a) = { $s \in S \mid a \in L(s)$ }
  - Sat($\neg\phi$) = S \ Sat(ϕ)
  - Sat($\phi_1 \wedge \phi_2$) = Sat(ϕ_1) $\cap$ Sat(ϕ_2)

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



PCTL until for DTMCs

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

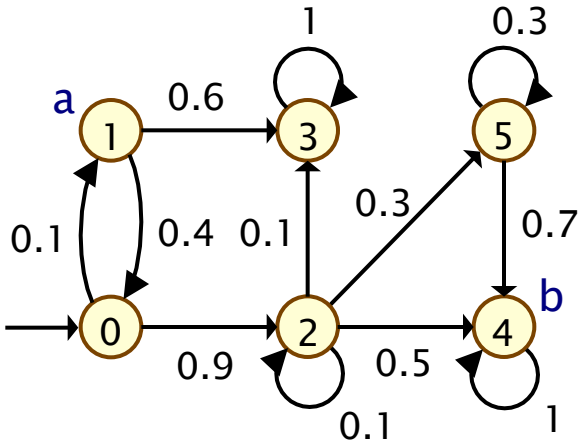
- We refer to the first phase as "precomputation"
  - two algorithms: Prob0 (for S^{no}) and Prob1 (for S^{yes})
  - algorithms work on underlying graph (probabilities irrelevant)
- Important for several reasons
  - reduces the set of states for which probabilities must be computed numerically
  - gives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

Precomputation – Prob0

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$

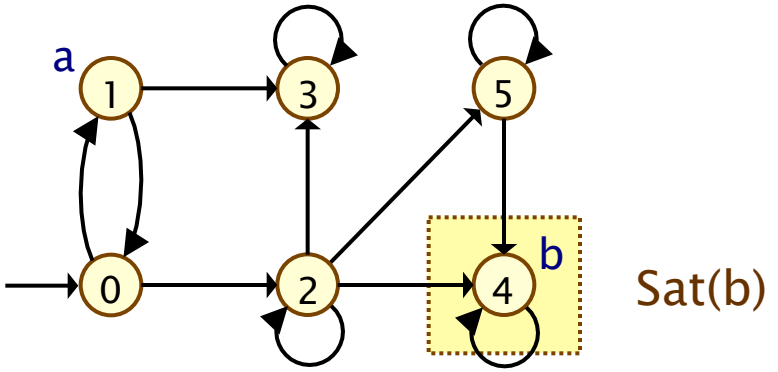


Precomputation – Prob0

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$

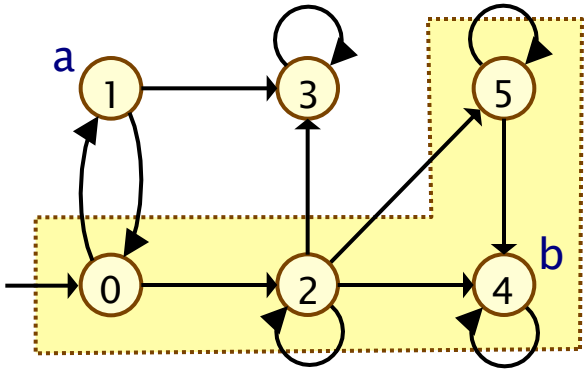


Precomputation – Prob0

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$



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

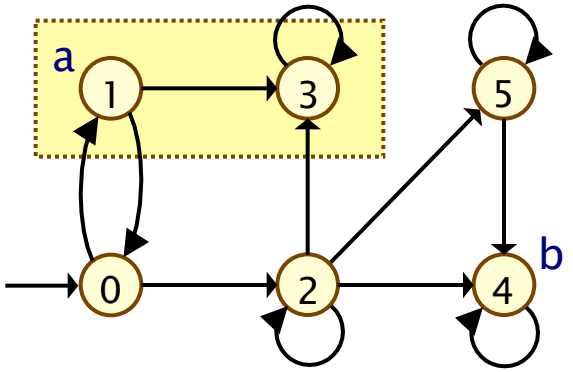
Precomputation – Prob0

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$

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

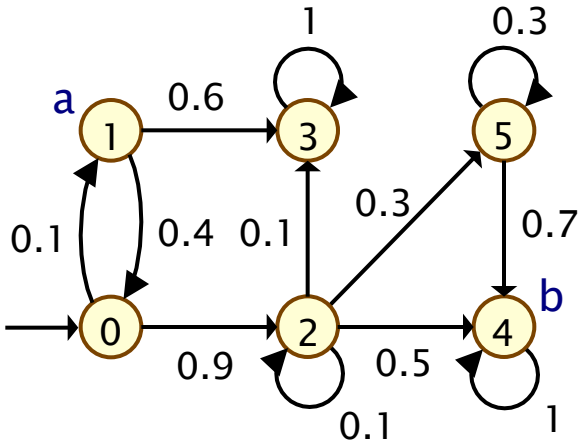


Precomputation – Prob1

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$



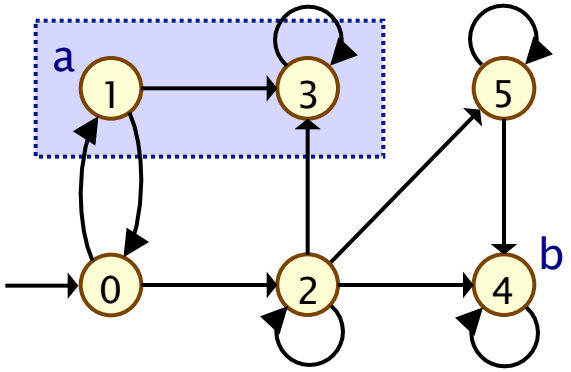
Precomputation – Prob1

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

Example:

$P_{\sim p}\ [\neg a\ U\ b\ ]$

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

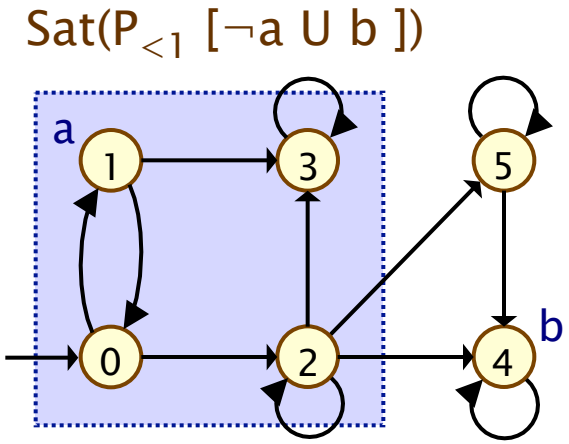


Precomputation – Prob1

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

Example:

$P_{\sim p}[\neg a\ U\ b\ ]$

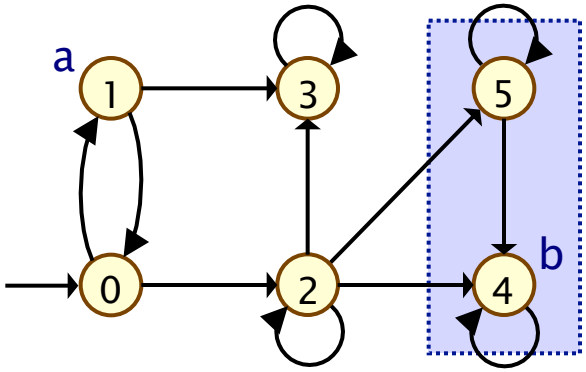


Precomputation – Prob1

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

Example:

$P_{\sim p}\ [\neg a\ U\ b\ ]$



$S^{yes} =$

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

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:

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

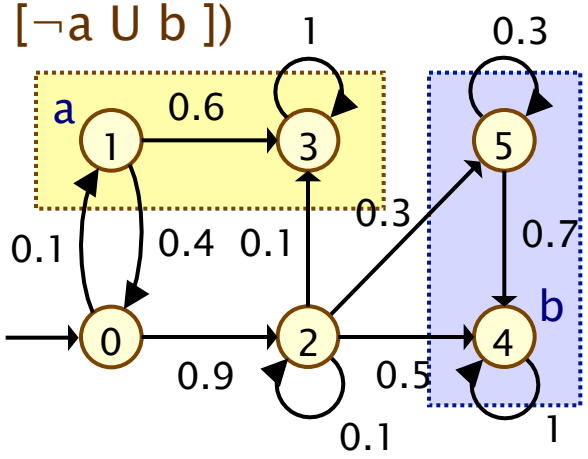
  - can be reduced to a system in $|S^?|$ unknowns instead of $|S|$ where $S^? = S \setminus (S^{yes} \cup S^{no})$

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

PCTL until – linear equations

- Example: $P_{\sim p}$ [¬a U b]
- Let x_s = Prob(s, ¬a U b)

S^{no} = Sat($P_{\leq 0}$ [¬a U b])



S^{yes} = Sat($P_{\geq 1}$ [¬a U b])

$x_1 = x_3 = 0$

$x_4 = x_5 = 1$

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

$x_0 = 0.1x_1 + 0.9x_2 = 0.8$

PCTL model checking – Summary

- Computation of set Sat(Φ) for DTMC D and PCTL formula Φ
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation
  - complexity is linear in $|\Phi|$ and polynomial in $|S|$

- Probabilistic operator P:
  - $X\ \Phi$: one matrix-vector multiplication, $O(|S|^2)$
  - $\Phi_1\ U^{\leq k}\ \Phi_2$: k matrix-vector multiplications, $O(k|S|^2)$
  - $\Phi_1\ U\ \Phi_2$: linear equation system, at most $|S|$ variables, $O(|S|^3)$

Overview

- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

Beyond PCTL

- PCTL, although useful in practice, has limited expressivity
  - essentially: probability of reaching states in X, passing only through states in Y, and (possibly) within k time-steps
- More expressive logics can be used, for example:
  - LTL, the non-probabilistic linear-time temporal logic
  - PCTL* [ASB+95,BdA95] which subsumes both PCTL and LTL
- Both allow combinations of temporal operators
  - e.g. for liveness: $P_{\sim p}$ [G F ϕ] – "…always eventually ϕ"
- Model checking algorithms exist (but more expensive)
  - translate to Rabin automata, construct product DTMC, graph algorithms (BSCCs) + probabilistic reachability
- Another direction: extend DTMCs with costs and rewards…

