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/

December 2, 2015

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 can also be used to specify properties over MDPs.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It can also be used to specify properties over MDPs.
- 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 can also be used to specify properties over MDPs.
- It is a branching-time temporal logic based on CTL.
- Formula interpretation is Boolean, i.e., a state satisfies a formula or not.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It can also be used to specify properties over MDPs.
- It is a branching-time temporal logic based on CTL.
- Formula interpretation is Boolean, i.e., a state satisfies a formula 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 can also be used to specify properties over MDPs.
- It is a branching-time temporal logic based on CTL.
- Formula interpretation is Boolean, i.e., a state satisfies a formula 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.

Probabilistic Computation Tree Logic

- PCTL is a language for formally specifying properties over DTMCs.
- It can also be used to specify properties over MDPs.
- It is a branching-time temporal logic based on CTL.
- Formula interpretation is Boolean, i.e., a state satisfies a formula 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.
  - ranges over all possible resolutions of nondeterminism.

PCTL syntax [Bianco & De Alfaro, 1995]

Probabilistic Computation Tree Logic: Syntax

PCTL consists of state- and path-formulas.

PCTL syntax [Bianco & De Alfaro, 1995]

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 [Bianco & De Alfaro, 1995]

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 [Bianco & De Alfaro, 1995]

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 $J \subseteq [0,1]$, $J \neq \varnothing$.

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

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

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 $J \subseteq [0, 1]$, $J \neq \varnothing$.

- 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 $J \subseteq [0,1]$, $J \neq \varnothing$.

- 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

- $s_0 \alpha_0 s_1 \alpha_1 s_2 \alpha_2 \ldots \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi$ if Φ holds until Ψ holds within n steps (where $s_i \alpha_{i+1}$ is a single step).

Probabilistic Computation Tree Logic

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

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

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

- 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

- $s_0\alpha_0 s_1\alpha_1 s_2\alpha_2 \ldots \models \Phi \,\mathrm{U}^{\leqslant n}\, \Psi$ if Φ holds until Ψ holds within n steps (where $s_i\alpha_{i+1}$ is a single step).
- $s \models \mathbb{P}_J(\varphi)$ if the probability under all policies that paths starting in s fulfill φ lies in J.

Overview

Markov decision process (MDP)

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\text{init}} : S \rightarrow [0, 1]$

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\text{init}} : S \to [0, 1]$
- Act is a finite set of actions

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\text{init}} : S \to [0, 1]$
- Act is a finite set of actions
- $\mathbf{P} : S \times Act \times S \to [0, 1]$, transition probability function

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\mathrm{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\mathrm{init}} : S \to [0, 1]$
- Act is a finite set of actions
- $\mathbf{P} : S \times Act \times S \to [0, 1]$, transition probability function such that:

$$\text{for all } s \in S \text{ and } \alpha \in Act : \sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{ 0, 1 \}$$

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\text{init}} : S \rightarrow [0,1]$
- Act is a finite set of actions
- $\mathbf{P} : S \times Act \times S \rightarrow [0,1]$, transition probability function such that:

$$\text{for all } s \in S \text{ and } \alpha \in Act : \sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{ 0, 1 \}$$

- AP is a set of atomic propositions and labeling $L : S \rightarrow 2^{AP}$.

Markov decision process (MDP)

Markov decision process

An MDP $\mathcal{M}$ is a tuple $(S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$ where

- S is a countable set of states with initial distribution $\iota_{\text{init}} : S \to [0, 1]$
- Act is a finite set of actions
- $\mathbf{P} : S \times Act \times S \to [0, 1]$, transition probability function such that:

$$\text{for all } s \in S \text{ and } \alpha \in Act : \sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{ 0, 1 \}$$

- AP is a set of atomic propositions and labeling $L : S \to 2^{AP}$.

Assumption: in each state at least one action is enabled.

PCTL semantics (1)

PCTL semantics (1)

Notation

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

PCTL semantics (1)

Notation

$\mathcal{M}, s \models \Phi$ if and only if state-formula Φ holds in state s of (possibly infinite) MDP $\mathcal{M}$. As $\mathcal{M}$ 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

$\mathcal{M}, s \models \Phi$ if and only if state-formula Φ holds in state s of (possibly infinite) MDP $\mathcal{M}$. As $\mathcal{M}$ 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:

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\,\Phi & \text{iff} & \text{not } (s \models \Phi) \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \mathbb{P}_J(\varphi) & \text{iff} & \text{for all policies } \mathfrak{S} \text{ on } \mathcal{M}.\ Pr^{\mathfrak{S}}(s \models \varphi) \in J
\end{array}
$$

where $Pr^{\mathfrak{S}}(s \models \varphi) = Pr_s^{\mathfrak{S}}\{\pi \in Paths(s) \mid \pi \models \varphi\}$.

Semantics of $\mathbb{P}$-operator

Semantics of $\mathbb{P}$-operator

The probabilistic operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for *all* policies.

Semantics of $\mathbb{P}$-operator

The probabilistic operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for *all* policies.

