Modeling and Verification of Probabilistic Systems

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/teaching/ws-1516/movep15/

November 10, 2015

Overview

Summary of previous lecture

Reachability probabilities

Can be obtained as a unique solution of a linear equation system.

Summary of previous lecture

Reachability probabilities

Can be obtained as a unique solution of a linear equation system.

Reachability probabilities are pivotal

The probability of satisfying an ω-regular property P in a Markov chain $\mathcal{D}$ $=$ reachability probability of accepting BSCCs in the product of $\mathcal{D}$ with a DRA for P.

Aim of this lecture

Introduce probabilistic CTL. Provide a polynomial-time model-checking algorithm for verifying a finite Markov chain against a PCTL formula.

Aim of this lecture

Introduce probabilistic CTL. Provide a polynomial-time model-checking algorithm for verifying a finite Markov chain against a PCTL formula.

Set up of this lecture

1. Syntax and formal semantics of probabilistic CTL.

Aim of this lecture

Introduce probabilistic CTL. Provide a polynomial-time model-checking algorithm for verifying a finite Markov chain against a PCTL formula.

Set up of this lecture

1. Syntax and formal semantics of probabilistic CTL.
2. Model checking algorithm for probabilistic CTL on Markov chains.

Aim of this lecture

Introduce probabilistic CTL. Provide a polynomial-time model-checking algorithm for verifying a finite Markov chain against a PCTL formula.

Set up of this lecture

1. Syntax and formal semantics of probabilistic CTL.
2. Model checking algorithm for probabilistic CTL on Markov chains.
3. Time complexity analysis.

Overview

Probabilistic Computation Tree Logic

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It is a branching-time temporal logic (based on CTL).

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It is a branching-time temporal logic (based on CTL).
- Formula interpretation is Boolean, i.e., a formula is satisfied or not.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It is a branching-time temporal logic (based on CTL).
- Formula interpretation is Boolean, i.e., a formula is satisfied or not.
- The main operator is $\mathbb{P}_J(\varphi)$
  - where φ constrains the set of paths and J is a threshold on the probability.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It is a branching-time temporal logic (based on CTL).
- Formula interpretation is Boolean, i.e., a formula is satisfied or not.
- The main operator is $\mathbb{P}_J(\varphi)$
  - where φ constrains the set of paths and J is a threshold on the probability.
  - it is the probabilistic counterpart of $\exists$ and $\forall$ path-quantifiers in CTL.

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\text{init}}, AP, L)$ with:

- S is a countable nonempty set of states

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\text{init}}, AP, L)$ with:

- S is a countable nonempty set of states
- $\mathbf{P} : S \times S \to [0, 1]$, transition probability function s.t. $\sum_{s'} \mathbf{P}(s, s') = 1$

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\text{init}}, AP, L)$ with:

- S is a countable nonempty set of states
- $\mathbf{P} : S \times S \rightarrow [0, 1]$, transition probability function s.t. $\sum_{s'} \mathbf{P}(s, s') = 1$
- $\iota_{\text{init}} : S \rightarrow [0, 1]$, the initial distribution with $\sum\limits_{s \in S} \iota_{\text{init}}(s) = 1$

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\text{init}}, AP, L)$ with:

- S is a countable nonempty set of states
- $\mathbf{P} : S \times S \to [0, 1]$, transition probability function s.t. $\sum_{s'} \mathbf{P}(s, s') = 1$
- $\iota_{\text{init}} : S \to [0, 1]$, the initial distribution with $\sum_{s \in S} \iota_{\text{init}}(s) = 1$
- AP is a set of atomic propositions.

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\mathrm{init}}, AP, L)$ with:

- S is a countable nonempty set of states
- $\mathbf{P} : S \times S \to [0, 1]$, transition probability function s.t. $\sum_{s'} \mathbf{P}(s, s') = 1$
- $\iota_{\mathrm{init}} : S \to [0, 1]$, the initial distribution with $\sum\limits_{s \in S} \iota_{\mathrm{init}}(s) = 1$
- AP is a set of atomic propositions.
- $L : S \to 2^{AP}$, the labeling function, assigning to state s, the set $L(s)$ of atomic propositions that are valid in s.

DTMCs — A transition system perspective

Discrete-time Markov chain

A DTMC $\mathcal{D}$ is a tuple $(S, \mathbf{P}, \iota_{\mathrm{init}}, AP, L)$ with:

- S is a countable nonempty set of states
- $\mathbf{P} : S \times S \to [0, 1]$, transition probability function s.t. $\sum_{s'} \mathbf{P}(s, s') = 1$
- $\iota_{\mathrm{init}} : S \to [0, 1]$, the initial distribution with $\sum_{s \in S} \iota_{\mathrm{init}}(s) = 1$
- AP is a set of atomic propositions.
- $L : S \to 2^{AP}$, the labeling function, assigning to state s, the set $L(s)$ of atomic propositions that are valid in s.

Initial states

- $\iota_{\mathrm{init}}(s)$ is the probability that DTMC $\mathcal{D}$ starts in state s
- the set $\{ s \in S \mid \iota_{\mathrm{init}}(s) > 0 \}$ are the possible initial states.

PCTL syntax [Hansson & Jonsson, 1994]

Probabilistic Computation Tree Logic: Syntax

PCTL consists of state- and path-formulas.

PCTL syntax [Hansson & Jonsson, 1994]

Probabilistic Computation Tree Logic: Syntax

PCTL consists of state- and path-formulas.

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0, 1]$, $J \neq \varnothing$ is a non-empty interval.

PCTL syntax

[Hansson & Jonsson, 1994]

Probabilistic Computation Tree Logic: Syntax

PCTL consists of state- and path-formulas.

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \;::=\; \text{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$, $J \neq \varnothing$ is a non-empty interval.

- PCTL *path formulae* are formed according to the following grammar:

$$\varphi \;::=\; \bigcirc\Phi \;\Big|\; \Phi_1 \,\mathsf{U}\, \Phi_2 \;\Big|\; \Phi_1 \,\mathsf{U}^{\leqslant n}\, \Phi_2$$

  where Φ, Φ_1, and Φ_2 are state formulae and $n \in \mathbb{N}$.

PCTL syntax [Hansson & Jonsson, 1994]

Probabilistic Computation Tree Logic: Syntax

PCTL consists of state- and path-formulas.

