Quantitative Verification
Chapter 4: Markov decision processes

Jan Křetínský

Technical University of Munich

Winter 2020/21
Discrete-time
Markov Decision Processes
MDP
DTMC – purely probabilistic
Possible successor states are chosen based on probabilities but not on decisions.

We want decisions to model both

- controllable setting (game theory, operations theory, control theory);
- uncontrollable setting (interleaving in concurrent systems, abstractions of models, open systems)

How to introduce decisions, i.e., non-determinism, to DTMC?
Definition:
A (labelled) Markov Decision Process (MDP) is a tuple

\[ M = (S, \text{Act}, P, \pi_0, L) \]

where

- \( S \) is a countable set of states,
- \( \text{Act} \) is a finite set of actions,
- \( P : S \times \text{Act} \times S \to [0, 1] \) is the transition probability function, such that for each state \( s \) and action \( \alpha \),
  \[ \sum_{s' \in S} P(s, \alpha, s') = 1, \] then we say that \( \alpha \) is enabled in \( s \); or
  \[ P(s, \alpha, s') = 0 \] for all \( s' \), then we say that \( \alpha \) is not enabled in \( s \).
- \( \pi_0 \) is the initial distribution, and
- \( L : S \to 2^{\text{Act}} \) is the labeling function.

The set of actions enabled in \( s \) is denoted by \( \text{Act}(s) \). We assume that for each \( s \), we have \( \text{Act}(s) \neq \emptyset \).
Example:

Problem:
How is the non-determinism resolved?
MDP – Schedulers

Example:

Problem:
How is the non-determinism resolved?
Allowing memory and randomness:

Definition (Scheduler):
A scheduler (also called strategy or policy) on an MDP $\mathcal{M} = (S, Act, P, \pi_0, L)$ is a function $\Theta$ assigning to each history $s_0 \cdots s_n \in S^+$ a probability distribution over $Act$ such that $\alpha$ is enabled in $s_n$ whenever $\Theta(s_0 \cdots s_n)(\alpha) > 0$. 
Definition (Induced DTMC):

Let $\mathcal{M} = (S, Act, P, \pi_0, L)$ be a MDP and scheduler $\Theta$ on $\mathcal{M}$. The induced DTMC is given by

$$\mathcal{M}^\Theta = (S^+, P^\Theta, \pi_0, L'),$$

where for any $h = s_0s_1 \ldots s_n$, we define

$$P^\Theta(h, hs_{n+1}) = \sum_{\alpha \in Act} \Theta(h)(\alpha) \cdot P(s_n, \alpha, s_{n+1})$$

and $L'(h) = L(s_n)$. 
Example:

We choose a scheduler $\Theta$ that always takes action $\beta$ in state $s$ and action $\gamma$ in state $u$. The induced DTMC $M^\Theta$ for the previous example:

Notation

- $P^\Theta$ – the probability measure of $M^\Theta$
- There is a bijection $\xi$ mapping each sequence of states $s_0s_1s_2\cdots$ to a sequence of histories $s_0\ s_0s_1\ s_0s_1s_2\cdots$ (a path of $M^\Theta$).
- When using previous notation for sets of paths such as $\Diamond B$, we actually mean $\xi(\Diamond B)$
Classes of schedulers:

- A scheduler $\Theta$ is memoryless if for histories $s_0s_1\ldots s_n \in S^+$ and $s'_0s'_1\ldots s_n \in S^+$ with $s_n = s'_n$ it holds

\[ \Theta(s_0s_1\ldots s_n) = \Theta(s'_0s'_1\ldots s'_n). \]

- A scheduler $\Theta$ is deterministic if for all histories $s_0s_1\ldots s_n \in S^+$ it holds $\Theta(s_0s_1\ldots s_n)(\alpha) = 1$ for some action $\alpha$.

A memoryless deterministic (MD) $\Theta$ can be viewed as a function $\Theta : S \to \text{Act}$.

Example:
The scheduler of the previous example was memoryless and deterministic since the decision what action to take was fixed.

