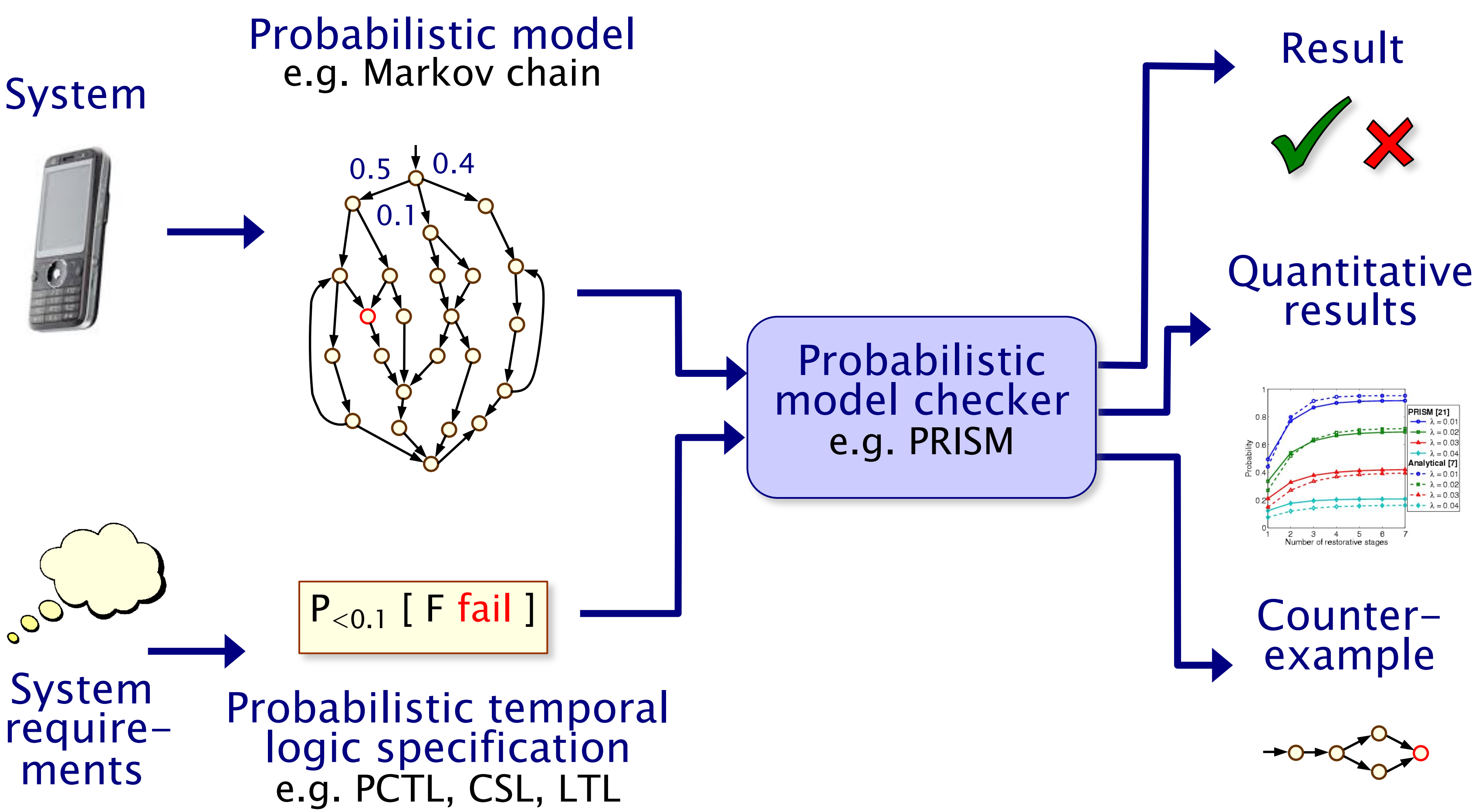
Lecture 5
PCTL Model Checking for DTMCs

Alessandro Abate

Department of Computer Science
University of Oxford

Probabilistic model checking



Overview

- PCTL model checking for DTMCs
- Computation of probabilities for PCTL formulae
  - next
  - bounded until
  - (unbounded) until
- Solving large systems of linear equations
  - 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 $D=(S,s_{init},\mathbf{P},L)$, PCTL formula ϕ
  - output: $Sat(\phi) = \{ s \in S \mid s \vDash \phi \}$ = set of states satisfying ϕ