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$, $J \neq \varnothing$ is a non-empty interval.

- PCTL *path formulae* are formed according to the following grammar:

$$\varphi \ ::= \ \bigcirc\Phi \ \Big| \ \Phi_1 \, \mathsf{U} \, \Phi_2 \ \Big| \ \Phi_1 \, \mathsf{U}^{\leqslant n} \, \Phi_2$$

  where Φ, Φ_1, and Φ_2 are state formulae and $n \in \mathbb{N}$.

  Abbreviate $\mathbb{P}_{[0,0.5]}(\varphi)$ by $\mathbb{P}_{\leqslant 0.5}(\varphi)$ and $\mathbb{P}_{]0,1]}(\varphi)$ by $\mathbb{P}_{>0}(\varphi)$.

Probabilistic Computation Tree Logic

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$, $J \neq \varnothing$ is a non-empty interval.

- PCTL *path formulae* are formed according to the following grammar:

$$\varphi \ ::= \ \bigcirc \Phi \ \Big| \ \Phi_1 \, \mathsf{U} \, \Phi_2 \ \Big| \ \Phi_1 \, \mathsf{U}^{\leqslant n} \, \Phi_2 \quad \text{where } n \in \mathbb{N}.$$

Intuitive semantics

Probabilistic Computation Tree Logic

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$, $J \neq \varnothing$ is a non-empty interval.

- PCTL *path formulae* are formed according to the following grammar:

$$\varphi \ ::= \ \bigcirc\Phi \ \Big| \ \Phi_1 \, \mathsf{U} \, \Phi_2 \ \Big| \ \Phi_1 \, \mathsf{U}^{\leqslant n} \, \Phi_2 \quad \text{where } n \in \mathbb{N}.$$

Intuitive semantics

- $s_0 s_1 s_2 \ldots \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi$ if Φ holds until Ψ holds within n steps.

Probabilistic Computation Tree Logic

- PCTL *state formulas* over the set AP obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathbb{P}_J(\varphi)$$

  where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$, $J \neq \varnothing$ is a non-empty interval.

- PCTL *path formulae* are formed according to the following grammar:

$$\varphi \ ::= \ \bigcirc\Phi \ \Big| \ \Phi_1 \, \mathrm{U} \, \Phi_2 \ \Big| \ \Phi_1 \, \mathrm{U}^{\leqslant n} \, \Phi_2 \quad \text{where } n \in \mathbb{N}.$$

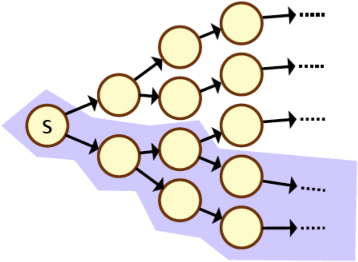
Intuitive semantics

- $s_0 s_1 s_2 \ldots \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi$ if Φ holds until Ψ holds within n steps.
- $s \models \mathbb{P}_J(\varphi)$ if probability that paths starting in s fulfill φ lies in J.

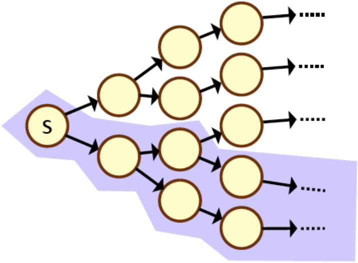
Overview

Semantics of $\mathbb{P}$-operator

Semantics of $\mathbb{P}$-operator

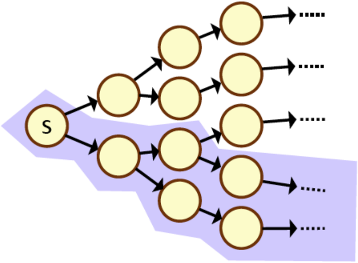


Semantics of $\mathbb{P}$-operator



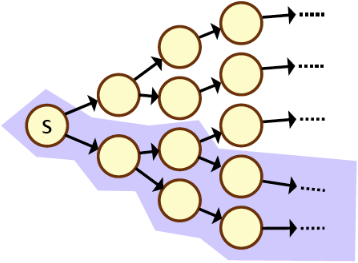
- $s \models \mathbb{P}_J(\varphi)$ if:
  - the probability of all paths starting in s fulfilling φ lies in J.

Semantics of $\mathbb{P}$-operator



- $s \models \mathbb{P}_J(\varphi)$ if:
  - the probability of all paths starting in s fulfilling φ lies in J.
- Example: $s \models \mathbb{P}_{>\frac{1}{2}}(\Diamond a)$ if
  - the probability to reach an a-labeled state from s exceeds $\frac{1}{2}$.

Semantics of $\mathbb{P}$-operator



- $s \models \mathbb{P}_J(\varphi)$ if:
  - the probability of all paths starting in s fulfilling φ lies in J.
- Example: $s \models \mathbb{P}_{>\frac{1}{2}}(\Diamond a)$ if
  - the probability to reach an a-labeled state from s exceeds $\frac{1}{2}$.
- Formally:
  - $s \models \mathbb{P}_J(\varphi)$ if and only if $Pr_s\{\, \pi \in Paths(s) \mid \pi \models \varphi \,\} \in J$.

Derived operators

$$\Diamond \Phi = \text{true} \, \mathsf{U} \, \Phi$$

Derived operators

$$\Diamond \Phi \;=\; \text{true}\,\mathsf{U}\,\Phi$$

$$\Diamond^{\leqslant n} \Phi \;=\; \text{true}\,\mathsf{U}^{\leqslant n} \Phi$$

Derived operators

$$\Diamond \Phi \ = \ \text{true} \, \mathsf{U} \, \Phi$$

$$\Diamond^{\leqslant n} \Phi \ = \ \text{true} \, \mathsf{U}^{\leqslant n} \Phi$$

$$\mathbb{P}_{\leqslant p}(\Box \Phi) \ = \ \mathbb{P}_{>1-p}(\Diamond \neg \Phi)$$

Derived operators

$$\Diamond \Phi \;=\; \mathrm{true}\, \mathsf{U}\, \Phi$$

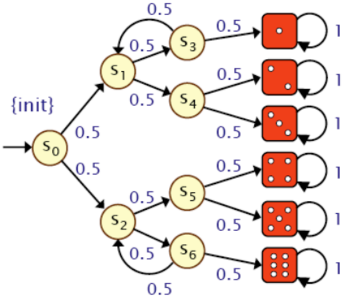
$$\Diamond^{\leqslant n} \Phi \;=\; \mathrm{true}\, \mathsf{U}^{\leqslant n} \Phi$$

$$\mathbb{P}_{\leqslant p}(\Box \Phi) \;=\; \mathbb{P}_{>1-p}(\Diamond \neg \Phi)$$