Note:
A scheduler has finite memory if representable by a finite automaton.
For MC:

- **Reachability**: \( x = Ax + b \)  
  (with \( (x(s))_{s \in S} \))

- **Probabilistic logics**: combination of the techniques

- **Transient analysis**: \( \pi_n = \pi_0 P^n \)

- **Steady-state analysis**: \( \pi P = \pi, \pi \tilde{1} = 1 \)  
  (ergodic)

- **Rewards**: reduction to steady-state analysis
Analysis questions

For MC:
- **Reachability:** \( x = Ax + b \) (with \((x(s))_{s \in S}\))
- **Probabilistic logics:** combination of the techniques
- **Transient analysis:** \( \pi_n = \pi_0 P^n \)
- **Steady-state analysis:** \( \pi P = \pi, \pi \vec{1} = 1 \) (ergodic)
- **Rewards:** reduction to steady-state analysis

For MDP:
- **Quantities not defined per se, but depend on the scheduler**
- **We can naturally consider the best case and the worst case among all schedulers**
  (recall that non-determinism can model controllable or uncontrollable choice)
MDP – Reachability
Min
When playing “Mensch Ärgere dich nicht” against a fixed opponent strategy, what is the minimal probability of having all pieces kicked out into the outside area again?

Max
What is the maximal probability of winning the game?
MDP - Reachability

Min
- Best case for reaching undesirable states when controlled
- Worst case for reaching desirable states when not controlled

The minimum probability to reach a set of states $B$ from a state $s$ (within $n$ steps) is

$$\inf \Theta P_s^{\Theta}(\Diamond B), \quad \inf \Theta P_s^{\Theta}(\Diamond \leq n B)$$

Max
- Best case for reaching desirable states when controlled
- Worst case for reaching undesirable states when not controlled

The maximum probability to reach a set of states $B$ from a state $s$ (within $n$ steps) is

$$\sup \Theta P_s^{\Theta}(\Diamond B), \quad \sup \Theta P_s^{\Theta}(\Diamond \leq n B)$$

Focus on maximum; minimum is similar
Recall for DTMC

Let \((S, P, \pi_0)\) be a finite DTMC and \(B \subseteq S\). The vector \(x\) with \(x(s) = P_s(\Diamond B)\) is the unique solution of the equation system

\[
x(s) = \begin{cases} 
1 & \text{if } s \in B, \\
0 & \text{if } s \in S_0 = \{s \mid P_s(\Diamond B) = 0\}, \\
\sum_{u \in S} P(s, u) \cdot x(u) & \text{otherwise}.
\end{cases}
\]
**MDP - Reachability**

**Recall for DTMC**

Let \((S, P, \pi_0)\) be a finite DTMC and \(B \subseteq S\). The vector \(x\) with 
\[ x(s) = P_s(\Diamond B) \]

is the unique solution of the equation system 
\[
    x(s) = \begin{cases} 
    1 & \text{if } s \in B, \\
    0 & \text{if } s \in S_0 = \{s \mid P_s(\Diamond B) = 0\}, \\
    \sum_{u \in S} P(s, u) \cdot x(u) & \text{otherwise.}
    \end{cases}
\]

**Theorem (Maximum Reachability Probability):**

Let \((S, \text{Act}, P, \pi_0, L)\) be a finite MDP and \(B \subseteq S\). The vector \(x\) with 
\[ x(s) = \sup_{\Theta} P_s^{\Theta}(\diamond B) \]

is the least solution of the equation system 
\[
    x(s) = \begin{cases} 
    1 & \text{if } s \in B, \\
    0 & \text{if } s \in S_0^{\max} = \{s \mid \sup_{\Theta} P_s^{\Theta}(\diamond B) = 0\}, \\
    \max_{\alpha \in \text{Act}(s)} \sum_{u \in S} P(s, \alpha, u) \cdot x(u) & \text{otherwise.}
    \end{cases}
\]
Theorem (Optimal Memoryless Scheduler):
Let \( \mathcal{M} \) be a finite MDP with state space \( S \), and \( B \subseteq S \). There exist memoryless deterministic schedulers \( \Theta_{\text{min}} \), \( \Theta_{\text{max}} \) such that for any \( s \in S \) it holds

\[
P_{s}^{\Theta_{\text{min}}} (\Diamond B) = \inf_{\Theta} P_{s}(\Diamond B), \quad P_{s}^{\Theta_{\text{max}}} (\Diamond B) = \sup_{\Theta} P_{s}(\Diamond B)
\]