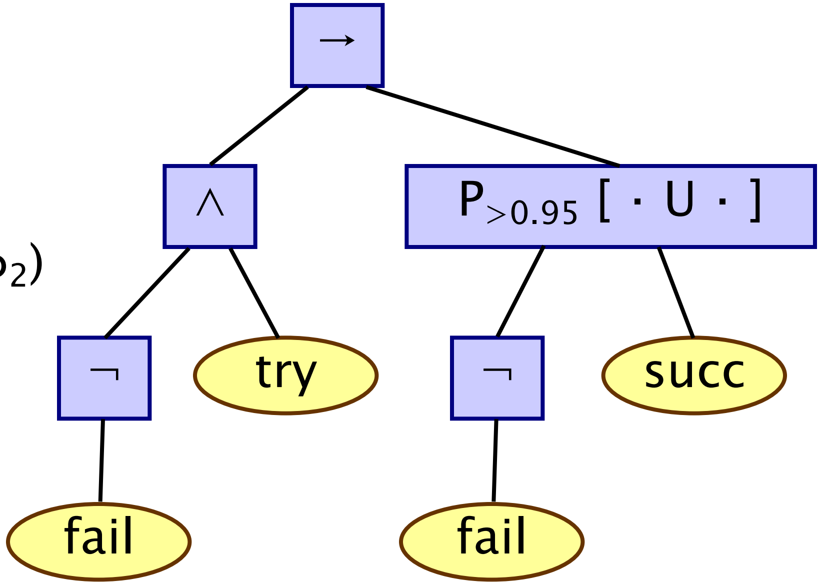
- What does it mean for a DTMC D to satisfy a formula ϕ?
  - often, just want to know if $s_{init} \vDash \phi$, i.e. if $s_{init} \in Sat(\phi)$
  - sometimes, want to check that $s \vDash \phi \ \forall \ s \in S$, i.e. $Sat(\phi) = S$

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



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

Probability computation

- Three temporal operators to consider:

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

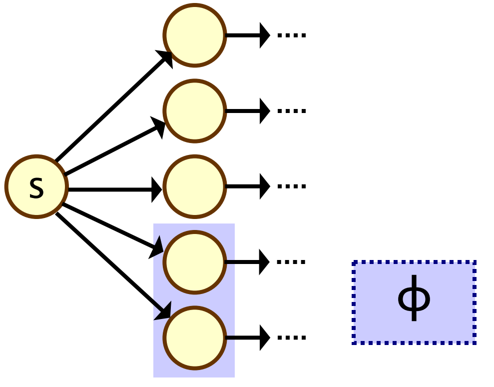
- Bounded until: $P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]$
  – adaptation of bounded reachability for DTMCs

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

PCTL next for DTMCs

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

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

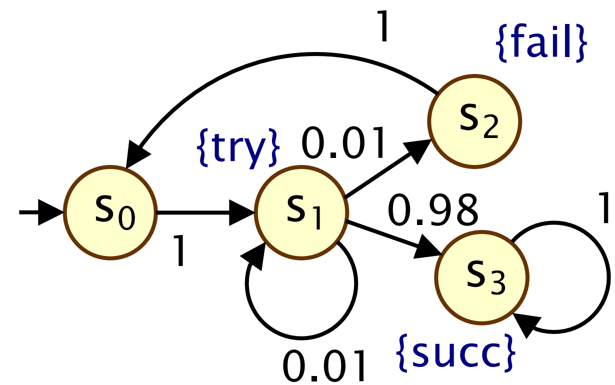


- Compute vector $\underline{Prob}(X\ \phi)$ of probabilities for all states s (useful for Sat set)
  - $\underline{Prob}(X\ \phi) = \mathbf{P} \cdot \underline{\phi}$
  - where $\underline{\phi}$ is a 0–1 vector over S with $\underline{\phi}(s) = 1$ iff $s \vDash \phi$
  - computation requires a single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