Costs and rewards

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

Reward-based properties

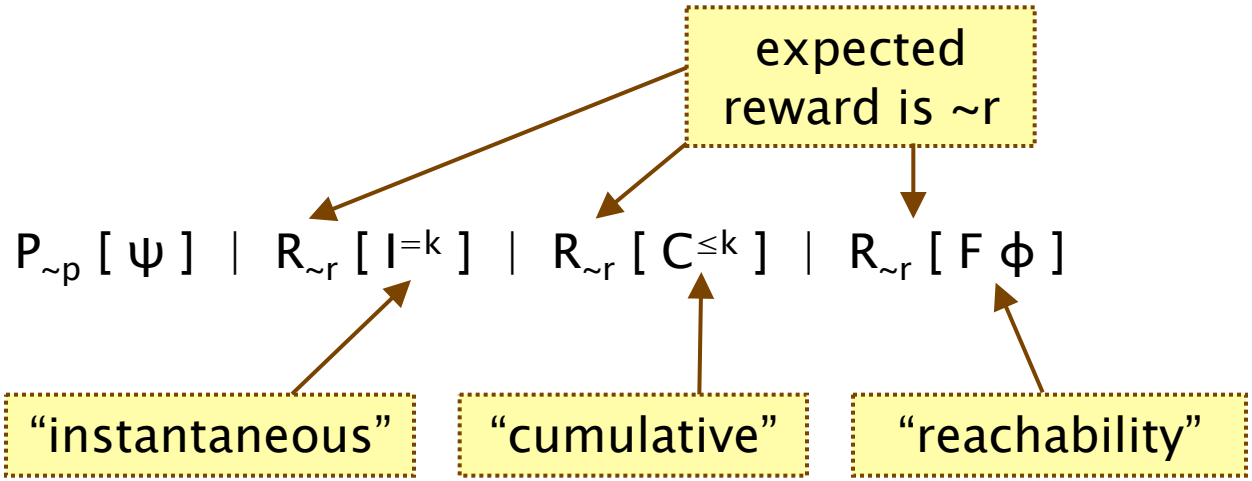
- Properties of DTMCs augmented with rewards
  - allow a wide range of quantitative measures of the system
  - basic notion: expected value of rewards
  - formal property specifications will be in an extension of PCTL
- More precisely, we use two distinct classes of property…
- Instantaneous properties
  - the expected value of the reward at some time point
- Cumulative properties
  - the expected cumulated reward over some period

DTMC reward structures

- For a DTMC $(S, s_{init}, \mathbf{P}, L)$, 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

PCTL and rewards

- Extend PCTL to incorporate reward-based properties
  - add an R operator, which is similar to the existing P operator
  - ϕ ::= … | $P_{\sim p}[\psi]$ | $R_{\sim r}[I^{=k}]$ | $R_{\sim r}[C^{\leq k}]$ | $R_{\sim r}[F\,\phi]$



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

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

Types of reward formulas

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

Reward formula semantics

- Formal semantics of the three reward operators
  - based on random variables over (infinite) paths
- Recall:
  - $s \models P_{\sim p}[\psi] \Leftrightarrow Pr_s\{\omega \in Path(s) \mid \omega \models \psi\} \sim p$
- For a state s in the DTMC:
  - $s \models R_{\sim r}[I^{=k}] \Leftrightarrow Exp(s, X_{I=k}) \sim r$
  - $s \models R_{\sim r}[C^{\leq k}] \Leftrightarrow Exp(s, X_{C\leq k}) \sim r$
  - $s \models R_{\sim r}[F\,\Phi] \Leftrightarrow Exp(s, X_{F\Phi}) \sim r$

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

Reward formula semantics

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

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

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

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

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

Model checking reward properties

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
- Cumulative: $R_{\sim r}$ [$C^{\leq t}$]
  - variant of the method for computing bounded until probabilities
  - solution of recursive equations
- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to computing until probabilities
  - precomputation phase (identify infinite reward states)
  - then reduces to solving a system of linear equation
- For more details, see e.g. [KNP07a]

Overview

- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

Tools – Probabilistic model checkers

- PRISM (Probabilistic Symbolic Model Checker)
  - DTMCs, MDPs, CTMCs + rewards, [Birmingham/Oxford]
- MRMC (Markov Reward Model Checker)
  - DTMCs, CTMCs + reward extensions, [Twente/Aachen]
- LiQuor: LTL model checking for MDPs, Probmela language (probabilistic version of SPIN's Promela), [Dresden]

- Simulation-based probabilistic model checking:
  - APMC, Ymer (both based on PRISM language), VESTA
- Many other related tools/prototypes
  - RAPTURE, CADP, Möbius, APNN-Toolbox, SMART, GreatSPN, GRIP, CASPA, Premo, PASS, …

The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL), Linux/Unix/Mac/Windows/64-bit
- Modelling of:
  - DTMCs, MDPs, CTMCs + costs/rewards
- Verification of:
  - PCTL, CSL + extensions + costs/rewards
- Features:
  - high-level modelling language
  - wide range of model analysis methods
  - graphical user interface, simulator/debugger, graph plotting
  - efficient symbolic (BDD-based) implementation
- See: www.prismmodelchecker.org

PRISM modelling language

- Simple, state-based language for DTMCs/MDPs/CTMCs
  - based on Reactive Modules [AH99]
- Modules (system components, composed in parallel)
- Variables (finite-valued, local or global)
- Guarded commands (labelled with probabilities/rates)
- Synchronisation (CSP-style) + process-algebraic operators (parallel composition, action hiding/renaming)

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

PRISM language example

```
// Herman's self-stabilisation algorithm [Her90]

dtmc // Algorithm is fully synchronous

module process1 // First of N=5 symmetric processes

   x1 : [0..1];  // One bit per process; xi=x(i-1) means proc i has a token
  [step] (x1=x5) -> 0.5 : (x1'=0) + 0.5 : (x1'=1);
   [step] !x1=x5 -> (x1'=x5);

endmodule

// Add further processes through renaming
module process2 = process1 [ x1=x2, x5=x1 ] endmodule
module process3 = process1 [ x1=x3, x5=x2 ] endmodule
module process4 = process1 [ x1=x4, x5=x3 ] endmodule
module process5 = process1 [ x1=x5, x5=x4 ] endmodule

// Can start in any possible configuration
init true endinit
```

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]

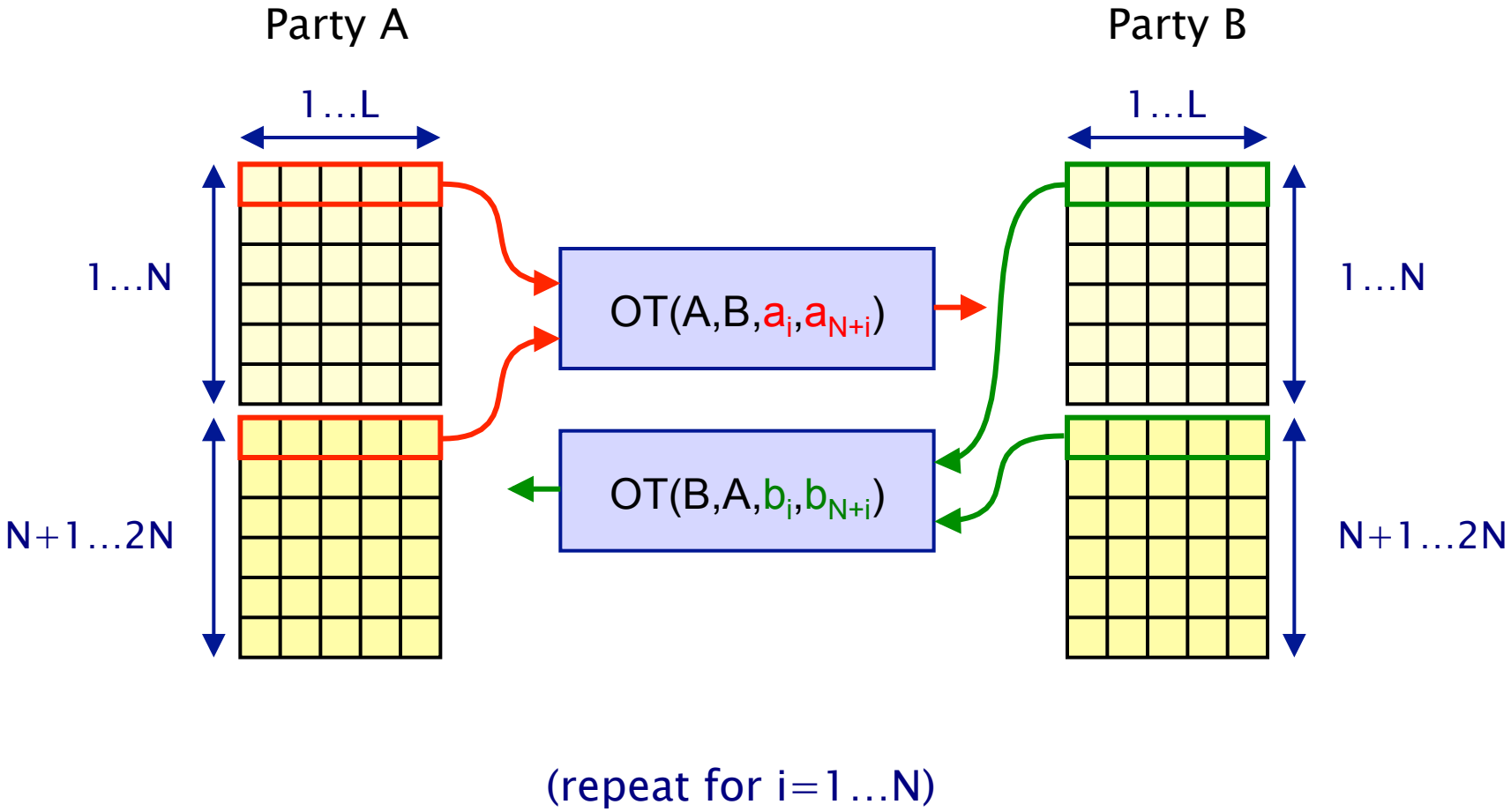
Contract signing – EGL protocol

- 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}) | i=1,…,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)
  - S sends x and y to 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

Contract signing – EGL protocol

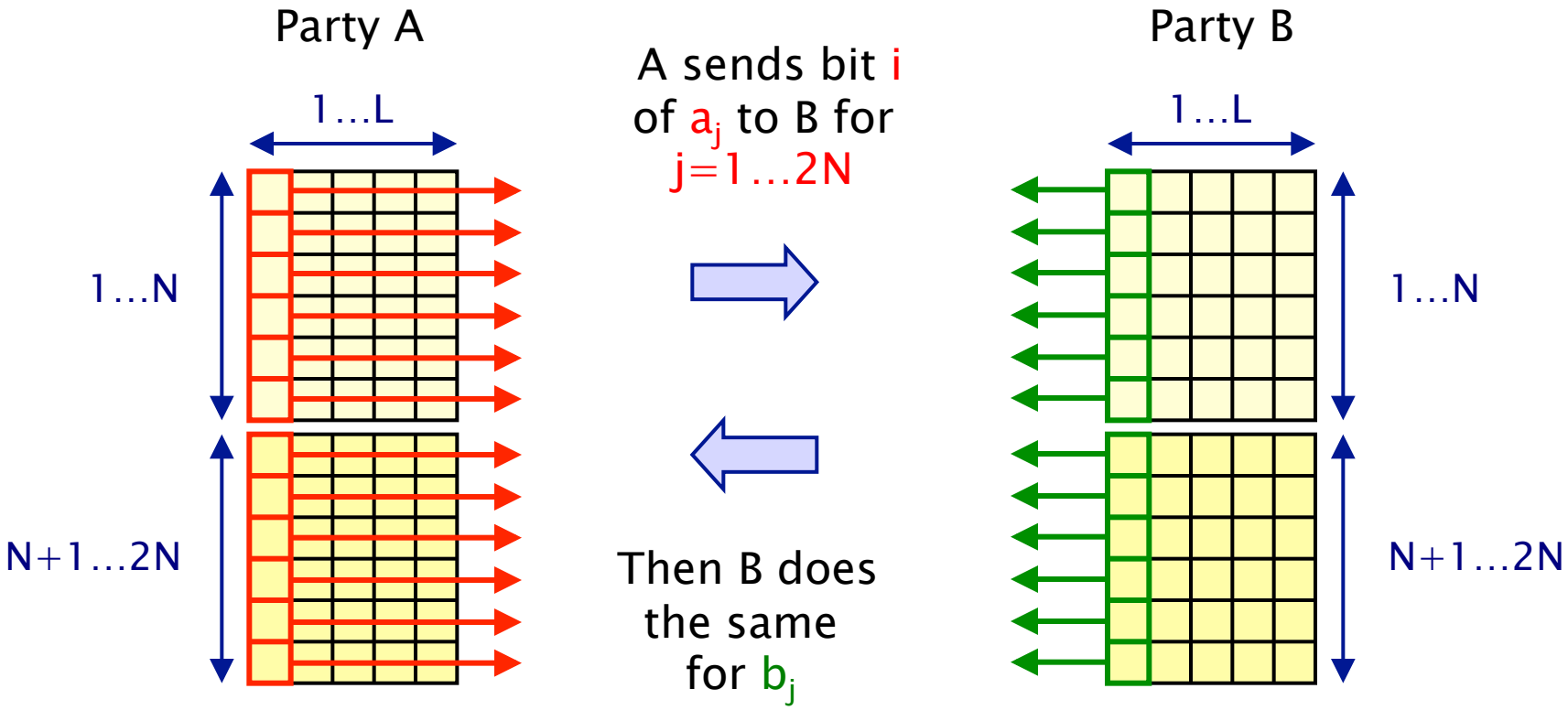
(step 1)
for (i=1,…,N)
    OT(A, B, a_i, a_{N+i})
    OT(B, A, b_i, b_{N+i})
(step 2)
    for (i=1,…,L) (where L is the bit length of the secrets)
        for (j=1,…,2N)
            A transmits bit i of secret a_j to B
        for (j=1,…,2N)
            B transmits bit i of secret b_j to A