Proof Sketch

▶ For \( \Theta_{\text{min}} \) it suffices to fix in each \( s \) an arbitrary action \( \alpha \) that minimizes \( \sum_{u \in S} P(s, \alpha, u) \cdot x_{u} \).

▶ Does not work for \( \Theta_{\text{max}} \)!

▶ For \( \Theta_{\text{max}} \) we fix in each \( s \) among the actions that maximize \( \sum_{u \in S} P(s, \alpha, u) \cdot x_{u} \) an arbitrary action \( \alpha \) that minimizes the number of steps needed to reach \( B \) with positive probability.

How can we compute the vectors of values?

▶ Linear programming

▶ Value iteration
Theorem (Optimal Memoryless Scheduler):

Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$. There exist memoryless deterministic schedulers $\Theta_{\text{min}}, \Theta_{\text{max}}$ such that for any $s \in S$ it holds

$$P_s^{\Theta_{\text{min}}} (\Diamond B) = \inf_{\Theta} P_s^{\Theta} (\Diamond B), \quad P_s^{\Theta_{\text{max}}} (\Diamond B) = \sup_{\Theta} P_s^{\Theta} (\Diamond B)$$

Proof Sketch

- For $\Theta_{\text{min}}$ it suffices to fix in each $s$ an arbitrary action $\alpha$ that minimizes $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$. 

Linear programming

Value iteration
Theorem (Optimal Memoryless Scheduler):
Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$. There exist memoryless deterministic schedulers $\Theta_{\text{min}}, \Theta_{\text{max}}$ such that for any $s \in S$ it holds

$$P_s^{\Theta_{\text{min}}} (\Diamond B) = \inf_{\Theta} P_s^\Theta (\Diamond B), \quad P_s^{\Theta_{\text{max}}} (\Diamond B) = \sup_{\Theta} P_s^\Theta (\Diamond B)$$

Proof Sketch

- For $\Theta_{\text{min}}$ it suffices to fix in each $s$ an arbitrary action $\alpha$ that minimizes $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$.
- Does not work for $\Theta_{\text{max}}$!
Theorem (Optimal Memoryless Scheduler):
Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$. There exist memoryless deterministic schedulers $\Theta^{\text{min}}, \Theta^{\text{max}}$ such that for any $s \in S$ it holds

$$P_s^{\Theta^{\text{min}}}(\Diamond B) = \inf_{\Theta} P_s^{\Theta}(\Diamond B), \quad P_s^{\Theta^{\text{max}}}(\Diamond B) = \sup_{\Theta} P_s^{\Theta}(\Diamond B)$$

Proof Sketch

- For $\Theta^{\text{min}}$ it suffices to fix in each $s$ an arbitrary action $\alpha$ that minimizes $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$.
- Does not work for $\Theta^{\text{max}}$!
- For $\Theta^{\text{max}}$ we fix in each $s$ among the actions that maximize $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$ an arbitrary action $\alpha$ that minimizes the number of steps needed to reach $B$ with positive probability.
Theorem (Optimal Memoryless Scheduler):
Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$. There exist memoryless deterministic schedulers $\Theta_{\text{min}}, \Theta_{\text{max}}$ such that for any $s \in S$ it holds

$$P_s^{\Theta_{\text{min}}}(\diamond B) = \inf_{\Theta} P_s^{\Theta}(\diamond B), \quad P_s^{\Theta_{\text{max}}}(\diamond B) = \sup_{\Theta} P_s^{\Theta}(\diamond B)$$

Proof Sketch

- For $\Theta_{\text{min}}$ it suffices to fix in each $s$ an arbitrary action $\alpha$ that minimizes $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$.
- Does not work for $\Theta_{\text{max}}$!
- For $\Theta_{\text{max}}$ we fix in each $s$ among the actions that maximize $\sum_{u \in S} P(s, \alpha, u) \cdot x_u$ an arbitrary action $\alpha$ that minimizes the number of steps needed to reach $B$ with positive probability.

