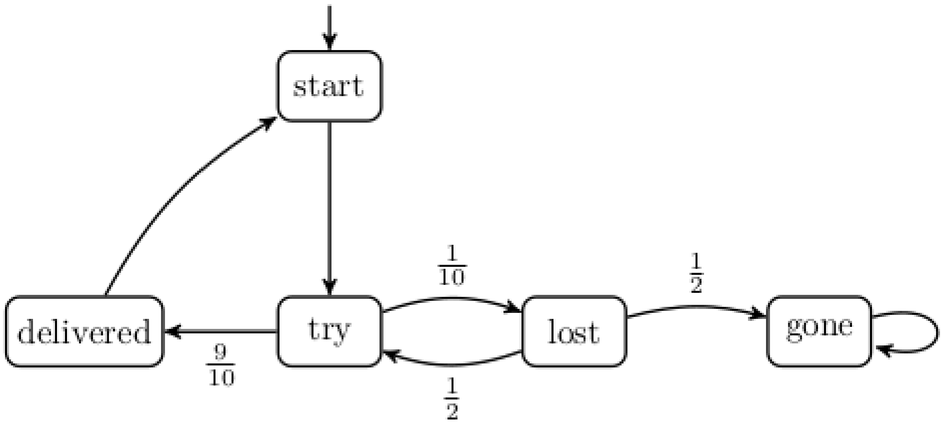
Probabilistic Model Checking

Stefan Pranger

17. 06. 2023

Communication Protocol with Faults



$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -\frac{1}{10} \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \cdot \mathbf{x} = \begin{pmatrix} 0 \\ \frac{9}{10} \\ 0 \end{pmatrix} \rightarrow \mathbf{x} = \begin{pmatrix} \frac{18}{19} \\ \frac{18}{19} \\ \frac{9}{19} \end{pmatrix}$$

Cowboy Shootout

- The three may shoot as long as anyone else is still alive. Due to differences in (re)loading times, we assume they shoot in turns. That is, The Good shoots first, then The Bad and finally The Ugly.
- The Good has a chance of a half of hitting anyone. If he hits, he does so uniformly over the living contestants.
- The Bad has a chance of 0.9 of hitting anyone. If The Ugly is alive, then he aims for him. If The Ugly already died, then he aims at The Good.
- The Ugly hits either no one or one of the living contestants and he does so with a uniform probability over these events.

```
module shootout
  cowboy: [1..3] init 1;
  good: bool init true;
  bad: bool init true;
  ugly: bool init true;
  [] cowboy=1 & good & bad & ugly   -> 1/2 :(cowboy'=2) +
                                       1/4 :(bad'=false) & (cowboy'=3) +
                                       1/4 :(ugly'=false) & (cowboy'=2);
  [] cowboy=1 & good & bad & !ugly  -> 1/2 :(cowboy'=2) +
                                       1/2 :(bad'=false) & (cowboy'=1);
  [] cowboy=1 & good & !bad & ugly  -> 1/2 :(cowboy'=3) +
                                       1/2 :(ugly'=false) & (cowboy'=1);
  [] cowboy=2 & good & bad & ugly   -> 0.1 :(cowboy'=3) +
                                       0.9 :(ugly'=false) & (cowboy'=1);
  [] cowboy=2 & good & bad & !ugly  -> 0.1 :(cowboy'=1) +
                                       0.9 :(good'=false) & (cowboy'=2);
  [] cowboy=2 & !good & bad & ugly  -> 0.1 :(cowboy'=3) +
                                       0.9 :(ugly'=false) & (cowboy'=2);
  [] cowboy=3 & good & bad & ugly  -> 1/3 :(cowboy'=1) +
                                      1/3 :(good'=false) & (cowboy'=2) +
                                      1/3 :(bad'=false) & (cowboy'=1);
  [] cowboy=3 & good & !bad & ugly -> 1/2 :(cowboy'=1) +
                                      1/2 :(good'=false) & (cowboy'=3);
  [] cowboy=3 & !good & bad & ugly -> 1/2 :(cowboy'=2) +
                                      1/2 :(bad'=false) & (cowboy'=3);
  []  good & !bad & !ugly -> true;
  [] !good &  bad & !ugly -> true;
  [] !good & !bad &  ugly -> true;
endmodule
```

Recap: Constrained Reachability

- Computing $Pr(\mathcal{M}, s_0 \models C \; \mathbf{U} \; B)$
- We have used a linear equation solver to compute the probability of satisfying the constrained reachability problem.

Probabilistic Computation Tree Logic

Probabilistic Computation Tree Logic [PCTL] is the probabilistic extension of CTL.

- Boolean state representation.
- $\forall$ and $\exists$ are replaced by $\Pr_J(\varphi)$, where $J \subseteq [0,1]$
  - The interpretation for each state $s \in S$: $Pr(\mathcal{M}, s \models \varphi) \in J$

PCTL - Syntax

Subdivision into *state* (Φ)- and *path*-formulae (φ):

$$
\begin{aligned}
\Phi ::= &\ \mathit{true} \\
\mid &\ a \\
\mid &\ \Phi_1 \wedge \Phi_2 \\
\mid &\ \neg\Phi \\
\mid &\ \Pr_J(\varphi)
\end{aligned}
\qquad\qquad
\begin{aligned}
\varphi ::= &\ \mathbf{X}\Phi \\
\mid &\ \Phi_1\ \mathbf{U}\ \Phi_2 \\
\mid &\ \Phi_1\ \mathbf{U}^{\leq n}\ \Phi_2
\end{aligned}
$$

where $a \in AP$ and $J \subseteq [0,1]$.