EGL protocol – Step 1



(repeat for i=1…N)

EGL protocol – Step 2



(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

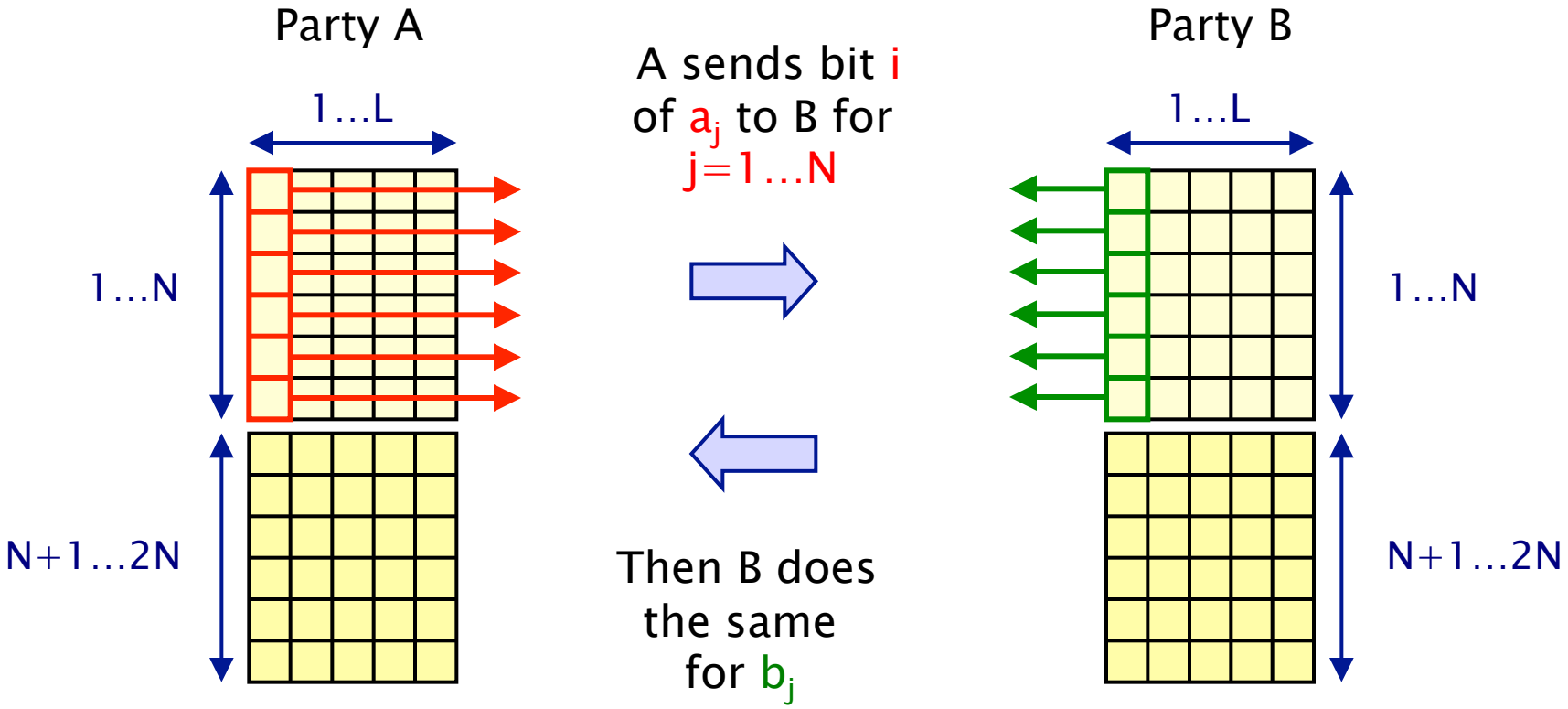
Contract signing – Results

- 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

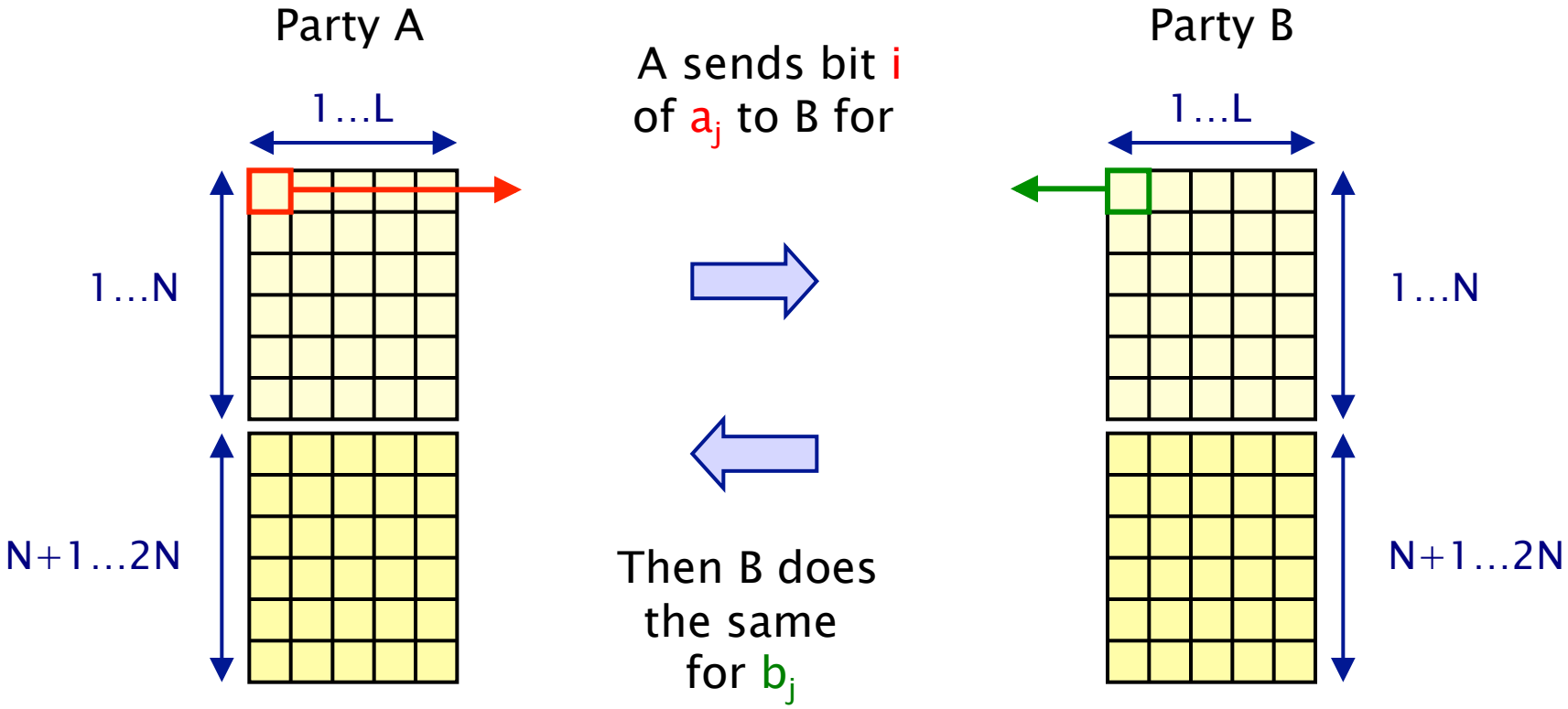


(after j=1…N, send j=N+1…2N)
(then repeat for i=1…L)