How can we compute the vectors of values?

- linear programming
- value iteration
Let \((S, \text{Act}, P, \pi_0, L)\) be a finite MDP and \(B \subseteq S\). The vector \(x\) with \(x(s) = \sup \Theta P \Theta s(\Box B)\) is the unique solution of the linear program

\[
\begin{align*}
\text{minimize} & \quad \sum_{s \in S} x(s) \\
\text{satisfying} & \quad x(s) = 1 \quad \forall s \in B, \\
& \quad x(s) = 0 \quad \forall s \in S \setminus (B \cup S), \\
& \quad \forall \alpha \in \text{Act}.
\end{align*}
\]
Linear Program:

Let \((S, \text{Act}, P, \pi_0, L)\) be a finite MDP and \(B \subseteq S\). The vector \(x\) with \(x(s) = \sup_\Theta P_s^\Theta(\diamond B)\) is the unique solution of the linear program

\[
\begin{align*}
\text{satisfying} & \quad x(s) = 1 \quad \forall s \in B, \\
x(s) & = 0 \quad \forall s \in S_{0}^{\text{max}}, \\
x(s) & \geq \sum_{u \in S} P(s, \alpha, u) \cdot x(u) \quad \forall s \in S \setminus (B \cup S_{0}^{\text{max}}), \forall \alpha \in \text{Act}. 
\end{align*}
\]
Linear Program:

Let \((S, \text{Act}, P, \pi_0, L)\) be a finite MDP and \(B \subseteq S\). The vector \(x\) with \(x(s) = \sup_{\Theta} P_s^\Theta(\diamondsuit B)\) is the unique solution of the linear program

\[
\begin{align*}
\text{minimize} & \quad \sum_{s \in S} x(s) \\
\text{satisfying} & \quad x(s) = 1 \quad \forall s \in B, \\
& \quad x(s) = 0 \quad \forall s \in S_0^{\text{max}}, \\
& \quad x(s) \geq \sum_{u \in S} P(s, \alpha, u) \cdot x(u) \quad \forall s \in S \setminus (B \cup S_0^{\text{max}}), \forall \alpha \in \text{Act}.
\end{align*}
\]
Value Iteration Algorithm:
Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$.
- Initialize $x_0(s)$ to 1 if $s \in B$ and to 0, otherwise.
- Iterate

$$x_{n+1}(s) = \begin{cases} 
1 & \text{if } s \in B, \\
0 & \text{if } s \in S_0^{\text{max}}, \\
\max_{\alpha \in \text{Act}(s)} \sum_{u \in S} P(s, \alpha, u) \cdot x_n(u) & \text{otherwise}
\end{cases}$$

until convergence, i.e., until $\max_{s \in S} |x_{n+1}(s) - x_n(s)| < \epsilon$
for a small $\epsilon > 0$
Value Iteration Algorithm:
Let $\mathcal{M}$ be a finite MDP with state space $S$, and $B \subseteq S$.
- Initialize $x_0(s)$ to 1 if $s \in B$ and to 0, otherwise.
- Iterate

\[
x_{n+1}(s) = \begin{cases} 
1 & \text{if } s \in B, \\
0 & \text{if } s \in S_0^{\text{max}}, \\
\max_{\alpha \in \text{Act}(s)} \sum_{u \in S} P(s, \alpha, u) \cdot x_n(u) & \text{otherwise}
\end{cases}
\]

until convergence, i.e., until $\max_{s \in S} |x_{n+1}(s) - x_n(s)| < \epsilon$

for a small $\epsilon > 0$

Theorem
- $x_n(s) = \sup_{\Theta} P_s^\Theta(\Diamond \preceq^n B)$.
- $\lim_{n \to \infty} x_n(s) = \sup_{\Theta} P_s^\Theta(\Diamond B)$. 