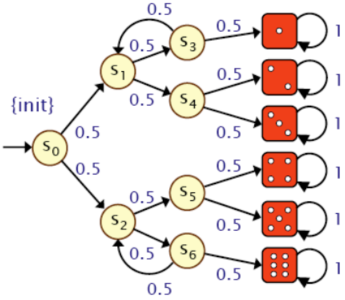
$$\mathbb{P}_{(p,q)}(\Box^{\leqslant n} \Phi) \;=\; \mathbb{P}_{[1-q,1-p]}(\Diamond^{\leqslant n} \neg \Phi)$$

Correctness of Knuth's die

Correctness of Knuth's die



Correctness of Knuth's die



Correctness of Knuth's die

$$\mathbb{P}_{=\frac{1}{6}}(\Diamond 1) \ \wedge\ \mathbb{P}_{=\frac{1}{6}}(\Diamond 2) \ \wedge\ \mathbb{P}_{=\frac{1}{6}}(\Diamond 3) \ \wedge\ \mathbb{P}_{=\frac{1}{6}}(\Diamond 4) \ \wedge\ \mathbb{P}_{=\frac{1}{6}}(\Diamond 5) \ \wedge\ \mathbb{P}_{=\frac{1}{6}}(\Diamond 6)$$

Example properties

- Transient probabilities to be in *goal* state at the fourth epoch:

$$\mathbb{P}_{\geqslant 0.92}\left(\Diamond^{=4}\ goal\right)$$

Example properties

- Transient probabilities to be in *goal* state at the fourth epoch:

$$\mathbb{P}_{\geqslant 0.92} \left(\Diamond^{=4} \; goal \right)$$

- With probability $\geqslant 0.92$, a goal state is reached legally:

$$\mathbb{P}_{\geqslant 0.92} \left(\neg \, illegal \;\; \mathrm{U} \;\; goal \right)$$

Example properties

- Transient probabilities to be in *goal* state at the fourth epoch:

$$\mathbb{P}_{\geqslant 0.92}\left(\lozenge^{=4}\ goal\right)$$

- With probability $\geqslant 0.92$, a goal state is reached legally:

$$\mathbb{P}_{\geqslant 0.92}\left(\neg\, illegal\ \mathsf{U}\ goal\right)$$

- ... in maximally 137 steps: $\mathbb{P}_{\geqslant 0.92}\left(\neg\, illegal\ \mathsf{U}^{\leqslant 137}\ goal\right)$

Example properties

- Transient probabilities to be in *goal* state at the fourth epoch:

$$\mathbb{P}_{\geqslant 0.92}\left(\Diamond^{=4}\ goal\right)$$

- With probability $\geqslant 0.92$, a goal state is reached legally:

$$\mathbb{P}_{\geqslant 0.92}\left(\neg\, illegal\ \mathsf{U}\ goal\right)$$

- . . . in maximally 137 steps: $\mathbb{P}_{\geqslant 0.92}\left(\neg\, illegal\ \mathsf{U}^{\leqslant 137}\ goal\right)$
- . . . once there, remain there almost surely for the next 31 steps:

$$\mathbb{P}_{\geqslant 0.92}\left(\neg\, illegal\ \mathsf{U}^{\leqslant 137}\ \mathbb{P}_{=1}(\Box^{[0,31]}\ goal)\right)$$

PCTL semantics (1)

PCTL semantics (1)

Notation

$\mathcal{D}, s \models \Phi$ if and only if state-formula Φ holds in state s of (possibly infinite) DTMC $\mathcal{D}$. As $\mathcal{D}$ is known from the context we simply write $s \models \Phi$.

PCTL semantics (1)

Notation

$\mathcal{D}, s \models \Phi$ if and only if state-formula Φ holds in state s of (possibly infinite) DTMC $\mathcal{D}$. As $\mathcal{D}$ is known from the context we simply write $s \models \Phi$.

Satisfaction relation for state formulas

The satisfaction relation $\models$ is defined for PCTL state formulas by:

$$\begin{array}{lll} s \models a & \text{iff} & a \in L(s) \\ s \models \neg \Phi & \text{iff} & \text{not } (s \models \Phi) \\ s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \end{array}$$

PCTL semantics (1)

Notation

$\mathcal{D}, s \models \Phi$ if and only if state-formula Φ holds in state s of (possibly infinite) DTMC $\mathcal{D}$. As $\mathcal{D}$ is known from the context we simply write $s \models \Phi$.

Satisfaction relation for state formulas

The satisfaction relation $\models$ is defined for PCTL state formulas by:

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\,\Phi & \text{iff} & \text{not } (s \models \Phi) \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \mathbb{P}_J(\varphi) & \text{iff} & Pr(s \models \varphi) \in J
\end{array}
$$

where $Pr(s \models \varphi) = Pr_s\{\,\pi \in Paths(s) \mid \pi \models \varphi\,\}$

PCTL semantics (2)

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\, s_1\, s_2 \ldots$ be an infinite path in (possibly infinite) DTMC $\mathcal{D}$.
Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

The satisfaction relation $\models$ is defined for state formulas by:

$$\pi \models \bigcirc \Phi \qquad \text{iff} \quad s_1 \models \Phi$$

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\, s_1\, s_2 \ldots$ be an infinite path in (possibly infinite) DTMC $\mathcal{D}$.
Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

The satisfaction relation $\models$ is defined for state formulas by:

$$\begin{array}{lll}
\pi \models \bigcirc\Phi & \text{iff} & s_1 \models \Phi \\
\pi \models \Phi \,\mathsf{U}\, \Psi & \text{iff} & \exists k \geqslant 0.(\, \pi[k] \models \Psi \text{ and } \forall 0 \leqslant i < k.\, \pi[i] \models \Phi\,)
\end{array}$$

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\, s_1\, s_2 \ldots$ be an infinite path in (possibly infinite) DTMC $\mathcal{D}$.
Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

The satisfaction relation $\models$ is defined for state formulas by:

$$
\begin{array}{lll}
\pi \models \bigcirc \Phi & \text{iff} & s_1 \models \Phi \\
\pi \models \Phi \, \mathsf{U} \, \Psi & \text{iff} & \exists k \geqslant 0. (\, \pi[k] \models \Psi \text{ and } \forall 0 \leqslant i < k.\, \pi[i] \models \Phi \,) \\
\pi \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi & \text{iff} & \exists k \geqslant 0. (\, k \leqslant n \text{ and } \pi[k] \models \Psi \text{ and} \\
 & & \qquad \forall 0 \leqslant i < k.\, \pi[i] \models \Phi \,)
\end{array}
$$

Examples

Measurability

Measurability

PCTL measurability

For any PCTL path formula φ and state s of DTMC $\mathcal{D}$,
the set $\{ \pi \in \mathit{Paths}(s) \mid \pi \models \varphi \}$ is measurable.