PCTL - Syntax

Subdivision into *state* (Φ)- and *path*-formulae (φ):

$$\begin{aligned} \Phi ::= &\ \mathit{true} \\ &\mid a \\ &\mid \Phi_1 \wedge \Phi_2 \\ &\mid \neg\Phi \\ &\mid \Pr_J(\varphi) \end{aligned}$$

$$\begin{aligned} \varphi ::= &\ \mathbf{X}\Phi \\ &\mid \Phi_1 \ \mathbf{U}\ \Phi_2 \\ &\mid \Phi_1 \ \mathbf{U}^{\leq n}\ \Phi_2 \end{aligned}$$

where $a \in AP$ and $J \subseteq [0,1]$.

- Note: path-formulae (φ) may not be nested!

PCTL - Satisfaction Relation

For a given state $s \in S$

$$
\begin{aligned}
s &\models a && \text{iff } a \in L(s), \\
s &\models \neg\varphi && \text{iff } s \not\models \varphi, \\
s &\models \varphi \wedge \psi && \text{iff } s \models \varphi \text{ and } s \models \psi, \\
s &\models \Pr_J(\varphi) && \text{iff } Pr(s \models \varphi) \in J
\end{aligned}
$$

For paths $\pi \in \mathcal{M}$:

$$
\begin{aligned}
\pi &\models \mathbf{X}\varphi && \text{iff} \quad \pi[1] \models \varphi \\
\pi &\models \varphi\,\mathbf{U}\,\psi && \text{iff} \quad \exists j \geq 0.\ (\pi[j] \models \psi \wedge (\forall 0 \leq k < j.\ \pi[k] \models \varphi) \\
\pi &\models \varphi\,\mathbf{U}^{\leq n}\psi && \text{iff} \quad \exists 0 \leq j \leq n.\ (\pi[j] \models \psi \wedge (\forall 0 \leq k < j.\ \pi[k] \models \varphi)
\end{aligned}
$$

Model Checking a PCTL Formula

- Checking the propositional part of PCTL is easy
- How to compute $Pr(\mathcal{M}, s_0 \models C\ \mathbf{U}\ B)$?
  - We solve a linear equation system. ✓
- How to compute $Pr(\mathcal{M}, s_0 \models \mathbf{X}a)$?

Model Checking a PCTL Formula

- Checking the propositional part of PCTL is easy
- How to compute $Pr(\mathcal{M}, s_0 \models C\ \mathbf{U}\ B)$?
  - We solve a linear equation system. ✓
- How to compute $Pr(\mathcal{M}, s_0 \models \mathbf{X}a)$?
  - Also easy: Simple Matrix-Vector-Multiplication! ✓

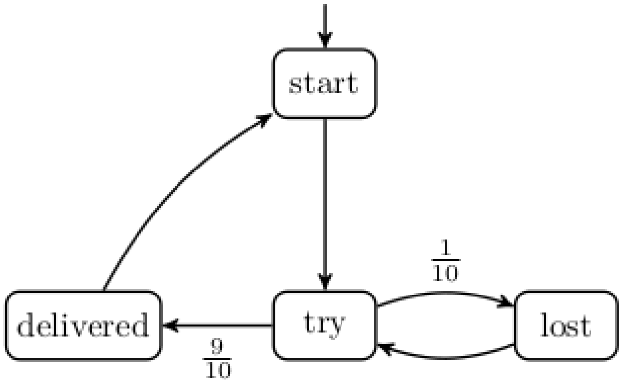
Model Checking a PCTL Formula

- Checking the propositional part of PCTL is easy
- How to compute $Pr(\mathcal{M}, s_0 \models C \ \mathbf{U} \ B)$?
  - We solve a linear equation system. ✓
- How to compute $Pr(\mathcal{M}, s_0 \models \mathbf{X}a)$?
  - Also easy: Simple Matrix-Vector-Multiplication! ✓
- How can we compute bounded reachabililty: $Pr(\mathcal{M}, s_0 \models \mathbf{F}^{<=k}a)$?

Model Checking a PCTL Formula

- Checking the propositional part of PCTL is easy
- How to compute $Pr(\mathcal{M}, s_0 \models C \mathbf{U} B)$?
  - We solve a linear equation system. ✓
- How to compute $Pr(\mathcal{M}, s_0 \models \mathbf{X} a)$?
  - Also easy: Simple Matrix-Vector-Multiplication! ✓
- How can we compute bounded reachabililty: $Pr(\mathcal{M}, s_0 \models \mathbf{F}^{<=k} a)$?
  - Again: Simple Matrix-Vector-Multiplication(s)! ✓