In particular, we have

$$s \models \mathbb{P}_{\leqslant p}(\varphi) \quad \text{iff} \quad Pr^{\max}(s \models \varphi) \leqslant p \quad \text{iff} \quad \sup_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p$$

Semantics of $\mathbb{P}$-operator

The probabilistic operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for *all* policies.

In particular, we have

$$s \models \mathbb{P}_{\leqslant p}(\varphi) \;\text{ iff }\; Pr^{\max}(s \models \varphi) \leqslant p \;\text{ iff }\; \sup_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p$$

and, dually,

$$s \models \mathbb{P}_{\geqslant p}(\varphi) \;\text{ iff }\; Pr^{\min}(s \models \varphi) \geqslant p \;\text{ iff }\; \inf_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \geqslant p.$$

Semantics of $\mathbb{P}$-operator

The probabilistic operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for *all* policies.

In particular, we have

$$s \models \mathbb{P}_{\leqslant p}(\varphi) \text{ iff } Pr^{\max}(s \models \varphi) \leqslant p \text{ iff } \sup_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p$$

and, dually,

$$s \models \mathbb{P}_{\geqslant p}(\varphi) \text{ iff } Pr^{\min}(s \models \varphi) \geqslant p \text{ iff } \inf_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \geqslant p.$$

For finite MDPs we have:

$$Pr^{\max}(s \models \varphi) = \max_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi)$$

Semantics of $\mathbb{P}$-operator

The probabilistic operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for *all* policies.

In particular, we have

$$s \models \mathbb{P}_{\leqslant p}(\varphi) \ \text{ iff } \ Pr^{\max}(s \models \varphi) \leqslant p \ \text{ iff } \ \sup_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p$$

and, dually,

$$s \models \mathbb{P}_{\geqslant p}(\varphi) \ \text{ iff } \ Pr^{\min}(s \models \varphi) \geqslant p \ \text{ iff } \ \inf_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \geqslant p.$$

For finite MDPs we have:

$$Pr^{\max}(s \models \varphi) = \max_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi) \text{ and } Pr^{\min}(s \models \varphi) = \min_{\mathfrak{S}} Pr^{\mathfrak{S}}(s \models \varphi)$$

as for any finite MDP an fm-policy exists that maximises or minimises φ.

PCTL semantics (2)

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ be an infinite path in (possibly infinite) MDP $\mathcal{M}$. Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ be an infinite path in (possibly infinite) MDP $\mathcal{M}$. 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 $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ be an infinite path in (possibly infinite) MDP $\mathcal{M}$. Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

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

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

PCTL semantics (2)

Satisfaction relation for path formulas

Let $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ be an infinite path in (possibly infinite) MDP $\mathcal{M}$. Recall that $\pi[i] = s_i$ denotes the $(i{+}1)$-st state along π.

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

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

There is indeed no difference with the PCTL semantics for DTMC paths.

Equivalence of PCTL formulas

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold.

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\mathrm{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\mathrm{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\mathrm{MDP}} \Psi$ implies $\Phi \equiv_{\mathrm{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\mathrm{MC}} \neg\mathbb{P}_{>p}(\varphi)$.

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\text{MC}} \neg\mathbb{P}_{>p}(\varphi)$. But, $\mathbb{P}_{\leqslant p}(\varphi) \not\equiv_{\text{MDP}} \neg\mathbb{P}_{>p}(\varphi)$.

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\text{MC}} \neg\mathbb{P}_{>p}(\varphi)$. But, $\mathbb{P}_{\leqslant p}(\varphi) \not\equiv_{\text{MDP}} \neg\mathbb{P}_{>p}(\varphi)$.

$$s \models \mathbb{P}_{\leqslant p}(\varphi) \quad \text{iff} \quad Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p \text{ for } \textit{all} \text{ policies } \mathfrak{S}\text{, but}$$

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\text{MC}} \neg\mathbb{P}_{>p}(\varphi)$. But, $\mathbb{P}_{\leqslant p}(\varphi) \not\equiv_{\text{MDP}} \neg\mathbb{P}_{>p}(\varphi)$.

$s \models \mathbb{P}_{\leqslant p}(\varphi)$ iff $Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p$ for *all* policies $\mathfrak{S}$, but

$s \models \neg\mathbb{P}_{>p}(\varphi)$ iff not $s \models \mathbb{P}_{>p}(\varphi)$

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\text{MC}} \neg\mathbb{P}_{>p}(\varphi)$. But, $\mathbb{P}_{\leqslant p}(\varphi) \not\equiv_{\text{MDP}} \neg\mathbb{P}_{>p}(\varphi)$.

$$
\begin{array}{lll}
s \models \mathbb{P}_{\leqslant p}(\varphi) & \text{iff} & Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p \text{ for } \textit{all} \text{ policies } \mathfrak{S}, \text{ but} \\
s \models \neg\mathbb{P}_{>p}(\varphi) & \text{iff} & \text{not } s \models \mathbb{P}_{>p}(\varphi) \\
 & \text{iff} & \text{not } \left(Pr^{\mathfrak{S}}(s \models \varphi) > p \text{ for all policies } \mathfrak{S} \right)
\end{array}
$$

Equivalence of PCTL formulas

PCTL equivalence

$\Phi \equiv_{\text{MDP}} \Psi$ if and only if for all MDPs $\mathcal{M}$, it holds: $Sat_{\mathcal{M}}(\Phi) = Sat_{\mathcal{M}}(\Psi)$.