Measurability

PCTL measurability

For any PCTL path formula φ and state s of DTMC $\mathcal{D}$,
the set $\{\, \pi \in \mathit{Paths}(s) \mid \pi \models \varphi \,\}$ is measurable.

Proof (sketch):

Three cases:

1. $\bigcirc \Phi$:
   - cylinder sets constructed from paths of length one.
2. $\Phi \, \mathsf{U}^{\leqslant n} \, \Psi$:
   - (finite number of) cylinder sets from paths of length at most n.
3. $\Phi \, \mathsf{U} \, \Psi$:
   - countable union of paths satisfying $\Phi \, \mathsf{U}^{\leqslant n} \, \Psi$ for all $n \geqslant 0$.

Overview

PCTL model checking

PCTL model checking

PCTL model checking problem

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\mathrm{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.
2. This is done recursively by a bottom-up traversal of Φ's parse tree.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.
2. This is done recursively by a bottom-up traversal of Φ's parse tree.
   - The nodes of the parse tree represent the subformulae of Φ.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.
2. This is done recursively by a bottom-up traversal of Φ's parse tree.
   - The nodes of the parse tree represent the subformulae of Φ.
   - For each node, i.e., for each subformula Ψ of Φ, determine $Sat(\Psi)$.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.
2. This is done recursively by a bottom-up traversal of Φ's parse tree.
   - The nodes of the parse tree represent the subformulae of Φ.
   - For each node, i.e., for each subformula Ψ of Φ, determine $Sat(\Psi)$.
   - Determine $Sat(\Psi)$ as function of the satisfaction sets of its children:

     e.g., $Sat(\Psi_1 \wedge \Psi_2) = Sat(\Psi_1) \cap Sat(\Psi_2)$ and $Sat(\neg\Psi) = S \setminus Sat(\Psi)$.

PCTL model checking

PCTL model checking problem

Input: a finite DTMC $\mathcal{D} = (S, \mathbf{P}, \iota_{\text{init}}, AP, L)$, state $s \in S$, and PCTL state formula Φ

Output: yes, if $s \models \Phi$; no, otherwise.

Basic algorithm

In order to check whether $s \models \Phi$ do:

1. Compute the satisfaction set $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$.
2. This is done recursively by a bottom-up traversal of Φ's parse tree.
   - The nodes of the parse tree represent the subformulae of Φ.
   - For each node, i.e., for each subformula Ψ of Φ, determine $Sat(\Psi)$.
   - Determine $Sat(\Psi)$ as function of the satisfaction sets of its children:

     e.g., $Sat(\Psi_1 \wedge \Psi_2) = Sat(\Psi_1) \cap Sat(\Psi_2)$ and $Sat(\neg\Psi) = S \setminus Sat(\Psi)$.
3. Check whether state s belongs to $Sat(\Phi)$.

Example

Core model checking algorithm

Core model checking algorithm

Propositional formulas

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as follows:

$$
\begin{aligned}
Sat(\mathrm{true}) &= S \\
Sat(a) &= \{\, s \in S \mid a \in L(s) \,\}, \text{ for any } a \in AP \\
Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi).
\end{aligned}
$$

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as follows:

$$\begin{aligned} Sat(\text{true}) &= S \\ Sat(a) &= \{ s \in S \mid a \in L(s) \}, \text{ for any } a \in AP \\ Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\ Sat(\neg\Phi) &= S \setminus Sat(\Phi). \end{aligned}$$

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_J(\varphi))$, the probability $Pr(s \models \varphi)$ for the event specified by φ needs to be established.

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as follows:

$$\begin{aligned} Sat(\text{true}) &= S \\ Sat(a) &= \{ s \in S \mid a \in L(s) \}, \text{ for any } a \in AP \\ Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\ Sat(\neg \Phi) &= S \setminus Sat(\Phi). \end{aligned}$$

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_J(\varphi))$, the probability $Pr(s \models \varphi)$ for the event specified by φ needs to be established. Then

$$Sat(\mathbb{P}_J(\varphi)) = \{ s \in S \mid Pr(s \models \varphi) \in J \}.$$

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as follows:

$$
\begin{array}{rcl}
Sat(\text{true}) & = & S \\
Sat(a) & = & \{\, s \in S \mid a \in L(s) \,\}, \text{ for any } a \in AP \\
Sat(\Phi \wedge \Psi) & = & Sat(\Phi) \cap Sat(\Psi) \\
Sat(\neg\Phi) & = & S \setminus Sat(\Phi).
\end{array}
$$

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_J(\varphi))$, the probability $Pr(s \models \varphi)$ for the event specified by φ needs to be established. Then

$$Sat(\mathbb{P}_J(\varphi)) = \{s \in S \mid Pr(s \models \varphi) \in J\}.$$

Let us consider the computation of $Pr(s \models \varphi)$ for all possible φ.

The next-step operator

The next-step operator

Recall that: $s \models \mathbb{P}_J(\bigcirc \Phi)$ if and only if $Pr(s \models \bigcirc \Phi) \in J$.

The next-step operator

Recall that: $s \models \mathbb{P}_J(\bigcirc \Phi)$ if and only if $Pr(s \models \bigcirc \Phi) \in J$.

Lemma

$Pr(s \models \bigcirc \Phi) = \sum_{s' \in \xrightarrow{s} at(\Phi)} \mathbf{P}(s, s')$.

The next-step operator

Recall that: $s \models \mathbb{P}_J(\bigcirc \Phi)$ if and only if $Pr(s \models \bigcirc \Phi) \in J$.