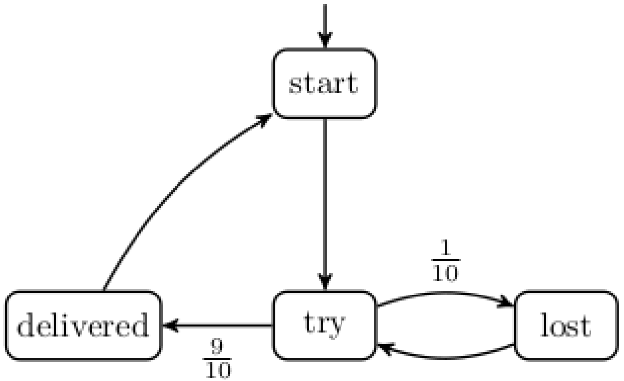
Communication Protocol



- *A message is eventually delivered or lost and our abstraction does not allow faulty values*:

```
"P>=1.0 [ F (delivered=1 | lost=1)] & P>=1.0 [G try<2 ]"
```

Communication Protocol



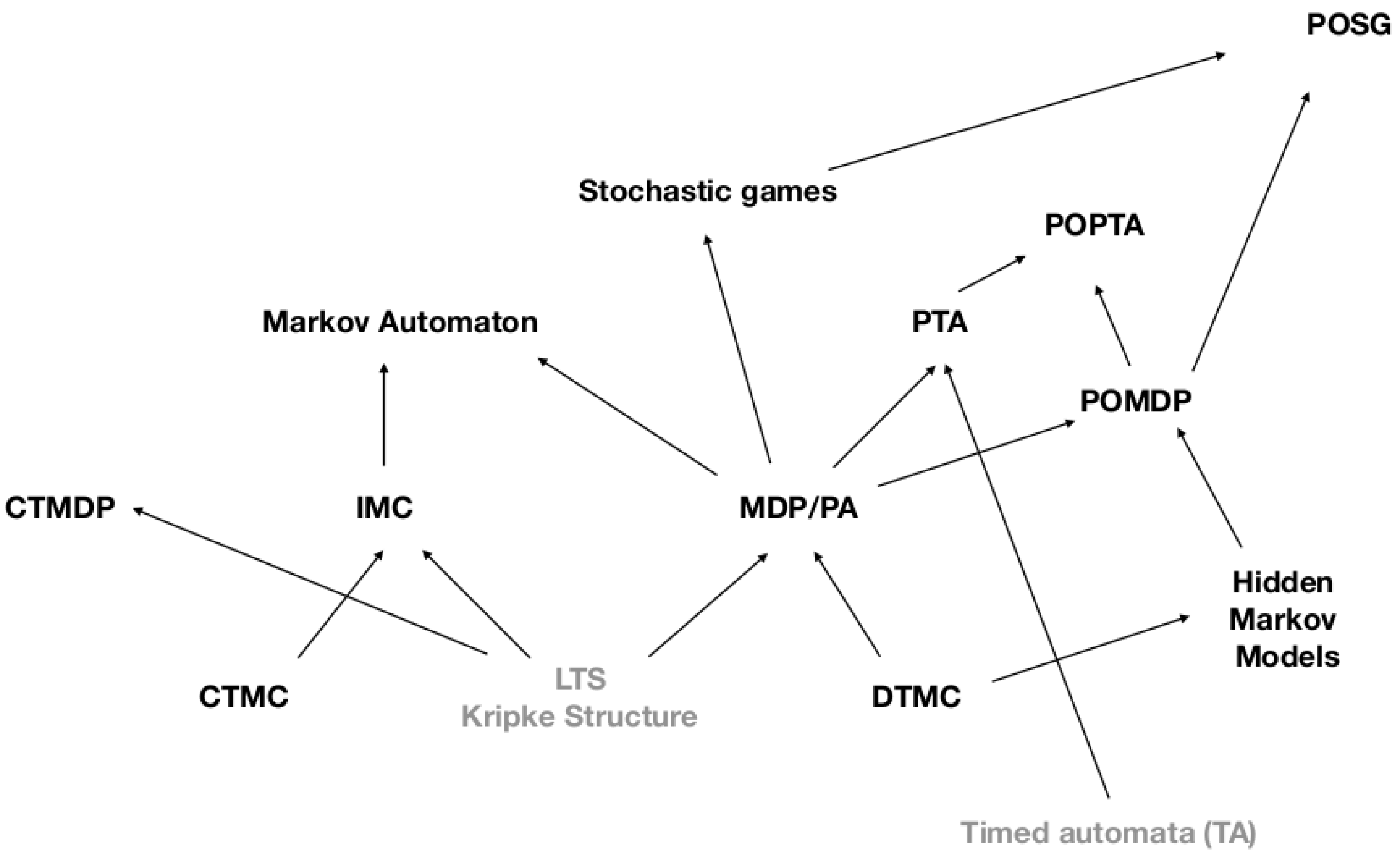
- *A message is eventually delivered or lost and our abstraction does not allow faulty values*:

```
"P>=1.0 [ F (delivered=1 | lost=1)] & P>=1.0 [G try<2 ]"
```