Is a memoryless deterministic scheduler enough for optimizing $\Diamond \leq^n B$?
Is a memoryless deterministic scheduler enough for optimizing $\diamondsuit \leq n B$?

No! For step-bounded reachability we might need finite memory.
(Intuition: Depending on the current step, different paths of different length might be optimal).
We rather compute the set

\[ S_{>0}^{\text{max}} = \{ s \mid \sup_{\Theta} P_s^{\Theta}(\diamond B) > 0 \} \]

and return

\[ S_0^{\text{max}} = S \setminus S_{>0}^{\text{max}} \]
We rather compute the set

\[ S_{>0}^{\text{max}} = \{ s \mid \sup_{\Theta} P_s^{\Theta}(\Diamond B) > 0 \} \]

and return

\[ S_0^{\text{max}} = S \setminus S_{>0}^{\text{max}} \]

**$S_{>0}^{\text{max}}$:**

Initialize the set to $B$ and in every iteration add states that reach the set in one step with positive probability for some enabled action. Repeat until fix-point is reached.
We rather compute the set

\[ S_{>0}^{\text{max}} = \{ s \mid \sup_{\Theta} P_s^{\Theta}(\diamond B) > 0 \} \]

and return

\[ S_{0}^{\text{max}} = S \setminus S_{>0}^{\text{max}} \]

\( S_{>0}^{\text{max}} \)  
Initialize the set to \( B \) and in every iteration add states that reach the set in one step with positive probability for some enabled action. Repeat until fix-point is reached.

(Similarly for \( S_{>0}^{\text{min}} \):)
We rather compute the set

\[ S_{max}^0 = \{ s \mid \sup_\Theta P_s^\Theta (\Diamond B) > 0 \} \]

and return

\[ S_{0}^{max} = S \setminus S_{max}^{0} \]

\[ S_{0}^{max} \]:

Initialize the set to \( B \) and in every iteration add states that reach the set in one step with positive probability for some enabled action. Repeat until fix-point is reached.

(Similarly for \( S_{0}^{min} \): replace “some” by “every”)
Analysis questions

- Reachability: LP or VI
- Probabilistic logics: combination of the techniques (in particular reachability and bounded reachability)
- Transient analysis
- Steady-state analysis
- Rewards
MDP – PCTL & LTL
We consider two different sources of non-determinism:

**Controllable** If we can control the choice of actions:
Is there possibly a scheduler guaranteeing the specified desirable behavior?

**Uncontrollable** If we cannot control the choice of actions:
Do all schedulers necessarily guarantee the specified desirable behavior?

**Note:** If we have undesirable behaviour specified, we can apply negation to obtain the desirable behaviour.
pLTL
Example: the probability that eventually red player is kicked out and then immediately kicks out blue player is possibly / necessarily $\geq 0.8$

$$\exists \Theta / \forall \Theta : P^{\Theta}(\mathcal{F} (r\text{kicked} \land X b\text{kicked})) \geq 0.8$$

PCTL
Example: with prob. necessarily $\geq 0.5$ the probability to return to initial state is always necessarily $\geq 0.1$: $P_{\geq 0.5} G P_{\geq 0.1} F \text{ init}$
Recall: DTMC
For a state $s$:

- $s \models true$ (always),
- $s \models a$ iff $a \in L(s)$,
- $s \models \phi_1 \land \phi_2$ iff $s \models \phi_1$ and $s \models \phi_2$,
- $s \models \neg \phi$ iff $s \not\models \phi$,
- $s \models \mathcal{P}_J(\psi)$ iff $P_s(Paths(\psi)) \in J$.