$\Phi \equiv_{\text{MC}} \Psi$ if and only if for all DTMCs $\mathcal{D}$, it holds: $Sat_{\mathcal{D}}(\Phi) = Sat_{\mathcal{D}}(\Psi)$.

Since any DTMC is an MDP, it follows: $\Phi \equiv_{\text{MDP}} \Psi$ implies $\Phi \equiv_{\text{MC}} \Psi$.

The converse, however, does not hold. For instance, for $p < 1$, we have $\mathbb{P}_{\leqslant p}(\varphi) \equiv_{\text{MC}} \neg\mathbb{P}_{>p}(\varphi)$. But, $\mathbb{P}_{\leqslant p}(\varphi) \not\equiv_{\text{MDP}} \neg\mathbb{P}_{>p}(\varphi)$.

$$
\begin{array}{lll}
s \models \mathbb{P}_{\leqslant p}(\varphi) & \text{iff} & Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p \text{ for } \textit{all} \text{ policies } \mathfrak{S}, \text{ but} \\
s \models \neg\mathbb{P}_{>p}(\varphi) & \text{iff} & \text{not } s \models \mathbb{P}_{>p}(\varphi) \\
 & \text{iff} & \text{not } \left(\, Pr^{\mathfrak{S}}(s \models \varphi) > p \text{ for all policies } \mathfrak{S} \, \right) \\
 & \text{iff} & Pr^{\mathfrak{S}}(s \models \varphi) \leqslant p \text{ for } \textit{some} \text{ policy } \mathfrak{S}.
\end{array}
$$

Overview

PCTL model checking

PCTL model checking

PCTL model checking problem

PCTL model checking

PCTL model checking problem

Input: a finite MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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.

PCTL model checking

PCTL model checking problem

Input: a finite MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{\text{init}}, AP, L)$, 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 $Sat(\Phi)$.

Core model checking algorithm

Core model checking algorithm

Propositional formulas

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Probabilistic operator $\mathbb{P}$

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_{\leqslant p}(\varphi))$, the probability $Pr^{\max}(s \models \varphi)$ needs to be established.

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_{\leqslant p}(\varphi))$, the probability $Pr^{\max}(s \models \varphi)$ needs to be established. Then

$$Sat(\mathbb{P}_{\leqslant p}(\varphi)) \; = \; \{ s \in S \mid Pr^{\max}(s \models \varphi) \leqslant p \}.$$

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_{\leqslant p}(\varphi))$, the probability $Pr^{\max}(s \models \varphi)$ needs to be established. Then

$$Sat(\mathbb{P}_{\leqslant p}(\varphi)) \;=\; \{ s \in S \mid Pr^{\max}(s \models \varphi) \leqslant p \}.$$

The same holds for strict upper bounds $< p$.

Core model checking algorithm

Propositional formulas

$Sat(\cdot)$ is defined by structural induction as for PCTL on DTMCs.

Probabilistic operator $\mathbb{P}$

In order to determine whether $s \in Sat(\mathbb{P}_{\leqslant p}(\varphi))$, the probability $Pr^{\max}(s \models \varphi)$ needs to be established. Then

$$Sat(\mathbb{P}_{\leqslant p}(\varphi)) \;=\; \{s \in S \mid Pr^{\max}(s \models \varphi) \leqslant p\}.$$

The same holds for strict upper bounds $< p$.

Similarly, lower bounds amount to determining $Pr^{\min}(s \models \varphi)$, e.g.,

$$Sat(\mathbb{P}_{>p}(\varphi)) \;=\; \{s \in S \mid Pr^{\min}(s \models \varphi) > p\}.$$

The next-step operator

The next-step operator

Recall that: $s \models \mathbb{P}_{\leqslant p}(\bigcirc \Phi)$ if and only if $Pr^{\max}(s \models \bigcirc \Phi) \leqslant p$.

The next-step operator

Recall that: $s \models \mathbb{P}_{\leqslant p}(\bigcirc \Phi)$ if and only if $Pr^{\max}(s \models \bigcirc \Phi) \leqslant p$.

Lemma

$$Pr^{\max}(s \models \bigcirc \Phi) \; = \; \max\{ \sum_{t \, \in \, Sat(\Phi)} \mathbf{P}(s, \alpha, t) \mid \alpha \in Act(s) \}.$$

The next-step operator

Recall that: $s \models \mathbb{P}_{\leqslant p}(\bigcirc \Phi)$ if and only if $Pr^{\max}(s \models \bigcirc \Phi) \leqslant p$.