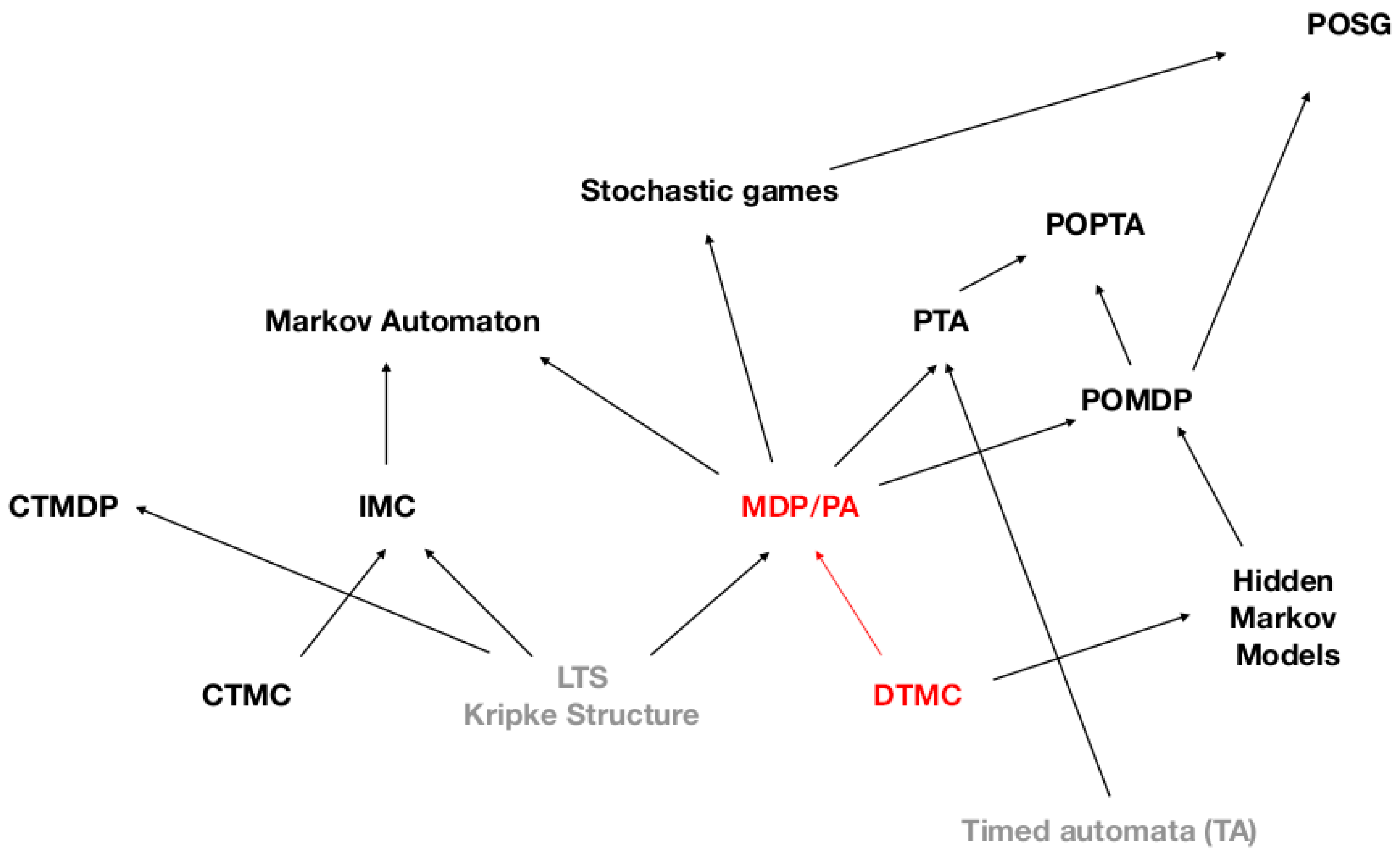
- *A message will almost surely be delivered eventually and trying to send a message implies that with a probability greater or equal 0.99 the message will be sent within three time steps.*

```
"P>=1.0 [ F (delivered=1)] & P>=1.0 [ G(!try=1| P>=0.99 [ (F<=3 delivered=1) ] ) ] "
```

Probabilistic Model Zoo



Probabilistic Model Zoo



Markov Decision Processes

Markov Decision Process $\mathcal{M} = (S, Act, \mathbb{P}, s_0, AP, L)$

- S a set of states and initial state s_0,
- Act a set of actions,
- $\mathbb{P} : S \times Act \times S \to [0, 1]$, s.t.

$$\sum_{s' \in S} \mathbb{P}(s, a, s') = 1 \; \forall (s, a) \in S \times Act$$

- AP set of atomic states and $L : S \to 2^{AP}$ a labelling function.

Markov Decision Processes

Markov Decision Process $\mathcal{M} = (S, Act, \mathbb{P}, s_0, AP, L)$

- S a set of states and initial state s_0,
- Act a set of actions,
- $\mathbb{P} : S \times Act \times S \to [0, 1]$, s.t.

$$\sum_{s' \in S} \mathbb{P}(s, a, s') = 1 \; \forall (s, a) \in S \times Act$$

- AP set of atomic states and $L : S \to 2^{AP}$ a labelling function.

The decision a defines the distribution over the next state.

Markov Decision Processes in Code and Memory

- Commands:

```
[moveNorth] x<height -> 0.9: (x'=x+1) + 0.1: true;
[moveEast]  y<width  -> 0.9: (y'=y-1) + 0.1: true;
```

Markov Decision Processes in Code and Memory

- Commands:

```
[moveNorth] x<height -> 0.9: (x'=x+1) + 0.1: true;
[moveEast]  y<width  -> 0.9: (y'=y-1) + 0.1: true;
```

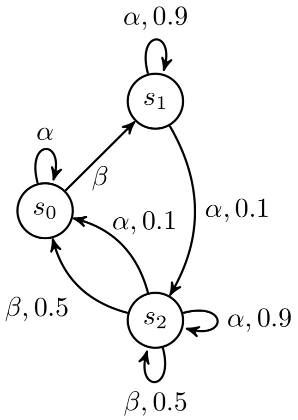
  - Guards do not need to be mutually exclusive anymore!