PCTL next – Example

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

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

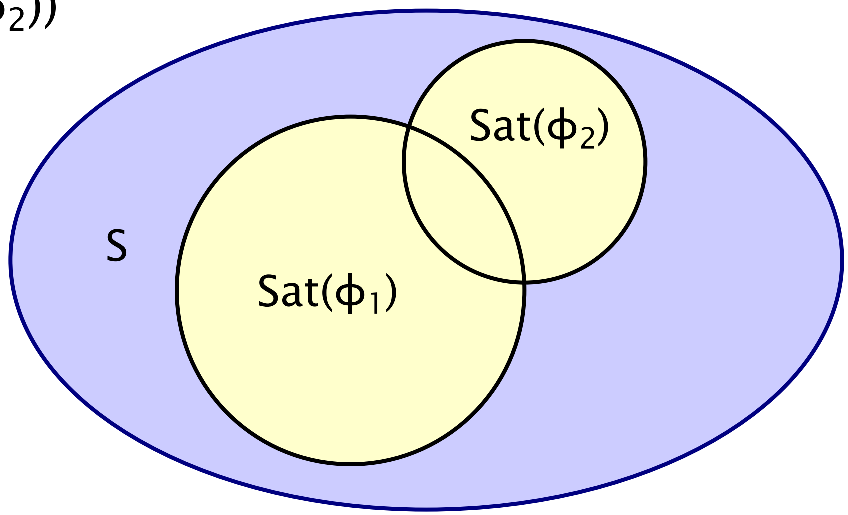


- Results:
  - <u>Prob</u>(X (¬try ∨ succ)) = [0, 0.99, 1, 1]
  - Sat($P_{\geq 0.9}$ [X (¬try ∨ succ)]) = $\{s_1, s_2, s_3\}$

PCTL bounded until for DTMCs

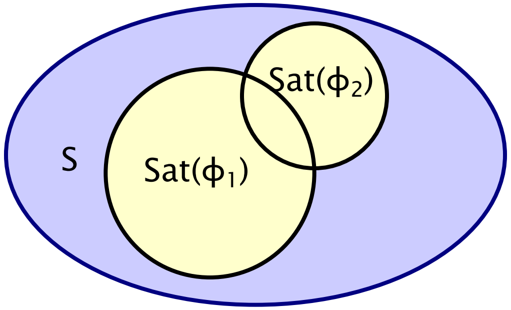
- Computation of probabilities for PCTL $U^{\leq k}$ operator
  - $Sat(P_{\sim p}[\ \phi_1\ U^{\leq k}\ \phi_2\ ]) = \{\ s \in S \mid Prob(s,\ \phi_1\ U^{\leq k}\ \phi_2) \sim p\ \}$
  - need to compute $Prob(s,\ \phi_1\ U^{\leq k}\ \phi_2)$ for all $s \in S$

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



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:

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

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 $s \in S$
  - (important in order to find Sat set of formula)

- Iteratively define in terms of matrices and vectors
  - define matrix **P'** as follows:
    - if $s \in S^?$ $\mathbf{P'}(s,s') = \mathbf{P}(s,s')$;
    - if $s \in S^{yes}$, $\mathbf{P'}(s,s') = 1$ if $s=s'$, otherwise $\mathbf{P'}(s,s') = 0$
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq 0}\ \phi_2) = \underline{\phi_2}$
  - $\underline{\text{Prob}}(\phi_1\ U^{\leq k}\ \phi_2) = \mathbf{P'} \cdot \underline{\text{Prob}}(\phi_1\ U^{\leq k-1}\ \phi_2)$
  - 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 can be inefficient, as $(\mathbf{P'})^k$ 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**
  - $\underline{\text{Prob}}(\text{true } U^{\leq 0} \text{ succ}) = \underline{\text{succ}} = [0, 0, 0, 1]$