Lemma

$Pr(s \models \bigcirc \Phi) = \sum_{s' \in \xrightarrow{s} at(\Phi)} \mathbf{P}(s, s')$.

Algorithm

Considering the above equation for all states simultaneously yields:

The next-step operator

Recall that: $s \models \mathbb{P}_J(\bigcirc \Phi)$ if and only if $Pr(s \models \bigcirc \Phi) \in J$.

Lemma

$Pr(s \models \bigcirc \Phi) = \sum_{s' \in \xrightarrow{S} at(\Phi)} \mathbf{P}(s, s')$.

Algorithm

Considering the above equation for all states simultaneously yields:

$$\big(Pr(s \models \bigcirc \Phi)\big)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_\Phi$$

with $\mathbf{b}_\Phi$ the characteristic vector of $Sat(\Phi)$, i.e., $b_\Phi(s) = 1$ iff $s \in \xrightarrow{S} at(\Phi)$.

The next-step operator

Recall that: $s \models \mathbb{P}_J(\bigcirc \Phi)$ if and only if $Pr(s \models \bigcirc \Phi) \in J$.

Lemma

$Pr(s \models \bigcirc \Phi) = \sum_{s' \in \xrightarrow{S} at(\Phi)} \mathbf{P}(s, s')$.

Algorithm

Considering the above equation for all states simultaneously yields:

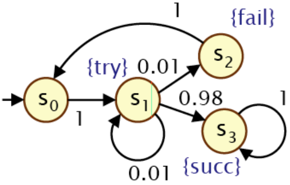
$$\big(Pr(s \models \bigcirc \Phi)\big)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_\Phi$$

with $\mathbf{b}_\Phi$ the characteristic vector of $Sat(\Phi)$, i.e., $b_\Phi(s) = 1$ iff $s \in \xrightarrow{S} at(\Phi)$.

Checking the next-step operator reduces to a single matrix-vector multiplication.

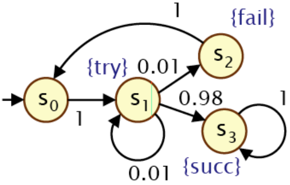
Example

Consider DTMC:



Example

Consider DTMC:

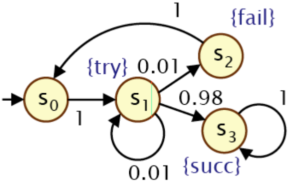


and PCTL-formula:

$$\mathbb{P}_{\geqslant 0.9}\,(\bigcirc\,(\neg try \vee\ succ))$$

Example

Consider DTMC:



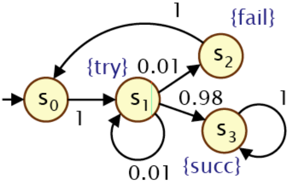
and PCTL-formula:

$$\mathbb{P}_{\geqslant 0.9}\,(\bigcirc\,(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) \;=\; (S \setminus Sat(try)) \cup Sat(succ) \;=\; \{\, s_0, s_2, s_3 \,\}$

Example

Consider DTMC:



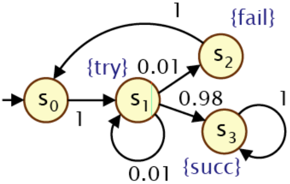
and PCTL-formula:

$$\mathbb{P}_{\geqslant 0.9}\left(\bigcirc\left(\neg try \vee succ\right)\right)$$

1. $Sat(\neg try \vee succ) \;=\; (S \setminus Sat(try)) \cup Sat(succ) \;=\; \{\, s_0, s_2, s_3 \,\}$
2. We know: $\big(Pr(s \models \bigcirc \Phi)\big)_{s \in S} \;=\; \mathbf{P} \cdot \mathbf{b}_{\Phi}$ where $\Phi = \neg try \vee succ$

Example

Consider DTMC:



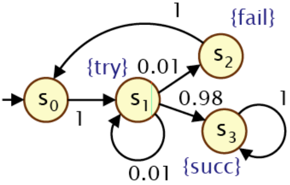
and PCTL-formula:

$$\mathbb{P}_{\geqslant 0.9}(\bigcirc(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) = (S \setminus Sat(try)) \cup Sat(succ) = \{ s_0, s_2, s_3 \}$
2. We know: $\left(Pr(s \models \bigcirc \Phi)\right)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_{\Phi}$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

Example

Consider DTMC:



and PCTL-formula:

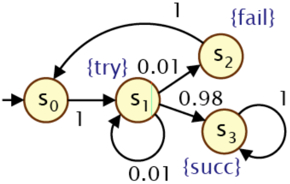
$$\mathbb{P}_{\geqslant 0.9}\left(\bigcirc\left(\neg try \vee succ\right)\right)$$

1. $Sat(\neg try \vee succ) \;=\; (S \setminus Sat(try)) \cup Sat(succ) \;=\; \{ s_0, s_2, s_3 \}$
2. We know: $\left(Pr(s \models \bigcirc \Phi)\right)_{s \in S} \;=\; \mathbf{P} \cdot \mathbf{b}_{\Phi}$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

$$\left(Pr(s \models \bigcirc \Phi)\right)_{s \in S} \;=$$

Example

Consider DTMC:



and PCTL-formula:

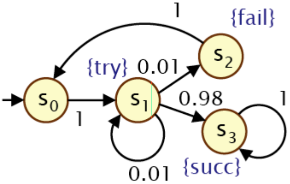
$$\mathbb{P}_{\geqslant 0.9}\,(\bigcirc\,(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) \;=\; (S \setminus Sat(try)) \cup Sat(succ) \;=\; \{ s_0, s_2, s_3 \}$
2. We know: $\big(Pr(s \models \bigcirc\, \Phi)\big)_{s \in S} \;=\; \mathbf{P} \cdot \mathbf{b}_{\Phi}$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

$$\big(Pr(s \models \bigcirc\, \Phi)\big)_{s \in S} \;=\; \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example

Consider DTMC:



and PCTL-formula:

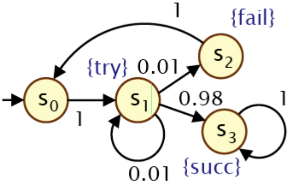
$$\mathbb{P}_{\geqslant 0.9}(\bigcirc(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) = (S \setminus Sat(try)) \cup Sat(succ) = \{ s_0, s_2, s_3 \}$
2. We know: $\big(Pr(s \models \bigcirc \Phi)\big)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_\Phi$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

$$\big(Pr(s \models \bigcirc \Phi)\big)_{s \in S} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Example