Markov Decision Processes in Code and Memory

- Commands:

```
[moveNorth] x<height -> 0.9: (x'=x+1) + 0.1: true;
[moveEast]  y<width  -> 0.9: (y'=y-1) + 0.1: true;
```

  - Guards do not need to be mutually exclusive anymore!



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & \frac{9}{10} & \frac{1}{10} \\ \hline \frac{9}{10} & 0 & \frac{1}{10} \\ \frac{5}{10} & 0 & \frac{5}{10} \end{bmatrix}$$

Paths in an MDP

- We extend our definition of a path for an MDP $\mathcal{M}$ as such:
- $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots \in (S \times Act)^\omega$, s.t. $\mathbb{P}(s_i, a_i, s_{i+1}) > 0, \forall i \geq 0$

Paths in an MDP

- We extend our definition of a path for an MDP $\mathcal{M}$ as such:
- $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots \in (S \times Act)^\omega$, s.t. $\mathbb{P}(s_i, a_i, s_{i+1}) > 0, \forall i \geq 0$
- Reasoning about events in an MDP resorts to the resolution of any non-determinism
  - This is done by the use of schedulers (also called strategies/policies/adversaries).

Schedulers

- A scheduler is a function that given the history of the current path returns a distribution over actions to be taken:

$$\sigma : S^* \times S \rightarrow Distr(Act)$$

- For simple properties such as reachability so called memoryless deterministic scheduler suffice:

$$\sigma : S \rightarrow Act$$

  - This means that the scheduler σ fixes an actions for each state.
  - We can then define the probability of prop under sched

$$Pr^{\sigma}(\mathcal{M}, s \models \mathbf{F} B)$$

Induced Markov Chain

Consider an MDP $\mathcal{M}$ and a memoryless deterministic scheduler:

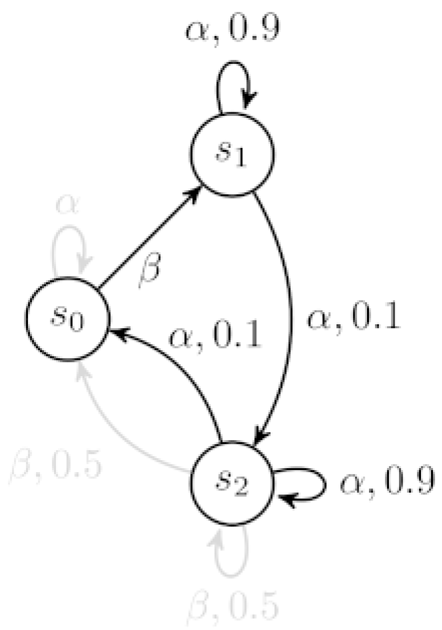
$$\sigma : S \to Act$$

Induced Markov Chain

Consider an MDP $\mathcal{M}$ and a memoryless deterministic scheduler:

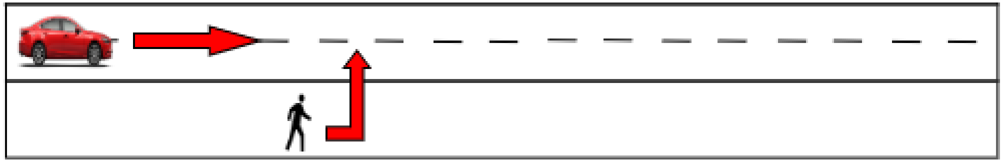
$$\sigma : S \rightarrow Act$$

$$\begin{aligned} s_0 &\mapsto \beta \\ s_1 &\mapsto \alpha \\ s_1 &\mapsto \alpha \end{aligned}$$

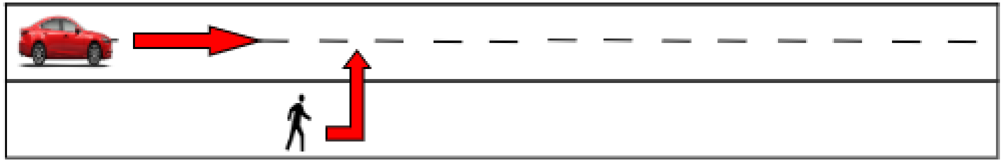


$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & \frac{9}{10} & \frac{1}{10} \\ \hline \frac{9}{10} & 0 & \frac{1}{10} \\ \frac{5}{10} & 0 & \frac{5}{10} \end{bmatrix}$$

Coding Example



Coding Example



- We introduce velocity and let the car decide whether to
  - switch lanes,
  - accelerate or
  - decelerate.