Lemma

$$Pr^{\max}(s \models \bigcirc \Phi) \; = \; \max\{ \sum_{t \in Sat(\Phi)} \mathbf{P}(s, \alpha, t) \mid \alpha \in Act(s) \}.$$

Algorithm

Determine $x_s = Pr^{\max}(s \models \bigcirc \Phi)$ and return $Sat(\mathbb{P}_{\leqslant p}(\bigcirc \Phi)) = \{ s \in S \mid x_s \leqslant p \}$.

The next-step operator

Recall that: $s \models \mathbb{P}_{\leqslant p}(\bigcirc \Phi)$ if and only if $Pr^{\max}(s \models \bigcirc \Phi) \leqslant p$.

Lemma

$$Pr^{\max}(s \models \bigcirc \Phi) \;=\; \max\Big\{ \sum_{t \in Sat(\Phi)} \mathbf{P}(s, \alpha, t) \mid \alpha \in Act(s) \Big\}.$$

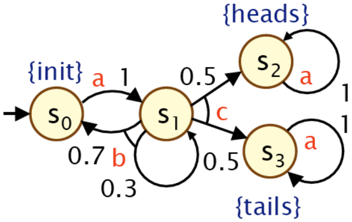
Algorithm

Determine $x_s = Pr^{\max}(s \models \bigcirc \Phi)$ and return $Sat(\mathbb{P}_{\leqslant p}(\bigcirc \Phi)) = \{ s \in S \mid x_s \leqslant p \}$.

The case for minimal probabilities is similar and omitted here.

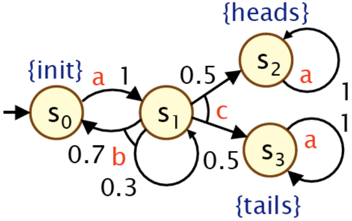
Example

Consider MDP:



Example

Consider MDP:

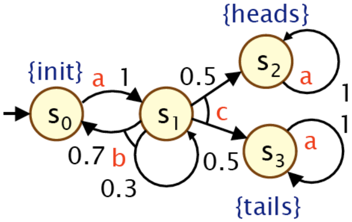


and PCTL-formula:

$$\mathbb{P}_{\geqslant \frac{1}{2}}\left(\bigcirc\, \mathit{heads}\right)$$

Example

Consider MDP:



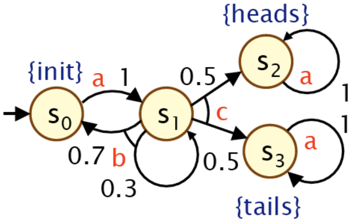
and PCTL-formula:

$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) = \{ s_2 \}$

Example

Consider MDP:



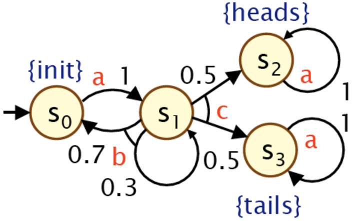
and PCTL-formula:

$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) \ = \ \{ \, s_2 \, \}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$

Example

Consider MDP:



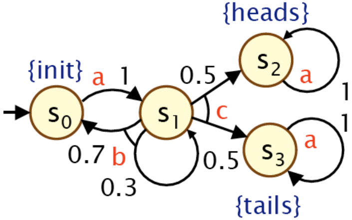
and PCTL-formula:

$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) \;=\; \{\, s_2 \,\}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

Example

Consider MDP:



and PCTL-formula:

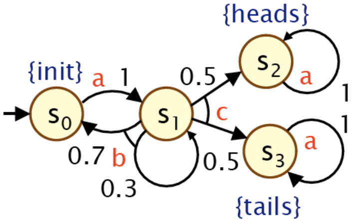
$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) \;=\; \{\, s_2 \,\}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

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

Example

Consider MDP:



and PCTL-formula:

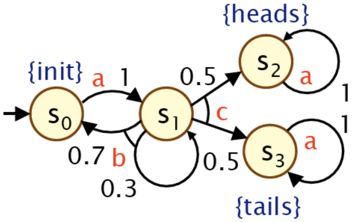
$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) \;=\; \{\, s_2 \,\}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

$$\left(Pr^{\min}(s \models \bigcirc\, \Phi)\right)_{s \in S} \;=\; \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

Example

Consider MDP:



and PCTL-formula:

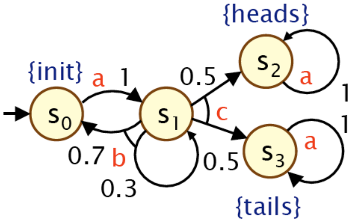
$$\mathbb{P}_{\geqslant \frac{1}{2}} (\bigcirc \, heads)$$

1. $Sat(heads) \; = \; \{ \, s_2 \, \}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc \, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

$$\left(Pr^{\min}(s \models \bigcirc \, \Phi) \right)_{s \in S} \; = \; \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Example

Consider MDP:



and PCTL-formula:

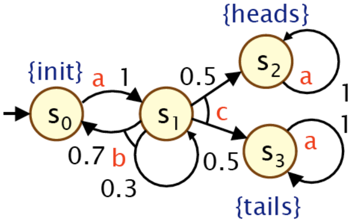
$$\mathbb{P}_{\geqslant \frac{1}{2}} (\bigcirc\, heads)$$