Consider DTMC:



and PCTL-formula:

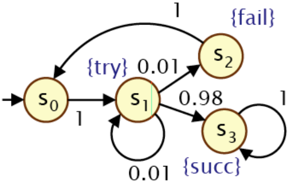
$$\mathbb{P}_{\geqslant 0.9}(\bigcirc(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) = (S \setminus Sat(try)) \cup Sat(succ) = \{ s_0, s_2, s_3 \}$
2. We know: $\left(Pr(s \models \bigcirc \Phi)\right)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_\Phi$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

$$\left(Pr(s \models \bigcirc \Phi)\right)_{s \in S} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{pmatrix}$$

Example

Consider DTMC:



and PCTL-formula:

$$\mathbb{P}_{\geqslant 0.9}\,(\bigcirc\,(\neg try \vee succ))$$

1. $Sat(\neg try \vee succ) = (S \setminus Sat(try)) \cup Sat(succ) = \{ s_0, s_2, s_3 \}$
2. We know: $\big(Pr(s \models \bigcirc\, \Phi)\big)_{s \in S} = \mathbf{P} \cdot \mathbf{b}_\Phi$ where $\Phi = \neg try \vee succ$
3. Applying that to this example yields:

$$\big(Pr(s \models \bigcirc\, \Phi)\big)_{s \in S} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{pmatrix}$$

4. Thus: $Sat(\mathbb{P}_{\geqslant 0.9}(\bigcirc\,(\neg try \vee succ)) = \{ s_1, s_2, s_3 \}$.

Bounded until (1)

Bounded until (1)

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \in J$.

Bounded until (1)

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \in J$.

Lemma

Bounded until (1)

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \in J$.

Lemma

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$.

Bounded until (1)

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \; \in \; J$.

Lemma

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \;=\; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ & \\ & \end{cases}$$

Bounded until (1)

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \in J$.

Lemma

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) = \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \sum\limits_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathsf{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi) \; = \; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \displaystyle\sum_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathrm{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi) \; = \; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \displaystyle\sum_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathrm{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Algorithm

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \;=\; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \sum\limits_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathsf{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi) \;=\; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \sum\limits_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathrm{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\big(Pr(s \models \Phi \, \mathrm{U}^{\leqslant 0} \, \Psi)\big)_{s \in S} \;=\; \mathbf{b}_{\Psi}$

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \;=\; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \sum\limits_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathsf{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\big)_{s \in S} \;=\; \mathbf{b}_{\Psi}$
3. And $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\big)_{s \in S} \;=\; \mathbf{P}_{\Phi,\Psi} \cdot \big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\big)_{s \in S}$.

Bounded until (2)

Let $S_{=1} = Sat(\Psi)$, $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi))$, and $S_? = S \setminus (S_{=0} \cup S_{=1})$. Then:

$$Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \;=\; \begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ 0 & \text{if } s \in S_? \wedge n{=}0 \\ \sum\limits_{s' \in S} \mathbf{P}(s, s') \cdot Pr(s' \models \Phi \, \mathsf{U}^{\leqslant n-1} \, \Psi) & \text{otherwise} \end{cases}$$

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\right)_{s \in S} \;=\; \mathbf{b}_{\Psi}$
3. And $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\right)_{s \in S} \;=\; \mathbf{P}_{\Phi,\Psi} \cdot \left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\right)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\right)_{s \in S} \;=\; \mathbf{b}_{\Psi}$
3. And $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\right)_{s \in S} \;=\; \mathbf{P}_{\Phi,\Psi} \cdot \left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\right)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\right)_{s \in S} \; = \; \mathbf{b}_{\Psi}$
3. And $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\right)_{s \in S} \; = \; \mathbf{P}_{\Phi,\Psi} \cdot \left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\right)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\big)_{s \in S} \; = \; \mathbf{b}_{\Psi}$
3. And $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}_{\Phi,\Psi} \cdot \big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\big)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

1. In terms of matrix powers: $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}^{n}_{\Phi,\Psi} \cdot \mathbf{b}_{\Psi}$.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\big)_{s \in S} \; = \; \mathbf{b}_{\Psi}$
3. And $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}_{\Phi,\Psi} \cdot \big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\big)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

1. In terms of matrix powers: $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}^{n}_{\Phi,\Psi} \cdot \mathbf{b}_{\Psi}$.
    - Computing $\mathbf{P}^{n}_{\Phi,\Psi}$ in $\log_2 n$ steps is inefficient due to fill-in.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\big)_{s \in S} \; = \; \mathbf{b}_\Psi$
3. And $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}_{\Phi,\Psi} \cdot \big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\big)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

1. In terms of matrix powers: $\big(Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)\big)_{s \in S} \; = \; \mathbf{P}^n_{\Phi,\Psi} \cdot \mathbf{b}_\Psi$.
   - Computing $\mathbf{P}^n_{\Phi,\Psi}$ in $\log_2 n$ steps is inefficient due to fill-in.
   - That is to say, $\mathbf{P}^n_{\Phi,\Psi}$ is much less sparse than $\mathbf{P}_{\Phi,\Psi}$.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant 0} \, \Psi)\right)_{s \in S} \; = \; \mathbf{b}_\Psi$
3. And $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i+1} \, \Psi)\right)_{s \in S} \; = \; \mathbf{P}_{\Phi,\Psi} \cdot \left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant i} \, \Psi)\right)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

1. In terms of matrix powers: $\left(Pr(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)\right)_{s \in S} \; = \; \mathbf{P}^n_{\Phi,\Psi} \cdot \mathbf{b}_\Psi$.
   - Computing $\mathbf{P}^n_{\Phi,\Psi}$ in $\log_2 n$ steps is inefficient due to fill-in.
   - That is to say, $\mathbf{P}^n_{\Phi,\Psi}$ is much less sparse than $\mathbf{P}_{\Phi,\Psi}$.
2. $\mathbf{P}^n_{\Phi,\Psi} \cdot \mathbf{b}_\Psi \; = \; \left(Pr(s \models \bigcirc^{=n} \, \Psi)\right)_{s \in S_?}$ in $\mathcal{D}[S_{=0} \cup S_{=1}]$.
   - Where $\bigcirc^0 \, \Psi = \Psi$ and $\bigcirc^{i+1} \, \Psi = \bigcirc \left(\bigcirc^i \, \Psi\right)$.