Reachability in MDPs

- We have introduced nondeterminism into probabilistic models
- Schedulers might maximize oder minimize the probability to satisfy a given property

Reachability in MDPs

- We have introduced nondeterminism into probabilistic models
- Schedulers might maximize oder minimize the probability to satisfy a given property
- We describe this with
  - $Pr^{max}(\mathcal{M}, s \models \mathbf{F}B) = sup_{\sigma} Pr^{\sigma}(\mathcal{M}, s \models \mathbf{F}B)$
  - $Pr^{min}(\mathcal{M}, s \models \mathbf{F}B) = inf_{\sigma} Pr^{\sigma}(\mathcal{M}, s \models \mathbf{F}B)$

Computing Maximum Reachability Probabilities in MDPs

We want to compute $(x_s) = Pr^{max}(\mathcal{M}, s \models \mathbf{F}B)$ using the following equation system:

- If $s \in B$: $x_s = 1$
- If $s \not\models \exists\mathbf{F}B$: $x_s = 0$
- If $s \notin B$ and $s \models \exists\mathbf{F}B$
  - $x_s = \max\{\sum_{s' \in S} \mathbb{P}(s, a, s') \cdot x_{s'} | a \in Act(s)\}$
- Such that $\sum_{x \in S} x_s$ is minimal.

Value Iteration - Method I

- Approximative method:
  - Compute the probability to reach B after n steps
  - Start with $n = 0$ and stop after some termination criterion is met

Value Iteration - Method I

- Approximative method:
  - Compute the probability to reach B after n steps
  - Start with $n = 0$ and stop after some termination criterion is met

More specifically:

$$
\begin{aligned}
x_s^{(0)} &= 1, \forall s \in B \\
x_s^{(n)} &= 0, \forall s \in S_{=0} \\
x_s^{(0)} &= 0, \qquad \forall s \in S \setminus S_{=0} \\
x_s^{(n+1)} &= \max\{\sum_{s' \in S} \mathbb{P}(s,a,s') \cdot x_{s'} | a \in Act(s)\}, \forall s \in S \setminus S_{=0}
\end{aligned}
$$

Linear Program - Method II

We can also express the problem as a linear program:

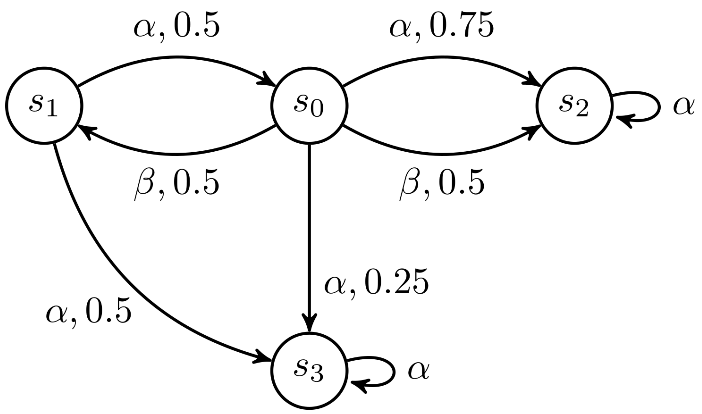
- Minimize $\sum_{x \in S} x_s$, such that:
  - $0 \leq x_s \leq 1$,
  - $x_s = 1$, if $s \in B$,
  - $x_s = 0$, if $s \not\models \exists \mathbf{F} B$,
  - $x_s \geq \sum_{s' \in S} \mathbb{P}(s, a, s') \cdot x_{s'}$, for all actions $a \in Act(s)$, if $s \notin B$ and $s \models \exists \mathbf{F} B$

Linear Program - Method II

We can also express the problem as a linear program:

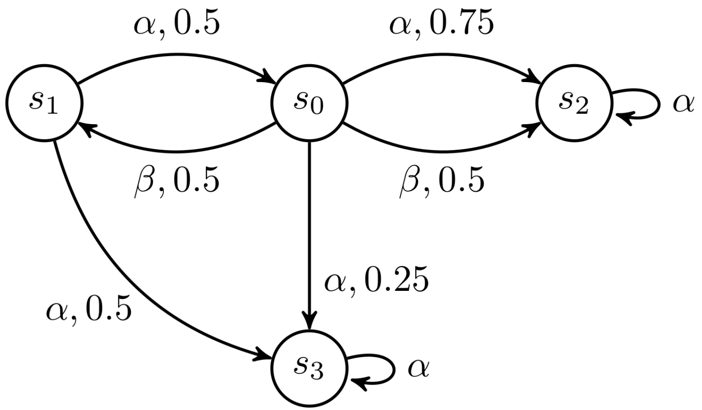
- Minimize $\sum_{x \in S} x_s$, such that:
  - $0 \leq x_s \leq 1$,
  - $x_s = 1$, if $s \in B$,
  - $x_s = 0$, if $s \not\models \exists \mathbf{F} B$,
  - $x_s \geq \sum_{s' \in S} \mathbb{P}(s, a, s') \cdot x_{s'}$, for all actions $a \in Act(s)$, if $s \notin B$ and $s \models \exists \mathbf{F} B$

Linear Program - Example



- Minimize $\sum_{x\in S} x_s$, such that:
  - $0 \leq x_s \leq 1$,
  - $x_s = 1$, if $s \in B$,
  - $x_s = 0$, if $s \not\models \exists \mathbf{F} B$,
  - $x_s \geq \sum_{s'\in S} \mathbb{P}(s,a,s') \cdot x_{s'}$, for all actions $a \in Act(s)$, if $s \notin B$ and $s \models \exists \mathbf{F} B$