MDP
Stays the same except for $\mathcal{P}_J$ defined in one of the following ways:

- Possibility (controllable): $s \models \mathcal{P}_J(\psi)$ iff $\exists \Theta : P_s^\Theta(Paths(\psi)) \in J$;
- Necessity (uncontrollable): $s \models \mathcal{P}_J(\psi)$ iff $\forall \Theta : P_s^\Theta(Paths(\psi)) \in J$.

Note
PCTL path formulae semantics stays the same.
Algorithm

Input: MDP $\mathcal{M}$, state $s$, PCTL state formula $\Phi$
Output: TRUE iff $s \models \Phi$.

The algorithm is conceptually the same as for DTMC:
Again, consider the bottom-up traversal of the parse tree of $\Phi$:
- The leaves are $a \in AP$ or $true$ and
- the inner nodes are:
  - unary – labelled with the operator $\neg$ or $\mathcal{P}_j(\mathcal{X})$;
  - binary – labelled with an operator $\land$, $\mathcal{P}_j(\mathcal{U})$, or $\mathcal{P}_j(\mathcal{U} \leq n)$.

Example: $\neg a \land \mathcal{P}_{\leq 0.2}(\neg b \land \mathcal{P}_{\geq 0.9}(\Box c))$

Compute $Sat(\Psi) = \{s \in S \mid s \models \Psi\}$ for each node $\Psi$ of the tree in a bottom-up fashion. Then $s \models \Phi$ iff $s \in Sat(\Phi)$. 
PCTL Verification (2) – Algorithm

As before:

- \( \text{Sat}(\text{true}) = S \),
- \( \text{Sat}(a) = \{ s \mid a \in L(s) \} \)
- \( \text{Sat}(\Phi_1 \land \Phi_2) = \text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2) \)
- \( \text{Sat}(\neg \Phi) = S \setminus \text{Sat}(\Phi) \)

Path operator for “possibly”

We need to restrict to path operators of the form \( P_{\times p} \) with \( p \in [0, 1] \) and \( \times \in \{ \leq, <, >, \geq \} \). We have

- for \( \times \in \{ \leq, < \} \):
  \[
  \text{Sat}(P_{\times p}(\Psi)) = \{ s \in S \mid \min_\Theta P_\Theta^s(\text{Paths}(\Psi)) \times p \}
  \]
- for \( \times \in \{ \geq, > \} \):
  \[
  \text{Sat}(P_{\times p}(\Psi)) = \{ s \in S \mid \max_\Theta P_\Theta^s(\text{Paths}(\Psi)) \times p \}
  \]
As before:

- \( Sat(true) = S \)
- \( Sat(a) = \{ s \mid a \in L(s) \} \)
- \( Sat(\Phi_1 \land \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2) \)
- \( Sat(\neg \Phi) = S \setminus Sat(\Phi) \)

Path operator for “possibly”
We need to restrict to path operators of the form \( \mathcal{P}_{\times p} \) with \( p \in [0, 1] \) and \( \times \in \{\leq, <, >, \geq\} \). We have

- for \( \times \in \{\leq, <\} \):
  \[ Sat(\mathcal{P}_{\times p}(\Psi)) = \{ s \in S \mid \min_{\Theta} P_s^{\Theta}(Paths(\Psi)) \times p \} \]

- for \( \times \in \{\geq, >\} \):
  \[ Sat(\mathcal{P}_{\times p}(\Psi)) = \{ s \in S \mid \max_{\Theta} P_s^{\Theta}(Paths(\Psi)) \times p \} \]

“Necessarily”
can be done similarly by swapping \( \max \) and \( \min \).
PCTL Verification – Algorithm (3)

Similar as before:

- **Next:**
  \[
  \max_{\Theta} P_s^\Theta \left( \text{Paths}(X \Phi) \right) =
  \]

- **Bounded Until:**
  \[
  \max_{\Theta} P_s^\Theta \left( \text{Paths}(\Phi_1 \cup_1 \leq^n \Phi_2) \right) =
  \]