1. $Sat(heads) \;=\; \{\, s_2 \,\}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

$$\left(Pr^{\min}(s \models \bigcirc\, \Phi)\right)_{s \in S} \;=\; \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \;=\; \left(\begin{array}{c} 0 \\ \hline 0 \\ 0.5 \\ \hline 1 \\ \hline 0 \end{array}\right)$$

Example

Consider MDP:



and PCTL-formula:

$$\mathbb{P}_{\geqslant \frac{1}{2}}(\bigcirc\, heads)$$

1. $Sat(heads) \;=\; \{\, s_2 \,\}$
2. $x_{s_1} = Pr^{\min}(s_1 \models \bigcirc\, heads) = \min(0, 0.5) = 0$
3. Applying that to all states yields:

$$\left(Pr^{\min}(s \models \bigcirc\, \Phi)\right)_{s \in S} = \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \left(\begin{array}{c} 0 \\ \hline 0 \\ 0.5 \\ \hline 1 \\ \hline 0 \end{array}\right)$$

4. Thus: $Sat(\mathbb{P}_{\geqslant 0.5}(\bigcirc\, heads)) \;=\; \{\, s_2 \,\}$.

Bounded until (1)

Bounded until (1)

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \geqslant p$.

Bounded until (1)

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \geqslant p$.

Lemma

Bounded until (1)

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \geqslant p$.

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: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathrm{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi) \geqslant p$.

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: $Pr^{\min}(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi)$ equals

$$\begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ & \end{cases}$$

Bounded until (1)

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \geqslant p$.

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Bounded until (1)

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi) \geqslant p$.

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

The case for maximal probabilities is analogous.

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathrm{U}^{\leqslant n} \, \Psi)$ equals

$$\begin{cases} 1 & \text{if } s \in S_{=1} \\ 0 & \text{if } s \in S_{=0} \\ & \\ & \end{cases}$$

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Algorithm

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Algorithm

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

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Algorithm

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

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Algorithm

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

Bounded until (2)

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: $Pr^{\min}(s \models \Phi \, \mathsf{U}^{\leqslant n} \, \Psi)$ equals

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

Algorithm

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

Example

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

Algorithm

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathrm{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathrm{U} \, \Psi) \geqslant p$.

Algorithm

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

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathrm{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathrm{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Importance of pre-computation

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Importance of pre-computation

1. Determining $S_{=0}$ ensures unique solution of linear program.

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Importance of pre-computation

1. Determining $S_{=0}$ ensures unique solution of linear program.
2. Determining $S_{=1}$ reduces the number of variables in the linear program.

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Importance of pre-computation

1. Determining $S_{=0}$ ensures unique solution of linear program.
2. Determining $S_{=1}$ reduces the number of variables in the linear program.
3. Gives exact results for the states in $S_{=1}$ and $S_{=0}$ (i.e., no round-off).

Until

Recall that: $s \models \mathbb{P}_{\geqslant p}(\Phi \, \mathsf{U} \, \Psi)$ if and only if $Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) \geqslant p$.

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 program (or use value iteration) over all remaining states.

Importance of pre-computation

1. Determining $S_{=0}$ ensures unique solution of linear program.
2. Determining $S_{=1}$ reduces the number of variables in the linear program.
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.

Precomputations

Qualitative reachability

Precomputations

Qualitative reachability

1. Determine all states for which probability is zero
   1.1 minimum: $\{ s \in S \mid Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) = 0 \}$
   1.2 maximum: $\{ s \in S \mid Pr^{\max}(s \models \Phi \, \mathsf{U} \, \Psi) = 0 \}$
2. Determine all states for which probability is one
   2.1 minimum: $\{ s \in S \mid Pr^{\min}(s \models \Phi \, \mathsf{U} \, \Psi) = 1 \}$
   2.2 maximum: $\{ s \in S \mid Pr^{\max}(s \models \Phi \, \mathsf{U} \, \Psi) = 1 \}$
3. Then solve a linear program (or use value iteration) over all remaining states.

The first case has been treated in the previous lecture (for $\Diamond G$).

Qualitative reachability

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \lozenge G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \lozenge G) < 1$

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists \Diamond G)$; this can be done by a graph analysis

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists \Diamond G)$; this can be done by a graph analysis
2. Remove all actions α from state u for which $Post(s, \alpha) \cap U_0 \neq \varnothing$

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists \Diamond G)$; this can be done by a graph analysis
2. Remove all actions α from state u for which $Post(s, \alpha) \cap U_0 \neq \varnothing$
3. If after removal of actions $Act(u) = \varnothing$, then remove state u

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists \Diamond G)$; this can be done by a graph analysis
2. Remove all actions α from state u for which $Post(s, \alpha) \cap U_0 \neq \varnothing$
3. If after removal of actions $Act(u) = \varnothing$, then remove state u
4. Repeat this procedure for all states, yielding the new MDP $\mathcal{M}'$