Linear Program - Example




```
var x0 >= 0;
var x1 >= 0;
var x2 >= 0;
var x3 >= 0;

minimize z:     x0+x1+x2+x3;
subject to c0: x0 >= 3/4*x2 + 1/4*x3;
subject to c1: x0 >= 1/2*x1 + 1/2*x2;
subject to c2: x2 = 1;
subject to c3: x3 = 0;
subject to c4: x1 >= 1/2*x0 + 1/2*x3;

subject to c20: x0 <= 1;
subject to c21: x1 <= 1;
subject to c22: x2 <= 1;
subject to c23: x3 <= 1;

end;
```

PCTL Model Checking for MDPs

- Syntax for PCTL does not need to be changed
- The satisfaction relation for the probabilistic operator needs to be adapted:

PCTL Model Checking for MDPs

- Syntax for PCTL does not need to be changed
- The satisfaction relation for the probabilistic operator needs to be adapted:
  - We need to consider **all** schedulers:
  - $\mathcal{M}, s \models \mathrm{Pr}_{\leq p}(\varphi)$ iff $Pr^{max}(\mathcal{M}, s \models \varphi) \leq p$
  - $\mathcal{M}, s \models \mathrm{Pr}_{\geq p}(\varphi)$ iff $Pr^{min}(\mathcal{M}, s \models \varphi) \geq p$

PCTL* syntax

Subdivision into *state* (Φ)- and *path*-formulae (φ):

$$
\begin{aligned}
\Phi ::= & \ true \\
& \mid a \\
& \mid \Phi_1 \wedge \Phi_2 \\
& \mid \neg \Phi \\
& \mid \Pr_J(\varphi)
\end{aligned}
\qquad\qquad
\begin{aligned}
\varphi ::= & \ \Phi \\
& \mid \varphi_1 \wedge \varphi_2 \\
& \mid \neg \varphi \\
& \mid \mathbf{X} \varphi \\
& \mid \varphi_1 \ \mathbf{U} \ \varphi_2
\end{aligned}
$$

where $a \in AP$ and $J \subseteq [0, 1]$.

PCTL* syntax

Subdivision into *state* (Φ)- and *path*-formulae (φ):

$$\begin{aligned}\Phi ::= &\ \mathit{true} \\ &\mid a \\ &\mid \Phi_1 \wedge \Phi_2 \\ &\mid \neg\Phi \\ &\mid \Pr_J(\varphi)\end{aligned} \qquad\qquad \begin{aligned}\varphi ::= &\ \Phi \\ &\mid \varphi_1 \wedge \varphi_2 \\ &\mid \neg\varphi \\ &\mid \mathbf{X}\varphi \\ &\mid \varphi_1\ \mathbf{U}\ \varphi_2\end{aligned}$$

where $a \in AP$ and $J \subseteq [0, 1]$.

We are now allowed to interchangly use state and path formulae as subformulae.

PCTL* syntax

Subdivision into *state* (Φ)- and *path*-formulae (φ):

$$\begin{aligned}\Phi ::= & \ true \\ & \mid a \\ & \mid \Phi_1 \land \Phi_2 \\ & \mid \neg\Phi \\ & \mid \Pr_J(\varphi)\end{aligned} \qquad \begin{aligned}\varphi ::= & \ \Phi \\ & \mid \varphi_1 \land \varphi_2 \\ & \mid \neg\varphi \\ & \mid \mathbf{X}\varphi \\ & \mid \varphi_1 \ \mathbf{U} \ \varphi_2\end{aligned}$$

where $a \in AP$ and $J \subseteq [0,1]$.

We are now allowed to interchangly use state and path formulae as subformulae.

```
P=? [ GF "return_to_start" ];
P=? [ G(! (try = 1) | lost_count<4 U delivered=1 ) | delivered_count=MAX_COUNT ]
Pmax=? [ FG "hatch_closed" ]
...
```

Checking Linear Time Properties

- Last building block to model check PCTL*

Checking Linear Time Properties

- Last building block to model check PCTL*

Let $\mathcal{M}$ be a Markov Chain and φ be an LTL formula.

We are interested in:

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

Computing Probabilities for LT-Properties

- Recall that LT-properties can be expressed using automata.

Computing Probabilities for LT-Properties

- Recall that LT-properties can be expressed using automata.
- We employ an automata-based approach:
  - Convert φ into a *deterministic Rabin automata* $\mathcal{A}$.
  - Compute the Product Markov Chain $M \times \mathcal{A}$.
  - Compute the probability to satisfy φ using the product (*more on that later*).

Deterministic Rabin Automata

A *deterministic Rabin automatonisatuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, Acc)$, with

- Q a set of states and initial state q_0,
- Σ an alphabet,
- $\delta : Q \times \Sigma \to Q$ a transition function and
- $Acc \subseteq 2^Q \times 2^Q$.

An automaton $\mathcal{A}$ accepts a run $\pi = q_0 q_1 q_2 \ldots$ iff there exists a pair $(L, K) \in Acc$ s.t.:

$$(\exists n \geq 0. \forall m \geq n.\, q_m \notin L) \wedge (\exists^{\text{inf}} n \geq 0. q_n \in K)$$

Product Markov Chain

Let $\mathcal{M}$ be a Markov chain and $\mathcal{A}$ be a DFA. The product $\mathcal{M} \times \mathcal{A} = (S \times Q, \mathbb{P}', i, \{accept\}, L')$ is a Markov chain where:

- $L'(\langle s, q\rangle) = \{accept\}$ if $q \in F$,
- $i = \langle s_0, q_1\rangle$ is the initial state with $q_1 = \delta(q_0, L(s))$ and
- $\mathbb{P}'(\langle s, q\rangle, \langle s', q'\rangle) = \mathbb{P}(s, s')$ if $q' = \delta(q, L(s'))$ and 0 otherwise.