- **Unbounded Until:**
  \[
  \max_{\Theta} P_s \left( \text{Paths}(\Phi_1 \cup \Phi_2) \right) =
  \]
PCTL Verification – Algorithm (3)

Similar as before:

- **Next:**
  \[ \max_{\Theta} P_{s}^{\Theta}(Paths(\forall \Phi)) = \max_{\alpha \in \text{Act}(s)} \sum_{s' \in \text{Sat}(\Phi)} P(s, s') \]

- **Bounded Until:**
  \[ \max_{\Theta} P_{s}^{\Theta}(Paths(\Phi_{1} U \leq_{n} \Phi_{2})) = \max_{\Theta} P_{s}^{\Theta}(\text{Sat}(\Phi_{1}) U \leq_{n} \text{Sat}(\Phi_{2})) \]

- **Unbounded Until:**
  \[ \max_{\Theta} P_{s}(Paths(\Phi_{1} U \Phi_{2})) = \max_{\Theta} P_{s}(\text{Sat}(\Phi_{1}) U \text{Sat}(\Phi_{2})) \]
Similar as before:

▶ **Next:**

\[
\max_{\Theta} P_{s}^{\Theta}\left(\text{Paths}(\mathcal{X} \Phi)\right) = \max_{\alpha \in \text{Act}(s)} \sum_{s' \in \text{Sat}(\Phi)} P(s, s')
\]

▶ **Bounded Until:**

\[
\max_{\Theta} P_{s}^{\Theta}\left(\text{Paths}(\Phi_1 \cup \leq^n \Phi_2)\right) = \max_{\Theta} P_{s}^{\Theta}\left(\text{Sat}(\Phi_1) \cup \leq^n \text{Sat}(\Phi_2)\right)
\]

▶ **Unbounded Until:**

\[
\max_{\Theta} P_{s}\left(\text{Paths}(\Phi_1 \cup \Phi_2)\right) = \max_{\Theta} P_{s}\left(\text{Sat}(\Phi_1) \cup \text{Sat}(\Phi_2)\right)
\]

▶ **similarly for min**
PCTL Verification – Algorithm (3)

Similar as before:

- **Next:**
  \[
  \max_{\Theta} P_{s}^{\Theta}(Paths(\mathcal{X} \Phi)) = \max_{\alpha \in \text{Act}(s)} \sum_{s' \in \text{Sat}(\Phi)} P(s, s')
  \]

- **Bounded Until:**
  \[
  \max_{\Theta} P_{s}^{\Theta}(Paths(\Phi_1 \cup \leq^{n} \Phi_2)) = \max_{\Theta} P_{s}^{\Theta}(\text{Sat}(\Phi_1) \cup \leq^{n} \text{Sat}(\Phi_2))
  \]

- **Unbounded Until:**
  \[
  \max_{\Theta} P_{s}(Paths(\Phi_1 \cup \Phi_2)) = \max_{\Theta} P_{s}(\text{Sat}(\Phi_1) \cup \text{Sat}(\Phi_2))
  \]

- **similarly for** \(\min_{\Theta}\)

**As before:**
can be reduced to step-bounded/unbounded max/min reachability.
Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P_s^\Theta(Paths(\Psi)) \geq p$.

Reducing subcases
We can reduce $\leq$ to $\geq$ by:
$\exists \Theta : P_s^\Theta(Paths(\Psi)) \leq p \iff \neg \exists \Theta : P_s^\Theta(Paths(\Psi)) \leq 1 - p$.
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:

$\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P_s^\Theta(\text{Paths}(\neg \Psi)) \geq 1 - p$
LTL Verification

Input: MDP \( \mathcal{M} \), state \( s \), LTL formula \( \Psi \), threshold \( p \in [0,1] \)

Output: TRUE iff \( \exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \geq p \).