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists\Diamond\, G)$; this can be done by a graph analysis
2. Remove all actions α from state u for which $Post(s, \alpha) \cap U_0 \neq \varnothing$
3. If after removal of actions $Act(u) = \varnothing$, then remove state u
4. Repeat this procedure for all states, yielding the new MDP $\mathcal{M}'$
5. As this may yield new states from which G is unreachable, repeat the above steps until all states can reach G

Qualitative reachability

- Goal is to compute $\{ s \in S \mid Pr^{\max}(s \models \Diamond G) = 1 \}$
- First make all states in G absorbing, i.e., $\mathbf{P}(s, \alpha_s, s) = 1$
- Iteratively remove state t for which $Pr^{\max}(t \models \Diamond G) < 1$

Sketch of algorithm

1. Let $U_0 = S \setminus Sat(\exists \Diamond G)$; this can be done by a graph analysis
2. Remove all actions α from state u for which $Post(s, \alpha) \cap U_0 \neq \varnothing$
3. If after removal of actions $Act(u) = \varnothing$, then remove state u
4. Repeat this procedure for all states, yielding the new MDP $\mathcal{M}'$
5. As this may yield new states from which G is unreachable, repeat the above steps until all states can reach G

This procedure is quadratic in the size of the MDP.

Algorithm

Algorithm 45 Computing the set of states s with $Pr^{\max}(s \models \Diamond B) = 1$

Input: MDP $\mathcal{M}$ with finite state space S, $B \subseteq S$ for $s \in B : Act(s) = \{ \alpha_s \}$ and $\mathbf{P}(s, \alpha_s, s) = 1$ (i.e., B is absorbing)
Output: $\{ s \in S \mid Pr^{\max}(s \models \Diamond B) = 1 \}$

$U := \{ s \in S \mid s \not\models \exists \Diamond B \}$;
repeat
  $R := U$;
  while $R \neq \varnothing$ **do**
    let $u \in R$;
    $R := R \setminus \{ u \}$;
    for all $(t, \alpha) \in Pre(u)$ such that $t \notin U$ **do**
      remove α from $Act(t)$;
      if $Act(t) = \varnothing$ **then**
        $R := R \cup \{ t \}$;
        $U := U \cup \{ t \}$;
      fi
    od
    (* all incoming edges of u have been removed *)
    remove u and its outgoing edges from $\mathcal{M}$
  od
  (* determine the states s that cannot reach B in the modified MDP *)
  $U := \{s \in S \setminus U \mid s \not\models \exists \Diamond B\}$;
until $U = \varnothing$
(* all states can reach B in the generated sub-MDP of $\mathcal{M}$ *)
return all states in the remaining MDP

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 of MDPs

For finite MDP $\mathcal{M}$ 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 of MDPs

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

$$\mathcal{O}\big(\, poly(size(\mathcal{M})) \cdot n_{\max} \cdot |\Phi| \,\big)$$

where $n_{\max} \;=\; \max\{\, n \mid \Psi_1 \, \mathsf{U}^{\leqslant n} \, \Psi_2 \text{ occurs in } \Phi \,\}$ with $\max \varnothing = 1$.

Overview

Dining cryptographers problem [Chaum, 1988]

Problem statement

Dining cryptographers problem [Chaum, 1988]

Problem statement

- Three cryptographers gather around a table for dinner.

Dining cryptographers problem [Chaum, 1988]

Problem statement

- Three cryptographers gather around a table for dinner.
- The waiter informs them that the meal has been paid by someone, who could be one of the cryptographers or their master.

Dining cryptographers problem [Chaum, 1988]

Problem statement

- Three cryptographers gather around a table for dinner.
- The waiter informs them that the meal has been paid by someone, who could be one of the cryptographers or their master.
- The cryptographers respect each other's right to make an anonymous payment, but want to find out whether the master paid or not.

Dining cryptographers problem [Chaum, 1988]

Problem statement

- Three cryptographers gather around a table for dinner.
- The waiter informs them that the meal has been paid by someone, who could be one of the cryptographers or their master.
- The cryptographers respect each other's right to make an anonymous payment, but want to find out whether the master paid or not.

Is it possible to obtain this information without revealing the identity of the cryptographer that paid?

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Dining cryptographers problem

Dining cryptographer's protocol

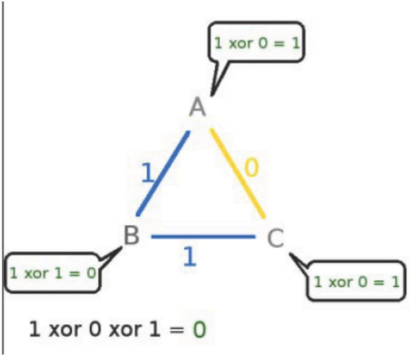
1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

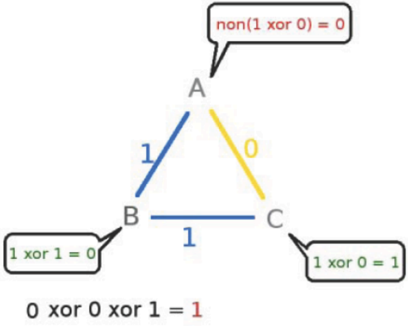
Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Claim

An odd number of agrees indicates that the master paid, while an even number indicates that a cryptographer paid.