Product Markov Chain

Let $\mathcal{M}$ be a Markov chain and $\mathcal{A}$ be a DFA. The product $\mathcal{M} \times \mathcal{A} = (S \times Q, \mathbb{P}', i, \{accept\}, L')$ is a Markov chain where:

- $L'(\langle s, q \rangle) = \{accept\}$ if $q \in F$,
- $i = \langle s_0, q_1 \rangle$ is the initial state with $q_1 = \delta(q_0, L(s))$ and
- $\mathbb{P}'(\langle s, q \rangle, \langle s', q' \rangle) = \mathbb{P}(s, s')$ if $q' = \delta(q, L(s'))$ and 0 otherwise.

  Post-Lecture-Note: This is the definition of a product with a DFA, the product with a DRA can be done in a similar way.

Product Markov Chain

Let $\mathcal{M}$ be a Markov chain and $\mathcal{A}$ be a DFA. The product $\mathcal{M} \times \mathcal{A} = (S \times Q, \mathbb{P}', i, \{accept\}, L')$ is a Markov chain where:

- $L'(\langle s, q \rangle) = \{accept\}$ if $q \in F$,
- $i = \langle s_0, q_1 \rangle$ is the initial state with $q_1 = \delta(q_0, L(s))$ and
- $\mathbb{P}'(\langle s, q \rangle, \langle s', q' \rangle) = \mathbb{P}(s, s')$ if $q' = \delta(q, L(s'))$ and 0 otherwise.

  Post-Lecture-Note: This is the definition of a product with a DFA, the product with a DRA can be done in a similar way.

Since $\mathcal{A}$ is deterministic it can be interpreted as a witness for its current state on the product trace:

$$\pi^+ = \langle s_0, q_1 \rangle, \langle s_1, q_2 \rangle, \langle s_2, q_3 \rangle, \ldots$$

Computing the Probability to Satisfy φ

- We want to use the product $\mathcal{M} \times \mathcal{A}$ and know
- $\mathcal{A}$'s acceptance condition:

$$(\exists n \geq 0. \forall m \geq n.\, q_m \notin L_i) \land (\exists^{\text{inf}} n \geq 0. q_n \in K_i)$$

- for a pair $L_i, K_i \in Acc$.

Computing the Probability to Satisfy φ

- We want to use the product $\mathcal{M} \times \mathcal{A}$ and know
- $\mathcal{A}$'s acceptance condition:

$$(\exists n \geq 0.\forall m \geq n.\, q_m \notin L_i) \wedge (\exists^{\text{inf}} n \geq 0.q_n \in K_i)$$

- for a pair $L_i, K_i \in Acc$.
- $\Rightarrow$ we need to compute the probability to see infinitely many labels from K_i and only finitely many labels from L_i for some i.

Bottom Strongly Connected Components

- Consider the underlying directed graph $G = (V, E)$ for a given Markov chain $\mathcal{M}$ and a component $C \in V$.
- C is *strongly connected* if $\forall s, t \in C$:
  - s is reachable from t and
  - t is reachable from s.

Bottom Strongly Connected Components

- Consider the underlying directed graph $G = (V, E)$ for a given Markov chain $\mathcal{M}$ and a component $C \in V$.
- C is *strongly connected* if $\forall s, t \in C$:
  - s is reachable from t and
  - t is reachable from s.
- C is *bottom* strongly connected if no state outside of C is reachable from C.

Bottom Strongly Connected Components

- Consider the underlying directed graph $G = (V, E)$ for a given Markov chain $\mathcal{M}$ and a component $C \in V$.
- C is *strongly connected* if $\forall s, t \in C$:
  - s is reachable from t and
  - t is reachable from s.
- C is *bottom* strongly connected if no state outside of C is reachable from C.
- For Markov chains we have that a bottom strongly connected component
  - cannot be left and
  - all states will be visited infinitely often with a probability of one.

Computing the Probability to Satisfy φ

According to the acceptance condition $Acc = \{(L_0, K_0), \ldots (L_m, K_m)\}$ of $\mathcal{A}$:

- Identify BSCCs C_j such that:
  - For some $i \in [0, m]$:

$$C_j \cap (S \times L_i) = \emptyset \text{ and } C_j \cap (S \times K_i) \neq \emptyset$$

- Let $U = \bigcup_{j,\, C_j\ accepting} C_j$

Computing the Probability to Satisfy φ

According to the acceptance condition $Acc = \{(L_0, K_0), \dots (L_m, K_m)\}$ of $\mathcal{A}$:

- Identify BSCCs C_j such that:
  - For some $i \in [0, m]$:

$$C_j \cap (S \times L_i) = \emptyset \text{ and } C_j \cap (S \times K_i) \neq \emptyset$$

- Let $U = \bigcup_{j,\, C_j\ accepting} C_j$
- We then have the following:

$$\Pr(\mathcal{M}, s \models \varphi) = \Pr(\mathcal{M} \times \mathcal{A}, \langle s, q_i \rangle \models \mathbf{F} U)$$