Bounded until (3)

Algorithm

1. Let $\mathbf{P}_{\Phi,\Psi}$ be the probability matrix of $\mathcal{D}[S_{=0} \cup S_{=1}]$.
2. Then $\left(Pr(s \models \Phi\, \mathsf{U}^{\leqslant 0}\, \Psi)\right)_{s \in S} \;=\; \mathbf{b}_{\Psi}$
3. And $\left(Pr(s \models \Phi\, \mathsf{U}^{\leqslant i+1}\, \Psi)\right)_{s \in S} \;=\; \mathbf{P}_{\Phi,\Psi} \cdot \left(Pr(s \models \Phi\, \mathsf{U}^{\leqslant i}\, \Psi)\right)_{s \in S}$.
4. This requires n matrix-vector multiplications in total.

Remarks

1. In terms of matrix powers: $\left(Pr(s \models \Phi\, \mathsf{U}^{\leqslant n}\, \Psi)\right)_{s \in S} \;=\; \mathbf{P}^{n}_{\Phi,\Psi} \cdot \mathbf{b}_{\Psi}$.
   - Computing $\mathbf{P}^{n}_{\Phi,\Psi}$ in $\log_2 n$ steps is inefficient due to fill-in.
   - That is to say, $\mathbf{P}^{n}_{\Phi,\Psi}$ is much less sparse than $\mathbf{P}_{\Phi,\Psi}$.
2. $\mathbf{P}^{n}_{\Phi,\Psi} \cdot \mathbf{b}_{\Psi} \;=\; \left(Pr(s \models \bigcirc^{=n}\, \Psi)\right)_{s \in S_?}$ in $\mathcal{D}[S_{=0} \cup S_{=1}]$.
   - Where $\bigcirc^{0}\, \Psi = \Psi$ and $\bigcirc^{i+1}\, \Psi = \bigcirc\, (\bigcirc^{i}\, \Psi)$.
   - This thus amounts to a transient analysis in DTMC $\mathcal{D}[S_{=0} \cup S_{=1}]$.

Optimization

The above procedure used:

- $S_{=1} = Sat(\Psi)$, and
- $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi)) = Sat(\neg\Phi \wedge \neg\Psi)$, and
- perform the matrix-vector multiplications on the remaining states

Optimization

The above procedure used:

- $S_{=1} = Sat(\Psi)$, and
- $S_{=0} = S \setminus (Sat(\Phi) \cup Sat(\Psi)) = Sat(\neg\Phi \wedge \neg\Psi)$, and
- perform the matrix-vector multiplications on the remaining states

This can be optimized (in practice) by enlarging $S_{=0}$ and$S_{=1}$:

- $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \,\mathsf{U}\, \Psi))$, obtained by a graph analysis
- $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \,\mathsf{U}\, \Psi))$, obtained by a graph analysis too, and
- perform the matrix-vector multiplications on the remaining states.

Example

Until

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathrm{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathrm{U} \, \Psi) \in J$.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U} \, \Psi) \in J$.

Algorithm

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \,\mathsf{U}\, \Psi)$ if and only if $Pr(s \models \Phi \,\mathsf{U}\, \Psi) \;\in\; J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \,\mathsf{U}\, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \,\mathsf{U}\, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathrm{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathrm{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathrm{U} \, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \, \mathrm{U} \, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Importance of pre-computation using graph analysis

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathrm{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathrm{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathrm{U} \, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \, \mathrm{U} \, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Importance of pre-computation using graph analysis

1. Ensures unique solution to linear equation system.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Importance of pre-computation using graph analysis

1. Ensures unique solution to linear equation system.
2. Reduces the number of variables in the linear equation system.

Until

Recall that: $s \models \mathbb{P}_J(\Phi \,\mathsf{U}\, \Psi)$ if and only if $Pr(s \models \Phi \,\mathsf{U}\, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \,\mathsf{U}\, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \,\mathsf{U}\, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Importance of pre-computation using graph analysis

1. Ensures unique solution to linear equation system.
2. Reduces the number of variables in the linear equation system.
3. Gives exact results for the states in $S_{=1}$ and $S_{=0}$ (i.e., no round-off).

Until

Recall that: $s \models \mathbb{P}_J(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr(s \models \Phi \, \mathsf{U} \, \Psi) \in J$.

Algorithm

1. Determine $S_{=1} = Sat(\mathbb{P}_{=1}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.
2. Determine $S_{=0} = Sat(\mathbb{P}_{=0}(\Phi \, \mathsf{U} \, \Psi))$ by a graph analysis.
3. Then solve a linear equation system over all remaining states.

Importance of pre-computation using graph analysis

1. Ensures unique solution to linear equation system.
2. Reduces the number of variables in the linear equation system.
3. Gives exact results for the states in $S_{=1}$ and $S_{=0}$ (i.e., no round-off).
4. For qualitative properties, no further computation is needed.

Example

Overview

Time complexity

Time complexity

Let $|\Phi|$ be the size of Φ, i.e., the number of logical and temporal operators in Φ.

Time complexity

Let $|\Phi|$ be the size of Φ, i.e., the number of logical and temporal operators in Φ.

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

Time complexity

Let $|\Phi|$ be the size of Φ, i.e., the number of logical and temporal operators in Φ.

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\left(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \right)$$

where $n_{\max} = \max\{ n \mid \Psi_1 \, \mathsf{U}^{\leqslant n} \Psi_2 \text{ occurs in } \Phi \}$ with and $n_{\max} = 1$ if Φ does not contain a bounded until-operator.

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\left(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \right).$$

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\big(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \big).$$

Proof (sketch)

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\big(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\mathrm{max}} \cdot |\Phi| \big).$$

Proof (sketch)

1. For each node in the parse tree, a model-checking is performed; this yields a linear complexity in $|\Phi|$.

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\big(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \big).$$

Proof (sketch)

1. For each node in the parse tree, a model-checking is performed; this yields a linear complexity in $|\Phi|$.
2. The worst-case operator is (unbounded) until.