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

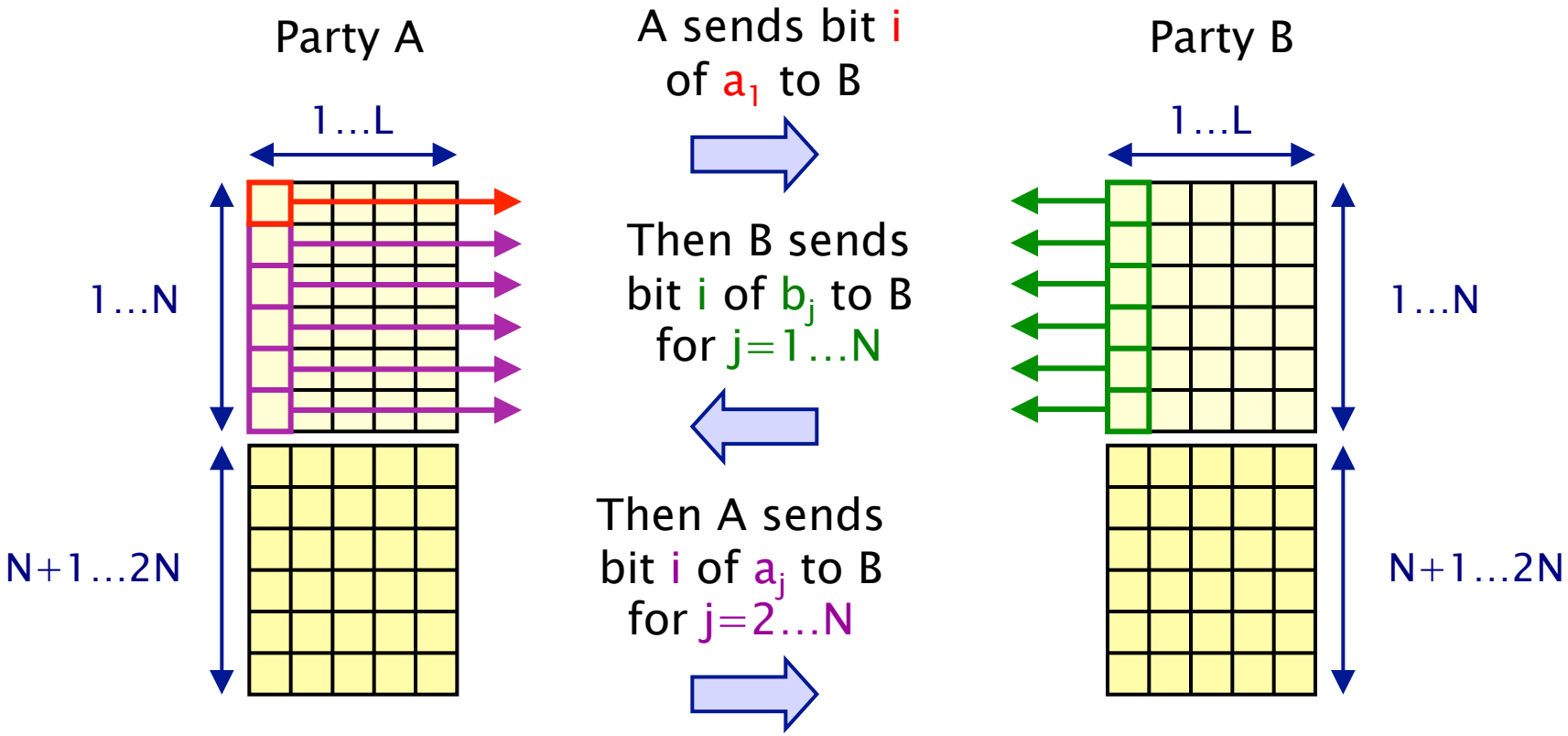


(repeat for j=1…N and for i=1…L)
(then send j=N+1…2N for i=1…L)

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

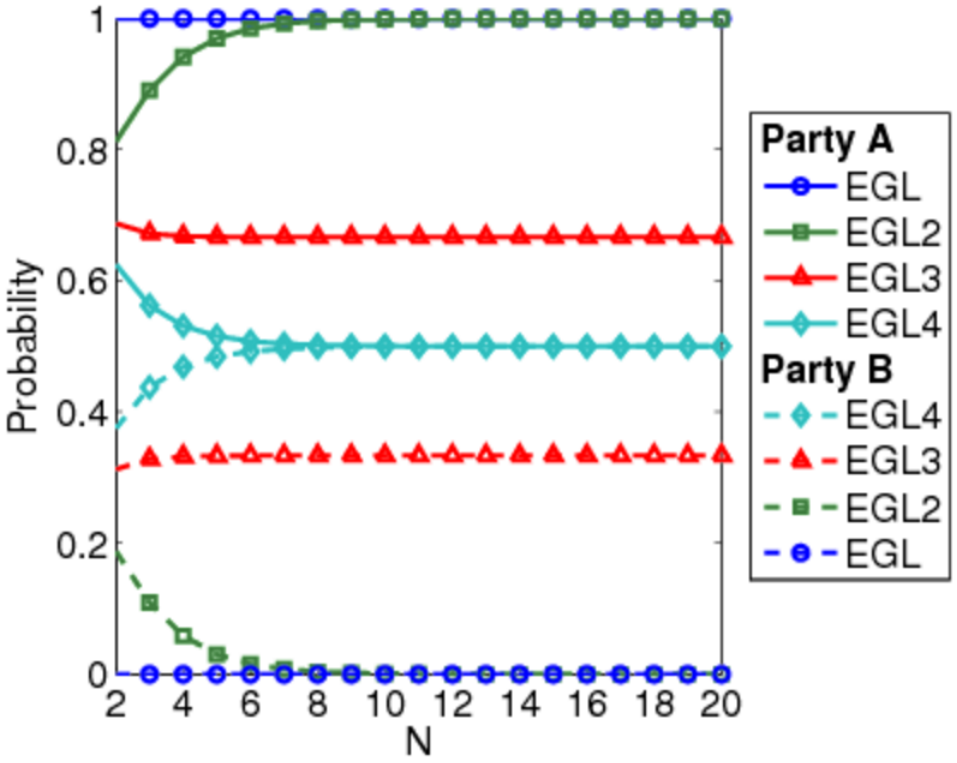
Modified step 2 for EGL4



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

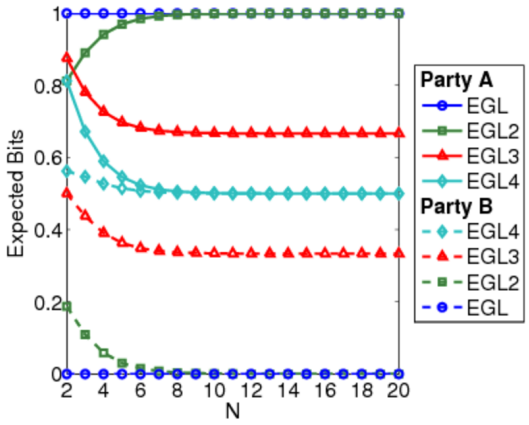
Contract signing – Results

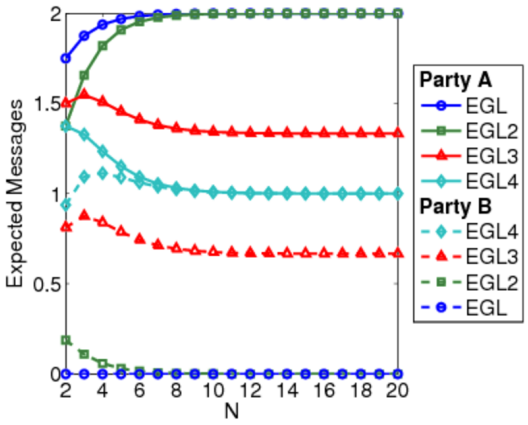
- The chance that the protocol is unfair
  - probability that one party gains knowledge first
  - $P_{=?}[\ F\ know_B \wedge \neg know_A\ ]$ and $P_{=?}[\ F\ know_A \wedge \neg know_B\ ]$

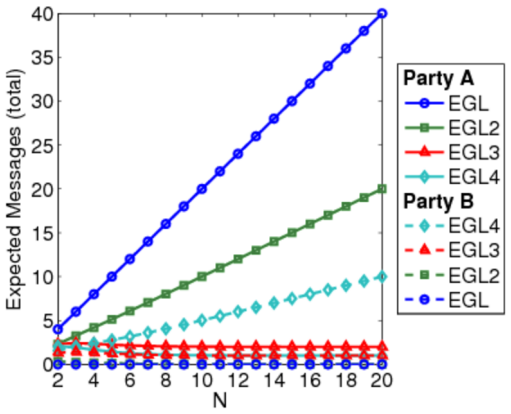


Contract signing – More properties

- (1) How unfair the protocol is to each party
  - expected number of bits that a party needs to know a pair once the other party knows a pair
- (2) 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
- (3) 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







Overview

- Discrete-time Markov chains (DTMCs)
- Properties of DTMCs: The logic PCTL
- PCTL model checking for DTMCs
- Beyond PCTL: Costs and rewards
- Tool support + A case study: contract signing
- Adding nondeterminism: Markov decision processes (MDPs)

Other models

- What's missing from DTMCs?
- Nondeterminism
  - Markov decision processes (MDPs)…
- Real-time
  - continuous-time Markov chains (CTMCs)
    - exponentially distributed delays
  - probabilistic timed automata (PTAs)
    - real-valued clocks, discrete probabilistic choice, nondeterminism

Nondeterminism

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

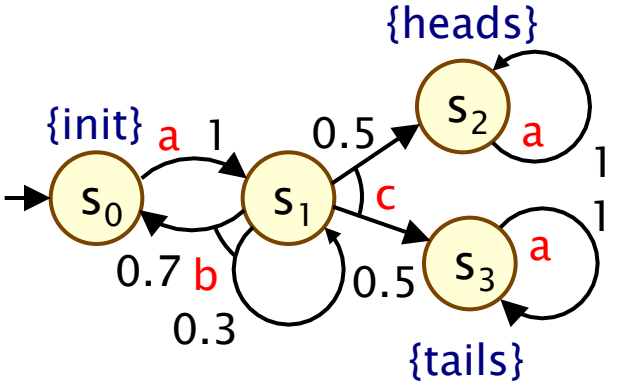
- Concurrency – scheduling of parallel components
  - e.g. randomised distributed algorithms – multiple probabilistic processes operating asynchronously

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

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

Markov decision processes

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

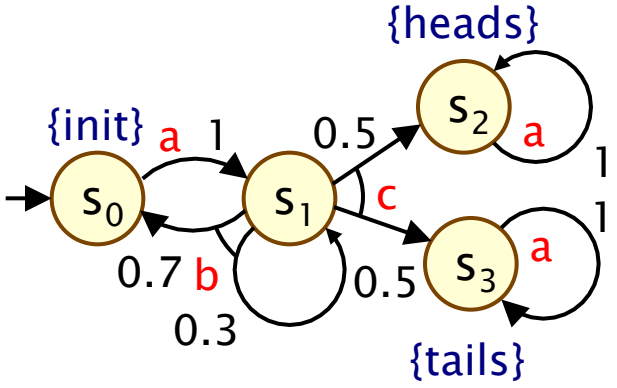


Markov decision processes

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

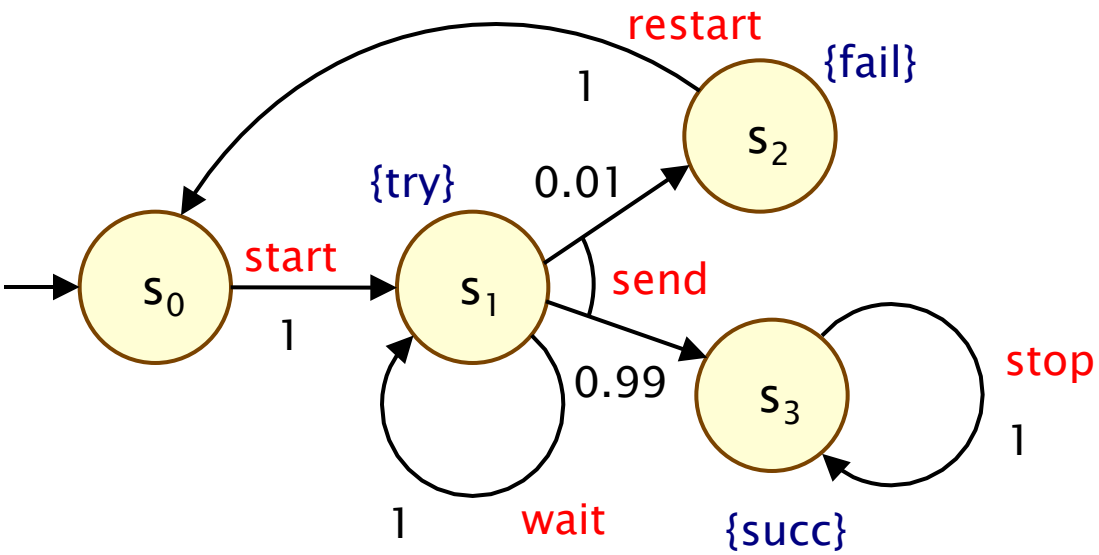
    where Act is a set of actions and Dist(S) is the set of discrete probability distributions over the set S
  - $L : S \rightarrow 2^{AP}$ is a labelling with atomic propositions