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

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

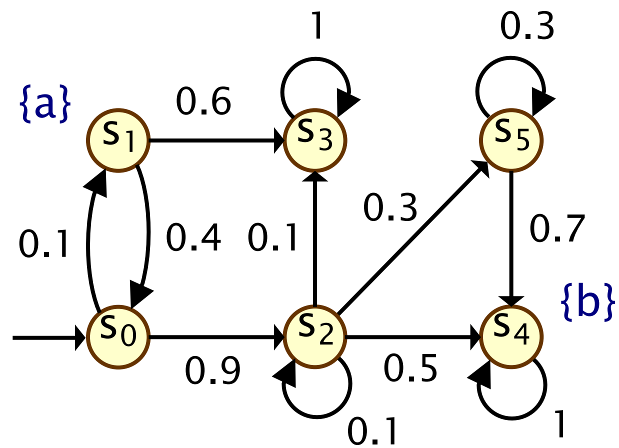
  - Sat($P_{>0.98}$ [$F^{\leq 2}$ succ]) = $\{s_1, s_3\}$

PCTL until for DTMCs

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

- Running example:

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



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)

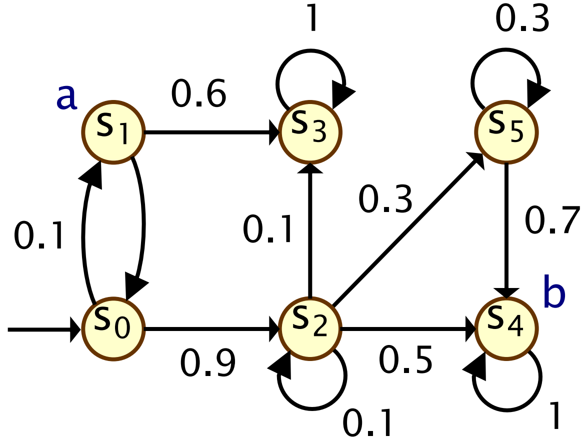
- Important for several reasons
  - Prob0 ensures unique solution to system of linear equations
  - both reduce the set of states for which probabilities must be computed numerically
  - give exact results for the states in S^{yes} and S^{no} (no round-off)
  - (of course, for model checking of qualitative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

Precomputation – Prob0

- Prob0 algorithm to compute $S^{no} = Sat(P_{\leq 0} [\ \phi_1 \text{ U } \phi_2\ ])$:
  - first compute $Sat(P_{>0} [\ \phi_1 \text{ U } \phi_2\ ]) \equiv Sat(E[\ \phi_1 \text{ U } \phi_2\ ])$
  - 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:

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

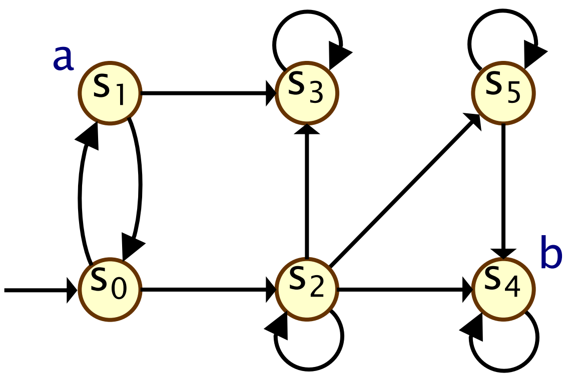


Precomputation – Prob0

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

Example:

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

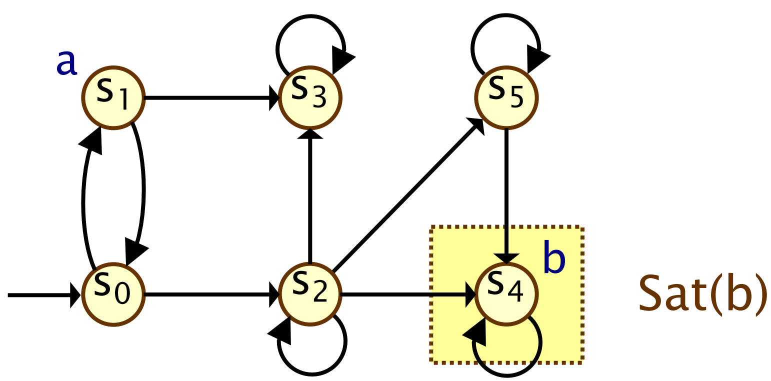


Precomputation – Prob0

- Prob0 algorithm to compute $S^{no} = Sat(P_{\leq 0}\,[\,\phi_1 \text{ U } \phi_2\,])$:
  - first compute $Sat(P_{>0}\,[\,\phi_1 \text{ U } \phi_2\,]) \equiv Sat(E[\,\phi_1 \text{ U } \phi_2\,])$
  - 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:

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

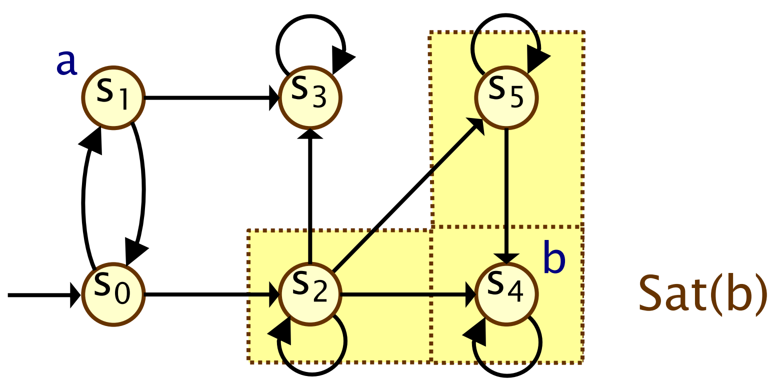


Precomputation – Prob0

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

Example:

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

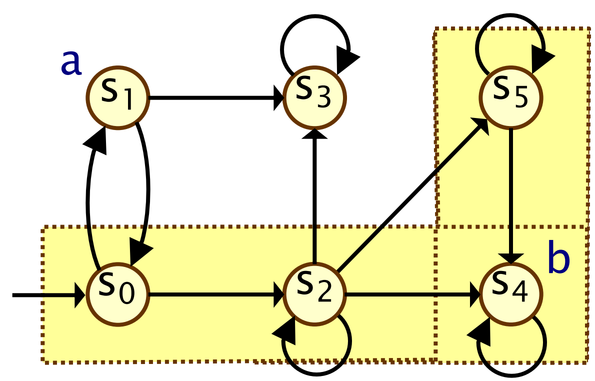


Precomputation – Prob0

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

Example:

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



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

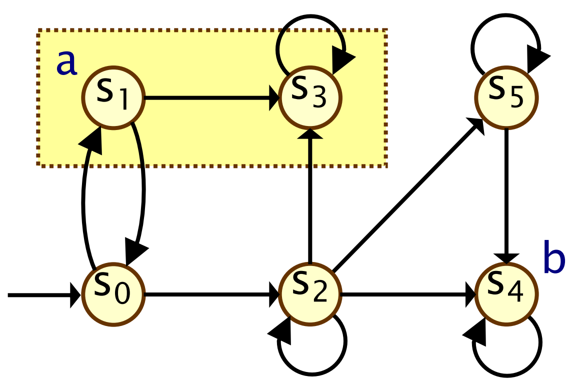
Precomputation – Prob0

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

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

Example:

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



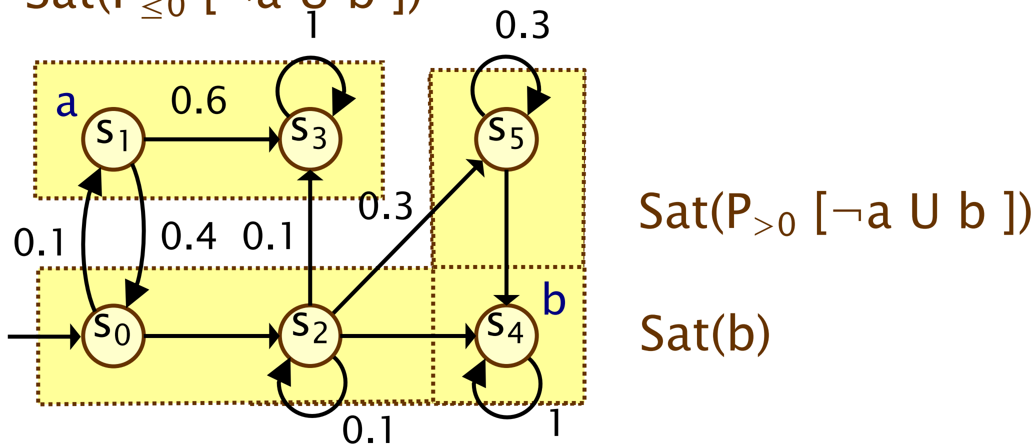
Precomputation – Prob0

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

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

Example:

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



Prob0 algorithm

PROB0($Sat(\Phi)$, $Sat(\Psi)$)

1. $R := Sat(\Psi)$
2. $done :=$ **false**
3. **while** $(done = \textbf{false})$
4. $\quad R' := R \cup \{ s \in Sat(\Phi) \mid \exists s' \in R \,.\, \mathbf{P}(s,s') > 0 \}$
5. $\quad$ **if** $(R' = R)$ **then** $done :=$ **true**
6. $\quad R := R'$
7. **endwhile**
8. **return** $S \backslash R$

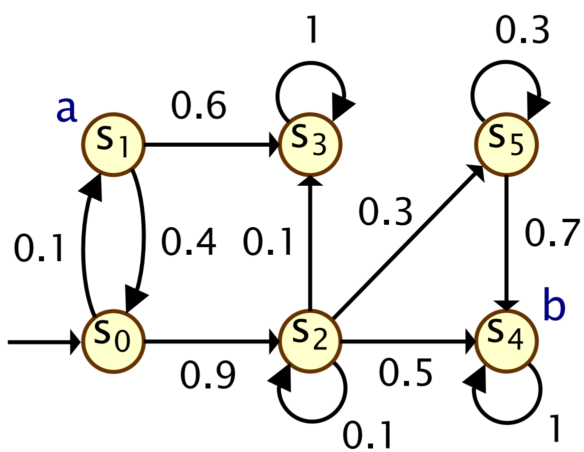
- Note: can be formulated as a least fixed point computation
  - also well suited to symbolic computations, e.g., with binary decision diagrams

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:

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

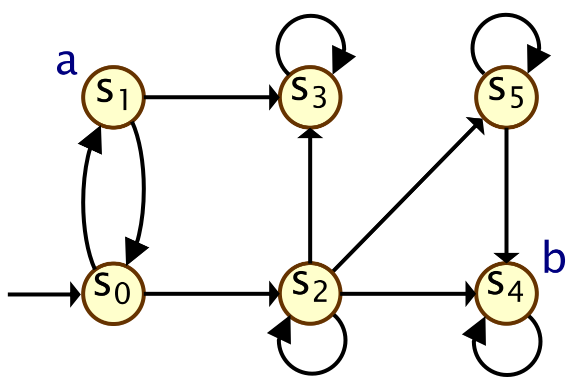


Precomputation – Prob1

- 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:

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



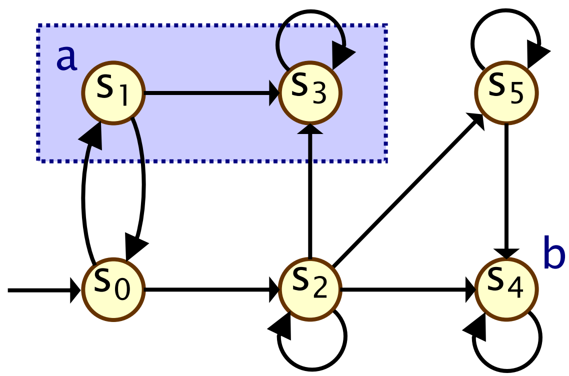
Precomputation – Prob1

- 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

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

Example:

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

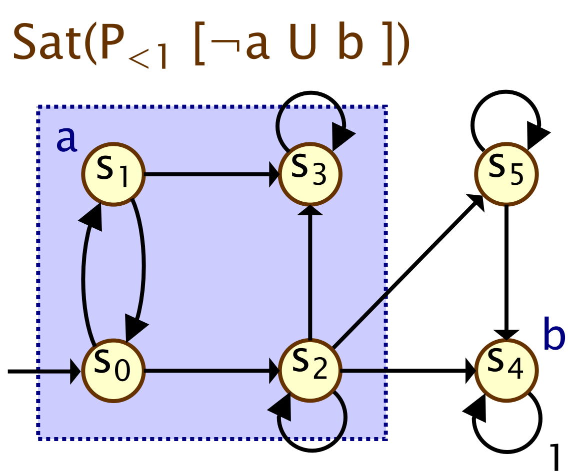


Precomputation – Prob1

- Prob1 algorithm to compute $S^{yes} = Sat(P_{\geq 1}$ [ϕ_1 U ϕ_2]) :
  - first compute $Sat(P_{<1}$ [ϕ_1 U ϕ_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:

$P_{>0.8}$ [¬a U b]

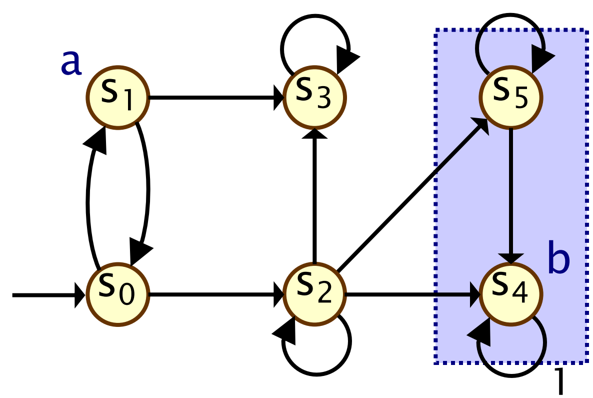


Precomputation – Prob1

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

Example:

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



$S^{yes} =$

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

Precomputation – Prob1

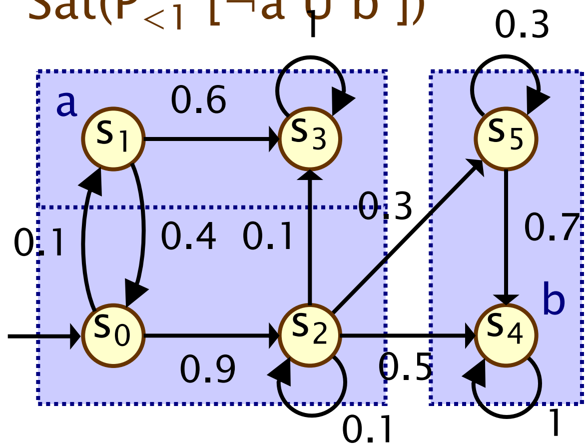
- Prob1 algorithm to compute $S^{yes} = Sat(P_{\geq 1} [\ \phi_1\ U\ \phi_2\ ])$:
  - first compute $Sat(P_{<1} [\ \phi_1\ U\ \phi_2\ ])$, reusing S^{no}
  - this is equivalent to the set of states which have a non-zero probability of reaching S^{no}, passing only through ϕ_1-states
  - again, this is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

Example:

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

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

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



$S^{yes} =$

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

Prob1 algorithm

$\textsc{Prob1}(Sat(\Phi), Sat(\Psi), Sat(\mathtt{P}_{\leqslant 0}[\Phi\ \mathtt{U}\ \Psi]))$

$R := Sat(\mathtt{P}_{\leqslant 0}[\Phi\ \mathtt{U}\ \Psi])$
$done :=$ **false**
while $(done =$ **false**$)$
$\qquad R' := R \cup \{s \in (Sat(\Phi) \backslash Sat(\Psi)) \mid \exists s' \in R \,.\, \mathbf{P}(s,s') > 0\}$
$\qquad$ **if** $(R' = R)$ **then** $done :=$ **true**
$\qquad R := R'$
endwhile
return $S \backslash R$