Dining cryptographers problem





Example scenario in which master paid (left) or cryptographer A paid (right) and provides a misleading vote.

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Generalisation

The dining cryptographer's protocol can be generalised to any number N of cryptographers.

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Generalisation

The dining cryptographer's protocol can be generalised to any number N of cryptographers. Then:

- if N is odd, then an odd number of agrees indicates that the master paid while an even number indicates that a cryptographer paid.

Dining cryptographers problem

Dining cryptographer's protocol

1. Each cryptographer flips an unbiased coin and only informs the cryptographer on the right of the outcome.
2. Each cryptographer states whether the two coins that it can see—the one it flipped and the one the left-hand neighbour flipped—are the same (agree) or different (disagree).

Caveat: if a cryptographer actually paid for the dinner, then it instead states the opposite (disagree if the coins are the same and agree if the coins are different).

Generalisation

The dining cryptographer's protocol can be generalised to any number N of cryptographers. Then:

- if N is odd, then an odd number of agrees indicates that the master paid while an even number indicates that a cryptographer paid.
- if N is even, then an even number of agrees indicates that the master paid while an odd number indicates that a cryptographer paid.

MDP generation times

MDP generation times

N:	Model:		Construction time (s):
	States:	Transitions:	
3	286	585	0.001
4	1,733	4,580	0.01
5	9,876	32,315	0.03
6	54,055	211,566	0.07
7	287,666	1,312,045	0.11
8	1,499,657	7,813,768	0.22
9	7,695,856	45,103,311	0.34
10	39,005,611	253,985,650	0.52
15	115,553,171,626	1,128,594,416,085	3.27
20	304,287,522,253,461	3,962,586,180,540,340	13.48

The number of states and transitions in the MDP representing the model for the dining cryptographers problem with N cryptographers.

Checking correctness

Checking correctness

N:	master pays:		cryptographers pay:	
	time:	iterations:	time:	iterations:
3	0.028	7	0.008	7
4	0.061	9	0.032	9
5	0.141	11	0.085	11
6	0.322	13	0.292	13
7	0.778	15	0.563	15
8	1.467	17	2.25	17
9	2.67	19	4.14	19
10	6.30	21	7.63	21
15	56.9	31	185	31
20	268	41	954	41

$$pay \;\Rightarrow\; \mathbb{P}_{=1}\,(\Diamond(done \wedge par = N\%2)) \;\wedge\; \neg pay \;\Rightarrow\; \mathbb{P}_{=1}\,(\Diamond(done \wedge par \neq N\%2)).$$

Checking correctness

N:	master pays:		cryptographers pay:	
	time:	iterations:	time:	iterations:
3	0.028	7	0.008	7
4	0.061	9	0.032	9
5	0.141	11	0.085	11
6	0.322	13	0.292	13
7	0.778	15	0.563	15
8	1.467	17	2.25	17
9	2.67	19	4.14	19
10	6.30	21	7.63	21
15	56.9	31	185	31
20	268	41	954	41

$pay \Rightarrow \mathbb{P}_{=1}(\Diamond(done \wedge par = N\%2)) \wedge \neg pay \Rightarrow \mathbb{P}_{=1}(\Diamond(done \wedge par \neq N\%2))$.
That is: if the master paid, the parity of the number of agrees matches the parity of N and, if a cryptographer paid, it does not.

Checking anonymity

Checking anonymity

N:	minimum:			maximum:		
	time:	iterations:	probability:	time:	iterations:	probability:
3	0.099	8	0.25	0.004	8	0.25
4	0.041	10	0.125	0.006	10	0.125
5	0.172	12	0.0625	0.032	12	0.0625
6	0.231	14	0.03125	0.044	14	0.03125
7	0.595	16	0.015625	0.301	16	0.015625
8	1.111	18	0.0078125	0.540	18	0.0078125
9	2.12	20	0.00390625	1.31	20	0.00390625
10	3.53	22	0.001953125	2.67	22	0.001953125
15	45.1	32	6.103515625E-5	36.8	32	6.103515625E-5

To verify anonymity – when a cryptographer pays then no cryptographer can tell who has paid – we check that any possible combination of agree and disagree has the same likelihood no matter which of the cryptographers pays.

Checking anonymity

N:	minimum:			maximum:		
	time:	iterations:	probability:	time:	iterations:	probability:
3	0.099	8	0.25	0.004	8	0.25
4	0.041	10	0.125	0.006	10	0.125
5	0.172	12	0.0625	0.032	12	0.0625
6	0.231	14	0.03125	0.044	14	0.03125
7	0.595	16	0.015625	0.301	16	0.015625
8	1.111	18	0.0078125	0.540	18	0.0078125
9	2.12	20	0.00390625	1.31	20	0.00390625
10	3.53	22	0.001953125	2.67	22	0.001953125
15	45.1	32	6.103515625E-5	36.8	32	6.103515625E-5

To verify anonymity – when a cryptographer pays then no cryptographer can tell who has paid – we check that any possible combination of agree and disagree has the same likelihood no matter which of the cryptographers pays. This needs to be checked for all 2^N possible outcomes.