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



Simple MDP example

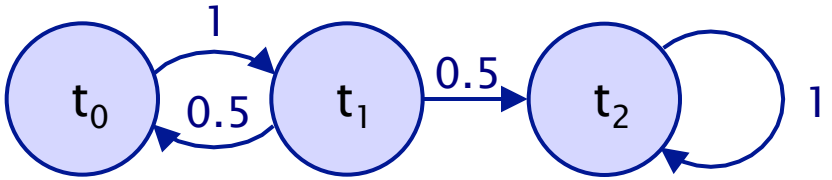
- Modification of the simple DTMC communication protocol
  - after one step, process starts trying to send a message
  - then, a nondeterministic choice between: (a) waiting a step because the channel is unready; (b) sending the message
  - if the latter, with probability 0.99 send successfully and stop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

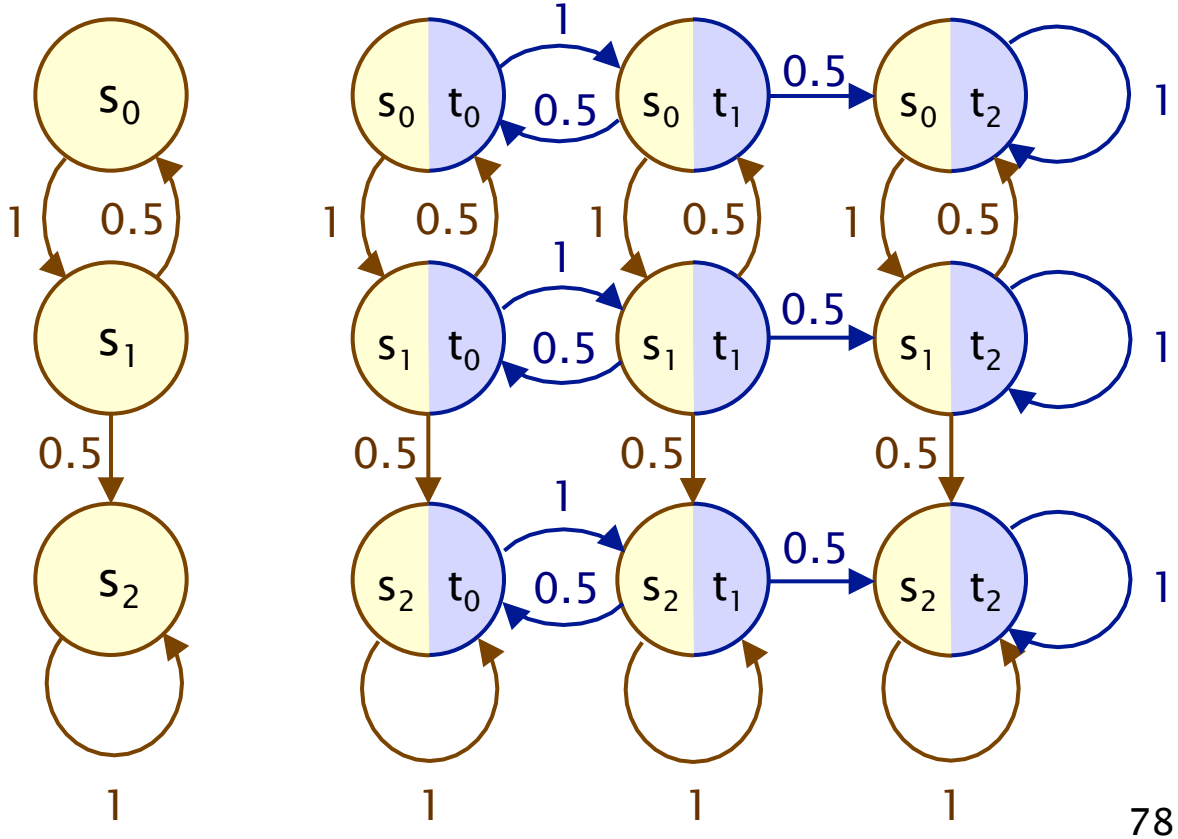


Example – Parallel composition

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 $(a_i,\mu_i) \in \mathbf{Steps}(s_i)$ 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
  - note that a path resolves both types of choices: nondeterministic and probabilistic

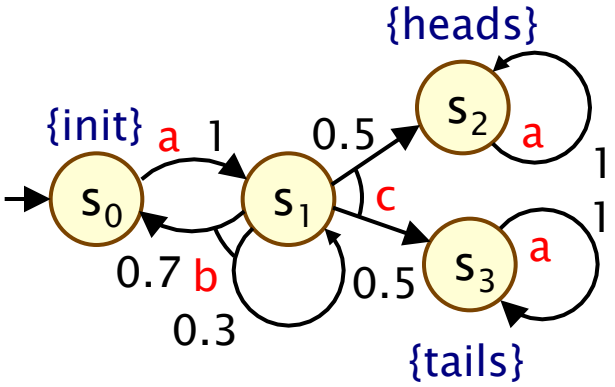
- To consider the probability of some behaviour of the MDP
  - first need to resolve the nondeterministic choices
  - …which results in a DTMC
  - …for which we can define a probability measure over paths

Adversaries

- An adversary resolves nondeterministic choice in an MDP
  - adversaries are also known as “schedulers” or “policies”
- Formally:
  - an adversary A of an MDP M is a function mapping every finite path $\omega = s_0(a_1,\mu_1)s_1 \ldots s_n$ to an element of **Steps**(s_n)
- For each A can define a probability measure Pr^A_s over paths
  - constructed through an infinite state DTMC $(Path^A_{fin}(s),s,\mathbf{P}^A_s)$
  - states of the DTMC are the finite paths of A starting in state s
  - initial state is s (the path starting in s of length 0)
  - $\mathbf{P}^A_s(\omega,\omega') = \mu(s)$ if $\omega' = \omega(a, \mu)s$ and $A(\omega)=(a,\mu)$
  - $\mathbf{P}^A_s(\omega,\omega') = 0$ otherwise

Adversaries – Examples

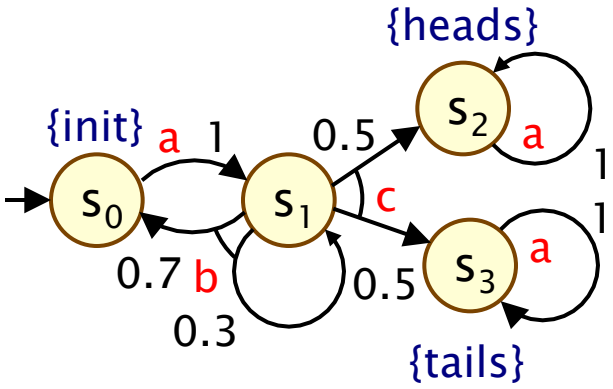
- Consider the simple MDP below
  - note that s_1 is the only state for which $|\mathbf{Steps}(s)| > 1$
  - i.e. s_1 is the only state for which an adversary makes a choice
  - let μ_b and μ_c denote the probability distributions associated with actions b and c in state s_1

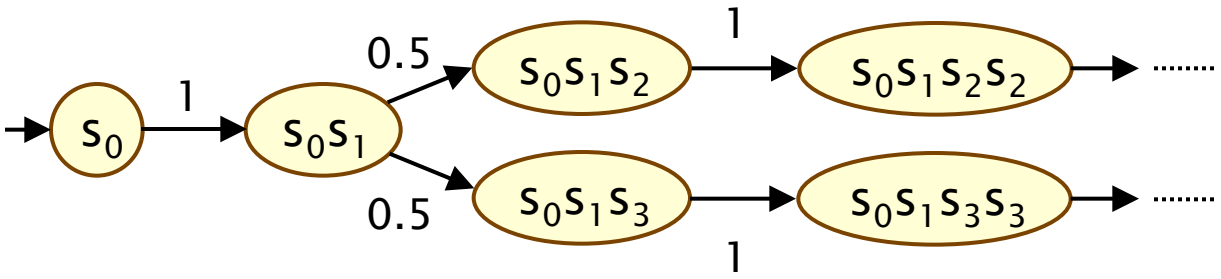


- Adversary A_1
  - picks action c the first time
  - $A_1(s_0s_1)=(c,\mu_c)$
- Adversary A_2
  - picks action b the first time, then c
  - $A_2(s_0s_1)=(b,\mu_b)$, $A_2(s_0s_1s_1)=(c,\mu_c)$, $A_2(s_0s_1s_0s_1)=(c,\mu_c)$

Adversaries – Examples

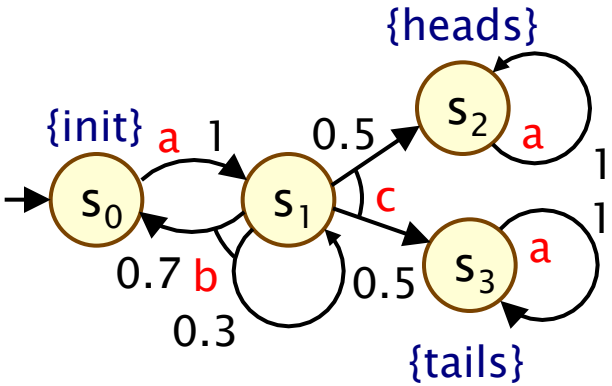
- Fragment of DTMC for adversary A_1
  - A_1 picks action c the first time