PCTL until – linear equations

- Probabilities $\mathrm{Prob}(s, \phi_1\ U\ \phi_2)$ can now be obtained as the unique solution of the following set of linear equations
  - essentially the same as for probabilistic reachability

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

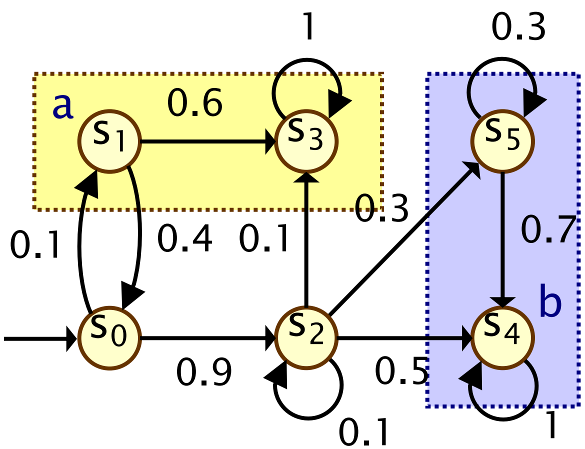
- Can also be reduced to a system in $|S^?|$ unknowns instead of $|S|$ where $S^? = S \setminus (S^{yes} \cup S^{no})$

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])

$x_1 = x_3 = 0$

$x_4 = x_5 = 1$

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

$x_0 = 0.1x_1+0.9x_2 = 0.8$

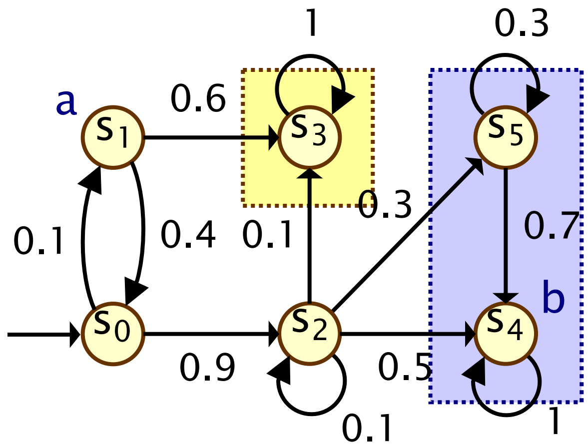
<u>Prob</u>(¬a U b) = <u>x</u> = [0.8, 0, 8/9, 0, 1, 1]

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)
  = Prob(s_i, ¬(F 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 }

System of linear equations

- Solution of large (likely sparse) systems of linear equations
  - 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. matrix 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 }

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

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 k^{th} 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

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

  - maximum relative difference

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

Jacobi method

- Based on fact that:

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

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

- can be rearranged as:

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

- yielding this update scheme:

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

Gauss-Seidel

- The update scheme for Jacobi:

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

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

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

- (uses just one vector, as opposed to Jacobi's)

Over-relaxation

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

- Example: Jacobi + over-relaxation

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

- where $\omega \in (0,2)$ is a parameter to the algorithm

Comparison

- Gauss-Seidel typically outperforms Jacobi
  - i.e. faster convergence
  - Also, it requires only storing single solution vector
- Both Gauss-Seidel and Jacobi usually outperform the matrix 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 in practice
  - 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 system of linear equations, 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 $Sat(\phi)$
  - recursive: bottom-up traversal of parse tree of ϕ
- Atomic propositions and logical connectives: trivial
- Key part: computing probabilities for $P_{\sim p}$ […] formulae
  - $X\ \Phi$: one matrix-vector multiplications
  - $\Phi_1\ U^{\leq k}\ \Phi_2$: k matrix-vector multiplications
  - $\Phi_1\ U\ \Phi_2$: graph-based precomputation algorithms + solution of linear equation system in at most $|S|$ variables
- Iterative methods to solve large systems of linear equations