Checking anonymity

N:	minimum:			maximum:		
	time:	iterations:	probability:	time:	iterations:	probability:
3	0.099	8	0.25	0.004	8	0.25
4	0.041	10	0.125	0.006	10	0.125
5	0.172	12	0.0625	0.032	12	0.0625
6	0.231	14	0.03125	0.044	14	0.03125
7	0.595	16	0.015625	0.301	16	0.015625
8	1.111	18	0.0078125	0.540	18	0.0078125
9	2.12	20	0.00390625	1.31	20	0.00390625
10	3.53	22	0.001953125	2.67	22	0.001953125
15	45.1	32	6.103515625E-5	36.8	32	6.103515625E-5

To verify anonymity – when a cryptographer pays then no cryptographer can tell who has paid – we check that any possible combination of agree and disagree has the same likelihood no matter which of the cryptographers pays. This needs to be checked for all 2^N possible outcomes. Above the results are listed for one possible outcome.

Overview

Fairness

- A policy $\mathfrak{S}$ is fair if for every state s, the probability under $\mathfrak{S}$ of all fair paths from s is one

Fairness

- A policy $\mathfrak{S}$ is fair if for every state s, the probability under $\mathfrak{S}$ of all fair paths from s is one

- A fairness assumption is realizable in MDP $\mathcal{M}$ if there is some fair policy for $\mathcal{M}$

Fairness

- A policy $\mathfrak{S}$ is fair if for every state s, the probability under $\mathfrak{S}$ of all fair paths from s is one
- A fairness assumption is realizable in MDP $\mathcal{M}$ if there is some fair policy for $\mathcal{M}$
- Realizable fairness assumptions are irrelevant for maximal reachability probabilities (i.e., safety)

Fairness

- A policy $\mathfrak{S}$ is fair if for every state s, the probability under $\mathfrak{S}$ of all fair paths from s is one
- A fairness assumption is realizable in MDP $\mathcal{M}$ if there is some fair policy for $\mathcal{M}$
- Realizable fairness assumptions are irrelevant for maximal reachability probabilities (i.e., safety)
- They are relevant for minimal reachability probabilities (i.e., liveness)

Fairness

- A policy $\mathfrak{S}$ is fair if for every state s, the probability under $\mathfrak{S}$ of all fair paths from s is one
- A fairness assumption is realizable in MDP $\mathcal{M}$ if there is some fair policy for $\mathcal{M}$
- Realizable fairness assumptions are irrelevant for maximal reachability probabilities (i.e., safety)
- They are relevant for minimal reachability probabilities (i.e., liveness)
- Computing minimal reachability probabilities under strongly fair policies is reducible to computing maximal reachability probabilities

Overview

Summary

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\min}(s \models \varphi)$.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\min}(s \models \varphi)$.
- Checking whether $s \models \mathbb{P}_{<p}(\varphi)$ amounts to determine $Pr^{\max}(s \models \varphi)$.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\mathrm{min}}(s \models \varphi)$.
- Checking whether $s \models \mathbb{P}_{<p}(\varphi)$ amounts to determine $Pr^{\mathrm{max}}(s \models \varphi)$.
- The next operator amounts to a single matrix-vector multiplication and a max/min.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\min}(s \models \varphi)$.
- Checking whether $s \models \mathbb{P}_{<p}(\varphi)$ amounts to determine $Pr^{\max}(s \models \varphi)$.
- The next operator amounts to a single matrix-vector multiplication and a max/min.
- The bounded-until operator $\mathsf{U}^{\leqslant n}$ amounts to n matrix-vector multiplications + n minimums (or maximums).

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\min}(s \models \varphi)$.
- Checking whether $s \models \mathbb{P}_{<p}(\varphi)$ amounts to determine $Pr^{\max}(s \models \varphi)$.
- The next operator amounts to a single matrix-vector multiplication and a max/min.
- The bounded-until operator $\mathsf{U}^{\leqslant n}$ amounts to n matrix-vector multiplications + n minimums (or maximums).
- The until-operator amounts to solving a linear inequation system.

Summary

- PCTL is a variant of CTL with operator $\Phi = \mathbb{P}_J(\varphi)$.
- PCTL model checking is performed by a recursive descent over Φ.
- Checking whether $s \models \mathbb{P}_{>p}(\varphi)$ amounts to determine $Pr^{\min}(s \models \varphi)$.
- Checking whether $s \models \mathbb{P}_{<p}(\varphi)$ amounts to determine $Pr^{\max}(s \models \varphi)$.
- The next operator amounts to a single matrix-vector multiplication and a max/min.
- The bounded-until operator $\mathsf{U}^{\leqslant n}$ amounts to n matrix-vector multiplications + n minimums (or maximums).
- The until-operator amounts to solving a linear inequation system.
- The worst-case time complexity is polynomial in the size of the MDP and linear in the size of the formula.