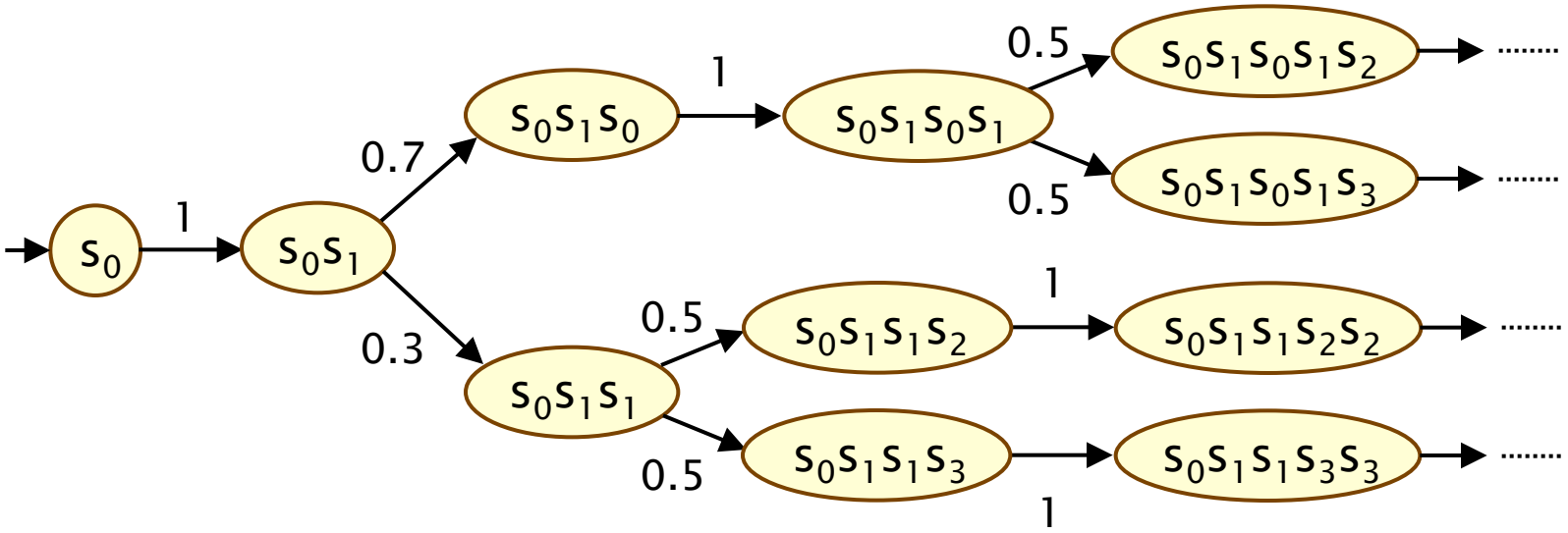




Adversaries – Examples

- Fragment of DTMC for adversary A_2
  - A_2 picks action b, then c

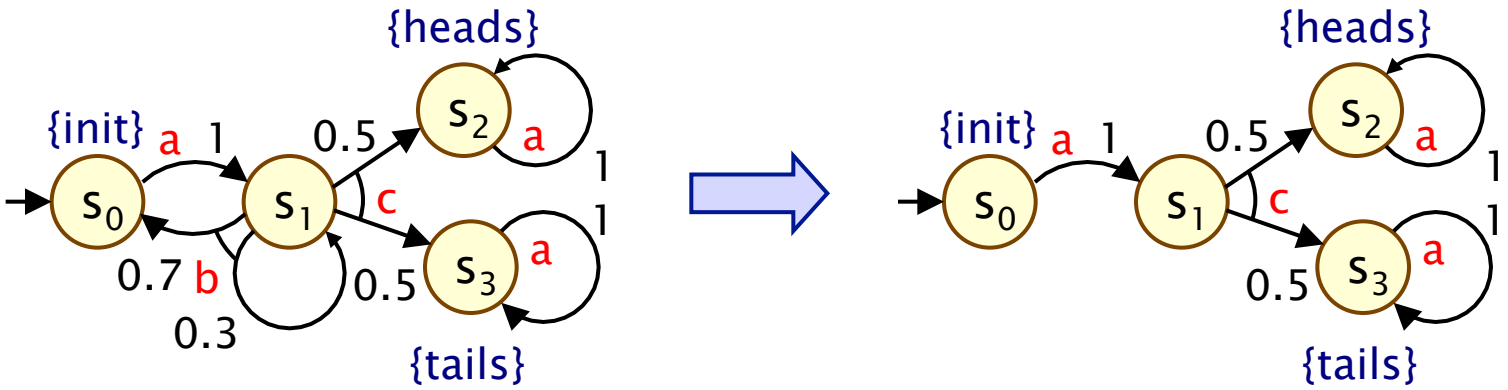




Simple adversaries

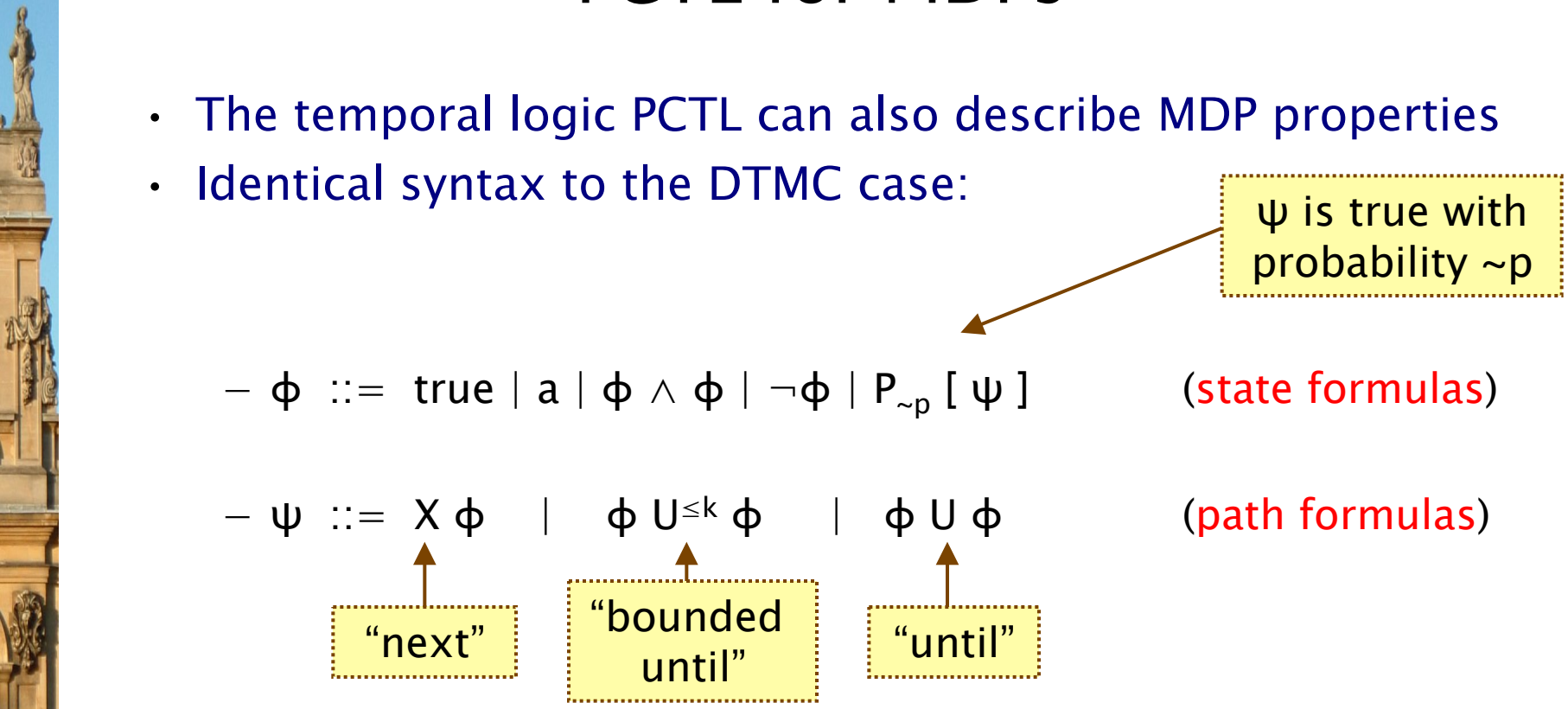
- Simple adversaries always pick same choice in a given state
  - formally, for adversary A:
  - $A(s_0(a_1,\mu_1)s_1...s_n)$ depends only on s_n
  - resulting DTMC can be mapped to a |S|-state DTMC

- From previous example:
  - adversary A_1 (picks c in s_1) is simple, A_2 is not



PCTL for MDPs

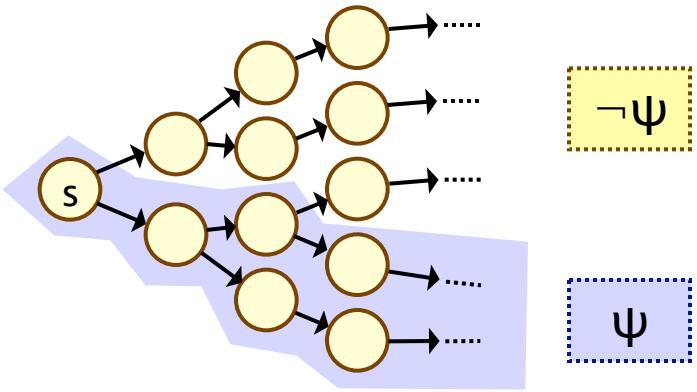
- The temporal logic PCTL can also describe MDP properties
- Identical syntax to the DTMC case:



- Semantics are also the same as DTMCs for:
  - atomic propositions, logical operators, path formulas

PCTL semantics for MDPs

- Semantics of the probabilistic operator P
  - can only define probabilities for a specific adversary A
  - $s \models P_{\sim p}[\psi]$ means "the probability, from state s, that ψ is true for an outgoing path satisfies ~p for all adversaries A"
  - formally $s \models P_{\sim p}[\psi] \Leftrightarrow Prob^A(s, \psi) \sim p$ for all adversaries A
  - where $Prob^A(s, \psi) = Pr^A_s \{ \omega \in Path^A(s) \mid \omega \models \psi \}$



$Prob^A(s, \psi) \sim p$

Minimum and maximum probabilities

- Letting:
  - $p_{max}(s, \psi) = \sup_A \text{Prob}^A(s, \psi)$
  - $p_{min}(s, \psi) = \inf_A \text{Prob}^A(s, \psi)$
- We have:
  - if $\sim \in \{\geq, >\}$, then $s \models P_{\sim p}[\ \psi\ ] \quad \Leftrightarrow \quad p_{min}(s, \psi) \sim p$
  - if $\sim \in \{<, \leq\}$, then $s \models P_{\sim p}[\ \psi\ ] \quad \Leftrightarrow \quad p_{max}(s, \psi) \sim p$
- 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
- Crucial result for model checking PCTL on MDPs
  - simple adversaries suffice, i.e. there are always simple adversaries A_{min} and A_{max} for which:
  - $\text{Prob}^{A_{min}}(s, \psi) = p_{min}(s, \psi)$ and $\text{Prob}^{A_{max}}(s, \psi) = p_{max}(s, \psi)$

Quantitative properties

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

- Example: CSMA/CD protocol
  - "min/max probability

    that a message is sent

    within the deadline"

Other classes of adversary

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

Some real PCTL examples

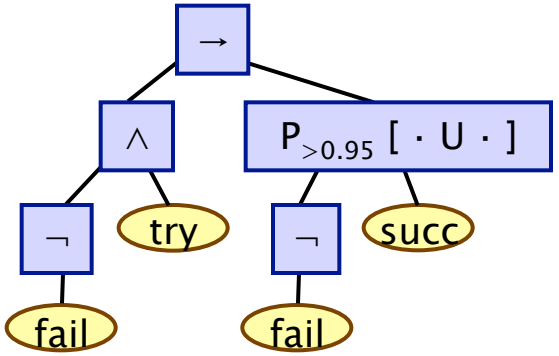
- Byzantine agreement protocol
  - $Pmin_{=?}$ [F (agreement $\wedge$ rounds$\leq$2)]
  - "what is the minimum probability that agreement is reached within two rounds?"
- CSMA/CD communication protocol
  - $Pmax_{=?}$ [F collisions=k]
  - "what is the maximum probability of k collisions?"
- Self-stabilisation protocols
  - $Pmin_{=?}$ [$F^{\leq t}$ 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: $Sat(\phi) = \{ s \in S \mid s \models \phi \}$ = set of states satisfying ϕ

- As for PCTL model checking for DTMCs
  - sometimes check: $s \models \phi$ for all $s \in S$, sometimes: $s_{init} \models \phi$
  - or compute quantitative results
    - e.g. compute result of $Pmax=?\ [\ F^{\leq k}\ error\ ]$ for $0 \leq k \leq 100$