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\left(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \right).$$

Proof (sketch)

1. For each node in the parse tree, a model-checking is performed; this yields a linear complexity in $|\Phi|$.
2. The worst-case operator is (unbounded) until.
   2.1 Determining $S_{=0}$ and $S_{=1}$ can be done in linear time.

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi|).$$

Proof (sketch)

1. For each node in the parse tree, a model-checking is performed; this yields a linear complexity in $|\Phi|$.
2. The worst-case operator is (unbounded) until.
   2.1 Determining $S_{=0}$ and $S_{=1}$ can be done in linear time.
   2.2 Direct methods to solve linear equation systems are in $\Theta(|S_?|^3)$.

Time complexity

Time complexity of PCTL model checking

For finite DTMC $\mathcal{D}$ and PCTL state-formula Φ, the PCTL model-checking problem can be solved in time

$$\mathcal{O}\left(\xrightarrow{p} oly(size(\mathcal{D})) \cdot n_{\max} \cdot |\Phi| \right).$$

Proof (sketch)

1. For each node in the parse tree, a model-checking is performed; this yields a linear complexity in $|\Phi|$.
2. The worst-case operator is (unbounded) until.
   2.1 Determining $S_{=0}$ and $S_{=1}$ can be done in linear time.
   2.2 Direct methods to solve linear equation systems are in $\Theta(|S_?|^3)$.
3. Strictly speaking, $\mathrm{U}^{\leqslant n}$ could be more expensive for large n.

   But it remains polynomial, and n is small in practice.

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution
  - selected router flips a biased coin:

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution
  - selected router flips a biased coin:
    - with probability $1 - p$: direct delivery to final destination
    - otherwise: select a next router randomly (uniformly)

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution
  - selected router flips a biased coin:
    - with probability $1 - p$: direct delivery to final destination
    - otherwise: select a next router randomly (uniformly)
  - once a routing path has been established, use it until crowd changes

Example: Crowds protocol

Security: Crowds protocol [Reiter & Rubin, 1998]

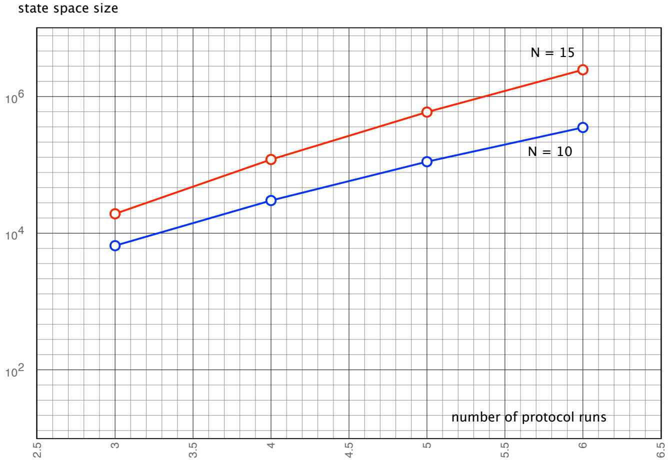
- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution
  - selected router flips a biased coin:
    - with probability $1 - p$: direct delivery to final destination
    - otherwise: select a next router randomly (uniformly)
  - once a routing path has been established, use it until crowd changes
- Rebuild routing paths on crowd changes

Example: Crowds protocol

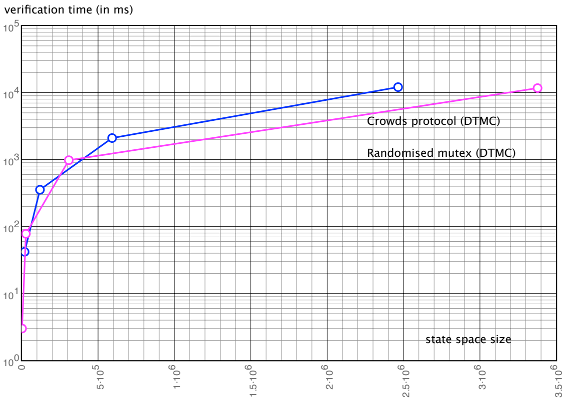
Security: Crowds protocol [Reiter & Rubin, 1998]

- A protocol for anonymous web browsing (variants: mCrowds, BT-Crowds)
- Hide user's communication by random routing within a crowd
  - sender selects a crowd member randomly using a uniform distribution
  - selected router flips a biased coin:
    - with probability $1 - p$: direct delivery to final destination
    - otherwise: select a next router randomly (uniformly)
  - once a routing path has been established, use it until crowd changes
- Rebuild routing paths on crowd changes
- Property: Crowds protocol ensures "probable innocence":
  - probability real sender is discovered $< \frac{1}{2}$ if $N \geqslant \frac{p}{p-\frac{1}{2}} \cdot (c+1)$
  - where N is crowd's size and c is number of corrupt crowd members

State space growth



Some practical verification times



- command-line tool MRMC ran on a Pentium 4, 2.66 GHz, 1 GB RAM laptop.
- PCTL formula $\mathbb{P}_{\leqslant p}(\Diamond obs)$ where *obs* holds when the sender's id is detected.

Overview

Summary

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.
- PCTL model checking is performed by a recursive descent over Φ.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.
- PCTL model checking is performed by a recursive descent over Φ.
- The next operator amounts to a single matrix-vector multiplication.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.
- PCTL model checking is performed by a recursive descent over Φ.
- The next operator amounts to a single matrix-vector multiplication.
- The bounded-until operator $\mathrm{U}^{\leqslant n}$ amounts to n matrix-vector multiplications.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.
- PCTL model checking is performed by a recursive descent over Φ.
- The next operator amounts to a single matrix-vector multiplication.
- The bounded-until operator $\mathsf{U}^{\leqslant n}$ amounts to n matrix-vector multiplications.
- The until-operator amounts to solving a linear equation system.

Summary

- PCTL is a branching-time logic with key operator $\mathbb{P}_J(\varphi)$.
- Sets of paths fulfilling PCTL path-formula φ are measurable.
- PCTL model checking is performed by a recursive descent over Φ.
- The next operator amounts to a single matrix-vector multiplication.
- The bounded-until operator $\mathsf{U}^{\leqslant n}$ amounts to n matrix-vector multiplications.
- The until-operator amounts to solving a linear equation system.
- Worst-case time complexity of $\mathcal{D} \models \Phi$ is polynomial in $|\mathcal{D}|$ and linear in $|\Phi|$.