- Basic algorithm proceeds by induction on parse tree of the PCTL formula ϕ



PCTL model checking for MDPs

- Only need to consider $P_{\sim p}$ [ψ] formulas
  - reduces to computation of $p_{min}(s, \psi)$ or $p_{max}(s, \psi)$ for all $s \in S$
  - dependent on whether $\sim \in \{\geq, >\}$ or $\sim \in \{<, \leq\}$
- Here, we cover the algorithm for computing $p_{min}(s, \psi)$
  - i.e. the case where $\sim \in \{\geq, >\}$
  - computation of $p_{max}(s, \psi)$ is very similar
- Focus on until formulas, i.e. $p_{min}(s, \phi_1 \, U \, \phi_2)$
  - next ($X \, \phi$) and bounded until ($\phi_1 \, U^{\leq k} \, \phi_2$) are simple extensions of DTMC case
  - see e.g. [BdA95], [KNP04a] for further details

PCTL until for MDPs

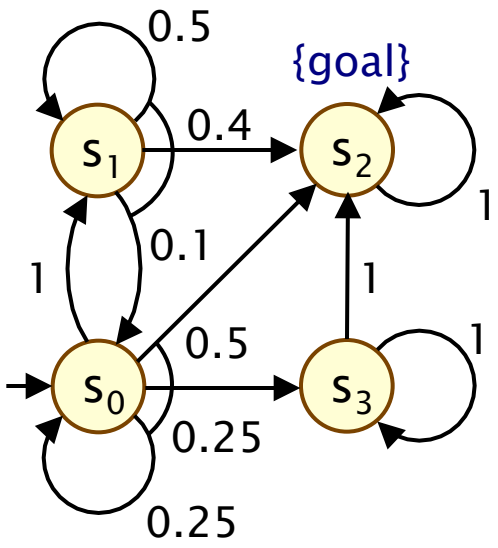
- Computation of probabilities $p_{min}(s, \phi_1 \, U \, \phi_2)$ for all $s \in S$
- First identify all states where the probability is 1 or 0
  - "precomputation" algorithms, yielding sets S^{yes}, S^{no}
- Then compute (min) probabilities for remaining states ($S^{?}$)
  - either: solve linear optimisation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

Example:

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

$\equiv$

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

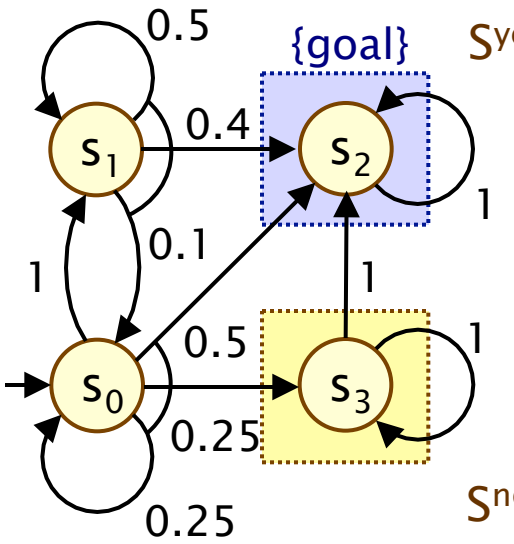


PCTL until – Precomputation

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

Example:

$P_{\geq p} [F \, goal]$



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

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

Method 1 – Linear optimisation problem

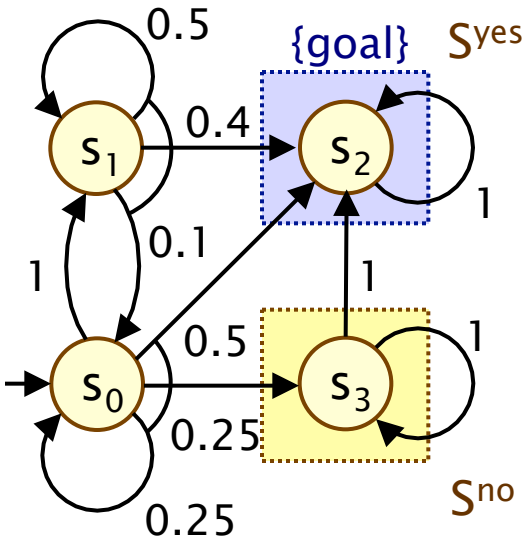
- Probabilities $p_{min}(s, \phi_1 \, U \, \phi_2)$ for remaining states in the set $S^? = S \setminus (S^{yes} \cup S^{no})$ can be obtained as the unique solution of the following linear optimisation problem:

$$\text{maximize } \sum_{s \in S^?} x_s \text{ subject to the constraints :}$$
$$x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s')$$
$$\text{for all } s \in S^? \text{ and for all } (a, \mu) \in \text{Steps}(s)$$

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

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

PCTL until – Example



Let $x_i = p_{min}(s_i, F\ goal)$

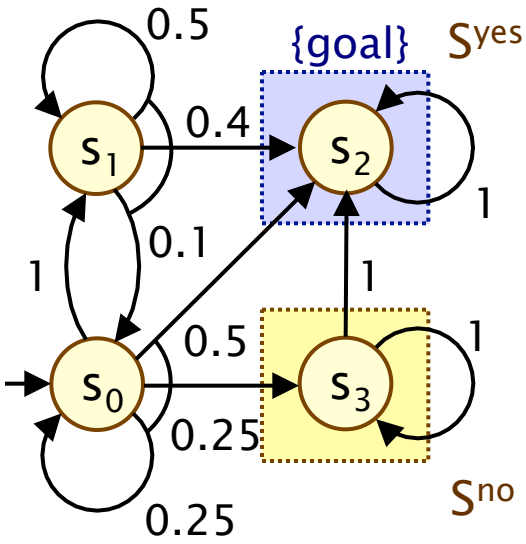
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

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

Maximise x_0+x_1 subject to constraints:

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

PCTL until – Example



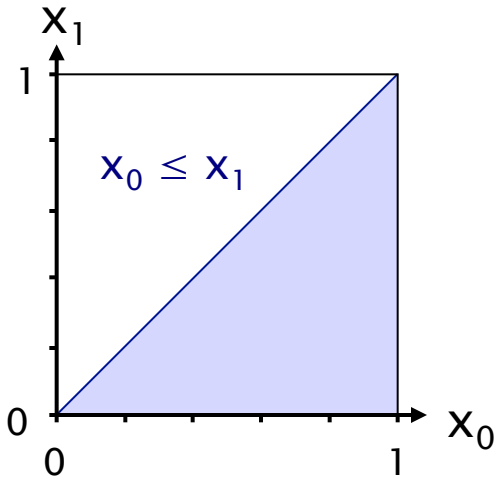
Let $x_i = p_{min}(s_i, F\ goal)$

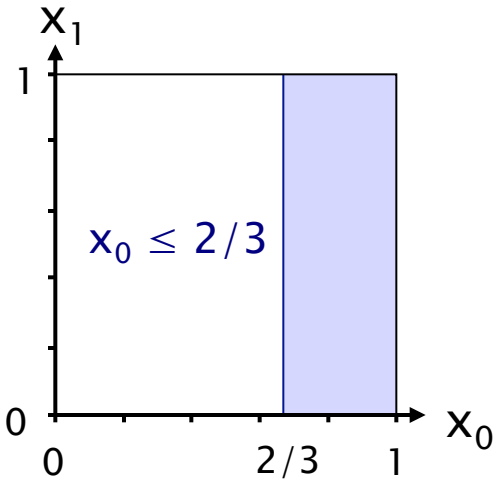
S^{yes}: $x_2=1$, S^{no}: $x_3=0$

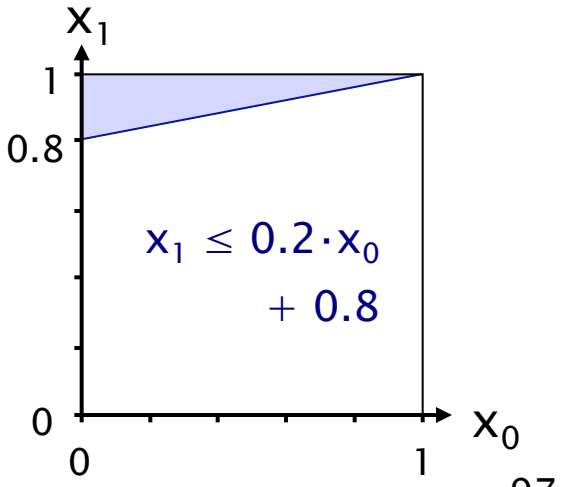
For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

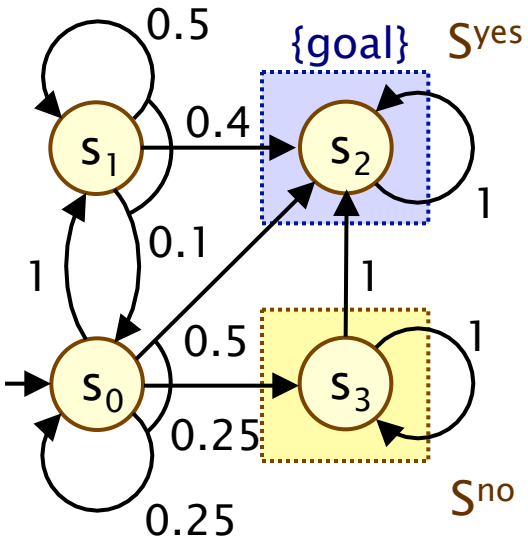
- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$







PCTL until – Example



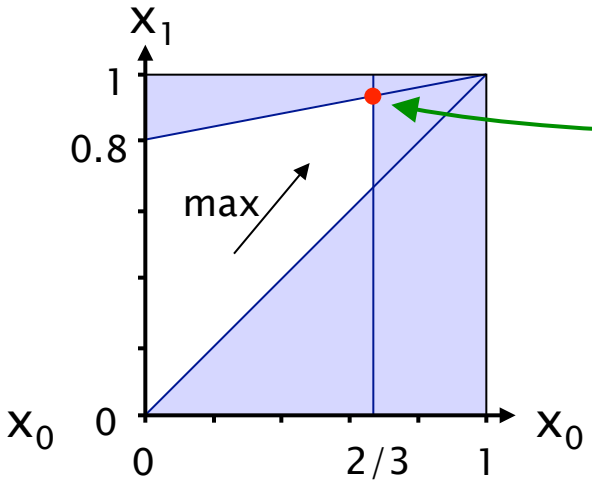
Let $x_i = p_{min}(s_i, F\ goal)$

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

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

Maximise x_0+x_1 subject to constraints:

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



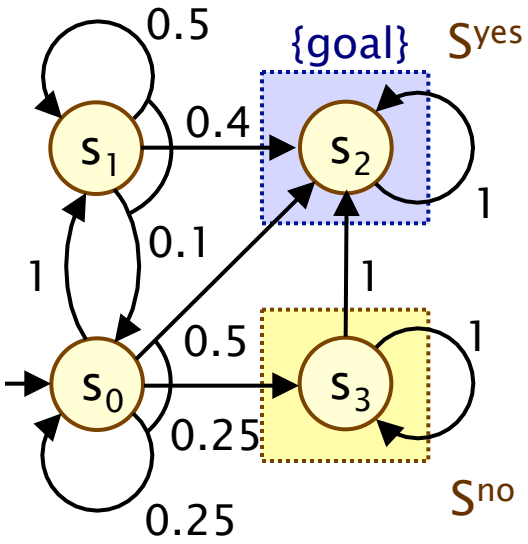
Solution:

(x_0, x_1)

=

(2/3, 14/15)

PCTL until – Example



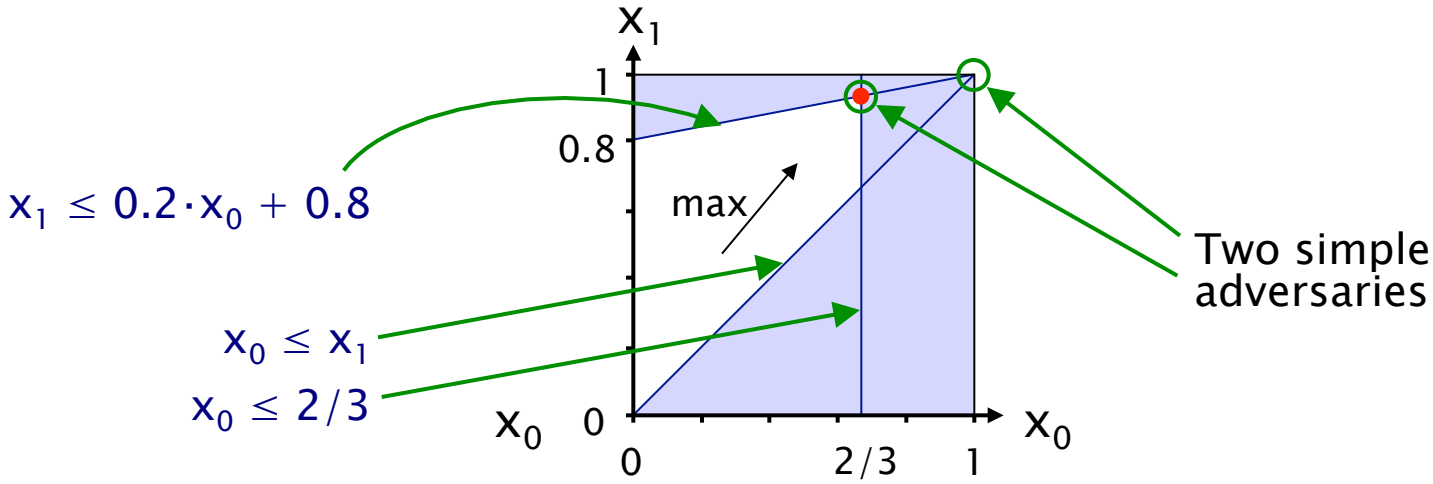
Let $x_i = p_{min}(s_i, F\ goal)$

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

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

Maximise x_0+x_1 subject to constraints:

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



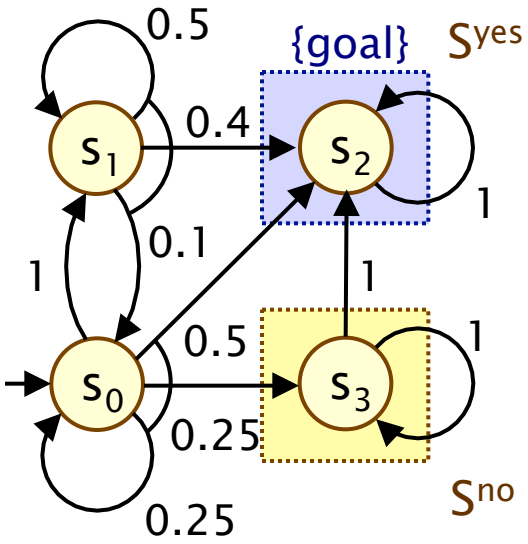
Method 2 – Iterative solution

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

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

- This forms the basis for an (approximate) iterative solution
  - iterations terminated when solution converges sufficiently
  - equivalent to well-known "value iteration" method for MDPs

PCTL until – Example



Compute: $p_{min}(s_i, F\ goal)$

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

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

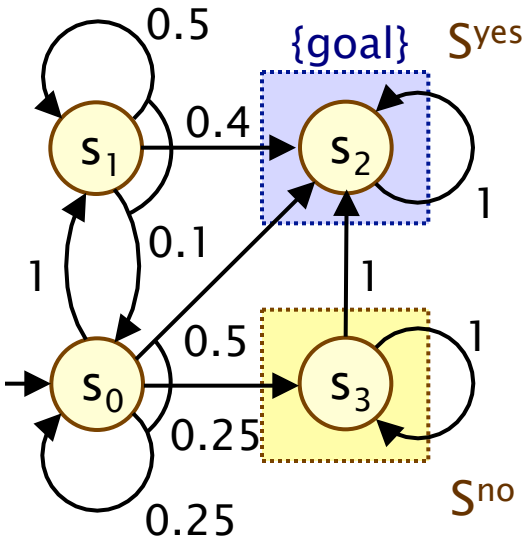
n=0: $[\ 0, 0, 1, 0\ ]$

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

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

n=3: …

PCTL until – Example



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

n=0: [0.000000, 0.000000, 1, 0]

n=1: [0.000000, 0.400000, 1, 0]

n=2: [0.400000, 0.600000, 1, 0]

n=3: [0.600000, 0.740000, 1, 0]

n=4: [0.650000, 0.830000, 1, 0]

n=5: [0.662500, 0.880000, 1, 0]

n=6: [0.665625, 0.906250, 1, 0]

n=7: [0.666406, 0.919688, 1, 0]

n=8: [0.666602, 0.926484, 1, 0]

n=9: [0.666650, 0.929902, 1, 0]

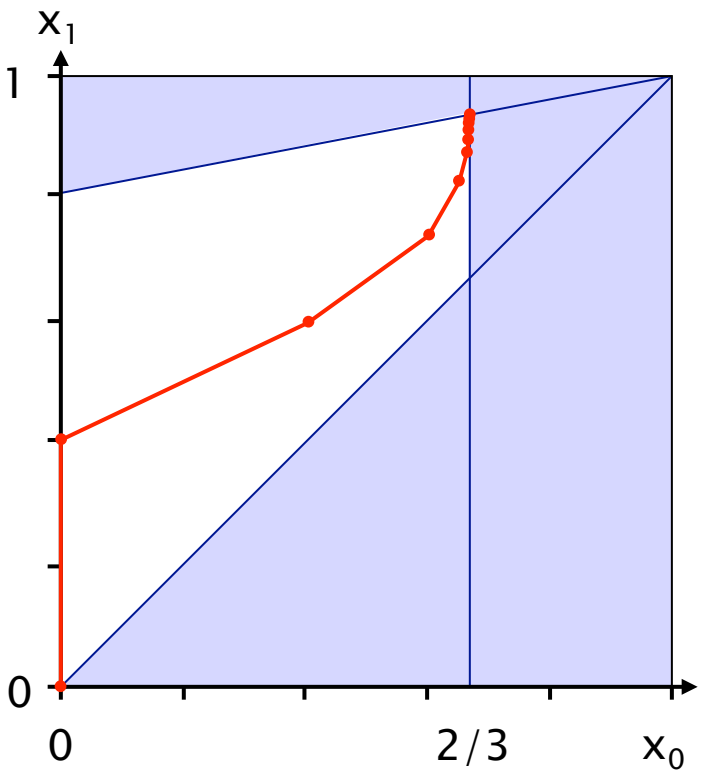
…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

$\approx$ [2/3, 14/15, 1, 0]

PCTL until – Example



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

n=0: [0.000000, 0.000000, 1, 0]

n=1: [0.000000, 0.400000, 1, 0]

n=2: [0.400000, 0.600000, 1, 0]

n=3: [0.600000, 0.740000, 1, 0]

n=4: [0.650000, 0.830000, 1, 0]

n=5: [0.662500, 0.880000, 1, 0]

n=6: [0.665625, 0.906250, 1, 0]

n=7: [0.666406, 0.919688, 1, 0]

n=8: [0.666602, 0.926484, 1, 0]

n=9: [0.666650, 0.929902, 1, 0]

…

n=20: [0.666667, 0.933332, 1, 0]

n=21: [0.666667, 0.933332, 1, 0]

$\approx$ [2/3, 14/15, 1, 0]

Costs and rewards

- Can use costs and rewards in similar fashion to DTMCs:
- Augment MDPs with rewards (or costs)
  - (but often assign to states/actions, not states/transitions)
- Extend logic PCTL with R operator
  - semantics extended in same way as P operator
  - e.g. $s \models R_{\sim r}[F\ \Phi] \Leftrightarrow Exp^{A}(s, X_{F\Phi}) \sim r$ for all adversaries A
  - quantitative properties: $Rmin_{=?}[\ldots]$ and $Rmax_{=?}[\ldots]$
- Examples:
  - "the minimum expected queue size after exactly 90 seconds"
  - "the maximum expected power consumption over one hour"
  - the maximum expected time for the algorithm to terminate

Model checking MDP reward formulas

- Instantaneous: $R_{\sim r}$ [$I^{=k}$]
  - similar to the computation of bounded until probabilities
  - solution of recursive equations
- Cumulative: $R_{\sim r}$ [$C^{\leq k}$]
  - extension of bounded until computation
  - solution of recursive equations
- Reachability: $R_{\sim r}$ [F ϕ]
  - similar to the case for P operator and until
  - graph-based precomputation (identify ∞-reward states)
  - then linear optimization problem (or iterative solution)

MDP model checking – Summary

- Computation of set Sat(Φ) for MDP M and PCTL formula Φ
  - recursive descent of parse tree
  - combination of graph algorithms, numerical computation
  - complexity is linear in |Φ| and polynomial in |S|
  - S is states in MDP, assume |Steps(s)| is constant

- Probabilistic operator P:
  - X Φ : one matrix-vector multiplication, $O(|S|^2)$
  - $\Phi_1\ U^{\leq k}\ \Phi_2$: k matrix-vector multiplications, $O(k|S|^2)$
  - $\Phi_1\ U\ \Phi_2$: linear optimisation problem, polynomial in |S|
- Expected reward operator R
  - $I^{=k}$: k matrix-vector multiplications, $O(k|S|^2)$
  - $C^{\leq k}$: k iterations of matrix-vector multiplication + summation
  - F Φ : linear optimisation problem in at most |S| variables

Summary

- Probabilistic model checking
  - automated quantitative verification of stochastic systems
  - to model randomisation, failures, …
- Probabilistic models
  - discrete-time Markov chains (DTMCs)
  - Markov decision processes (MDPs)
- Property specifications:
  - probabilistic extensions of temporal logic, e.g. PCTL
  - expected value of costs/rewards
- Model checking algorithms
  - combination of graph-based algorithms, numerical computation (linear equations, linear optimisation, …)
- Tool support, case studies
  - PRISM, randomised contract signing algorithm

Further information

- Slides from full lecture course at:
  - www.prismmodelchecker.org/lectures/
  - DTMCs/MDPs/CTMCs/PTAs
  - case studies, implementation, advanced topics

- See also the PRISM web site:
  - www.prismmodelchecker.org
  - related publications
  - case study repository
  - tool download, tutorial, manual
  - and much more…