Reducing subcases

We can reduce \( \leq \) to \( \geq \) by:

\[ \exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P_s^\Theta(\text{Paths}(\neg \Psi)) \geq 1 - p \]

and necessarily to possibly (\( \forall \rightarrow \exists \)) by:

\[ \forall \Theta : P_s^\Theta(\text{Paths}(\Psi)) > p \iff \]

How to do this?!?
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:
$\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P_s^\Theta(\text{Paths}(\neg \Psi)) \geq 1 - p$
and necessarily to possibly ($\forall \rightarrow \exists$) by:
$\forall \Theta : P_s^\Theta(\text{Paths}(\Psi)) > p \iff \neg \exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p$. 
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P^\Theta_s(Paths(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:
$\exists \Theta : P^\Theta_s(Paths(\Psi)) \leq p \iff \exists \Theta : P^\Theta_s(Paths(\neg\Psi)) \geq 1 - p$
and necessarily to possibly ($\forall \rightarrow \exists$) by:
$\forall \Theta : P^\Theta_s(Paths(\Psi)) > p \iff \neg \exists \Theta : P^\Theta_s(Paths(\Psi)) \leq p$.

Algorithm

How to do this?!
Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$

Output: TRUE iff $\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:

$$\exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P_s^\Theta(\text{Paths}(\neg \Psi)) \geq 1 - p$$

and necessarily to possibly ($\forall \rightarrow \exists$) by:

$$\forall \Theta : P_s^\Theta(\text{Paths}(\Psi)) > p \iff \neg \exists \Theta : P_s^\Theta(\text{Paths}(\Psi)) \leq p.$$  

Algorithm

The algorithm is conceptually the same as for DTMC:

1. transform $\Psi$ to a deterministic Rabin automaton $R$ with $\text{Lang}(R) = \text{Paths}(\Psi)$,
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P_{s}^{\Theta}(\text{Paths}(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:

$\exists \Theta : P_{s}^{\Theta}(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P_{s}^{\Theta}(\text{Paths}(\neg \Psi)) \geq 1 - p$

and necessarily to possibly ($\forall \rightarrow \exists$) by:

$\forall \Theta : P_{s}^{\Theta}(\text{Paths}(\Psi)) > p \iff \neg \exists \Theta : P_{s}^{\Theta}(\text{Paths}(\Psi)) \leq p$.

Algorithm

The algorithm is conceptually the same as for DTMC:

1. transform $\Psi$ to a deterministic Rabin automaton $R$ with $\text{Lang}(R) = \text{Paths}(\Psi)$,
2. construct product MDP $\mathcal{M} \times R$, 

LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0,1]$
Output: TRUE iff $\exists \Theta : P^\Theta_s(Paths(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:
$\exists \Theta : P^\Theta_s(Paths(\Psi)) \leq p \iff \exists \Theta : P^\Theta_s(Paths(\neg \Psi)) \geq 1 - p$
and necessarily to possibly ($\forall \rightarrow \exists$) by:
$\forall \Theta : P^\Theta_s(Paths(\Psi)) > p \iff \neg \exists \Theta : P^\Theta_s(Paths(\Psi)) \leq p$.

Algorithm

The algorithm is conceptually the same as for DTMC:

1. transform $\Psi$ to a deterministic Rabin automaton $R$ with $\text{Lang}(R) = Paths(\Psi)$,
2. construct product MDP $\mathcal{M} \times R$,
3. by graph algorithms, find in the product MDP all accepting end components,
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P_s^\Theta(Paths(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:
$$\exists \Theta : P_s^\Theta(Paths(\Psi)) \leq p \iff \exists \Theta : P_s^\Theta(Paths(\neg \Psi)) \geq 1 - p$$
and necessarily to possibly ($\forall \rightarrow \exists$) by:
$$\forall \Theta : P_s^\Theta(Paths(\Psi)) > p \iff \neg \exists \Theta : P_s^\Theta(Paths(\Psi)) \leq p.$$

Algorithm

The algorithm is conceptually the same as for DTMC:

1. transform $\Psi$ to a deterministic Rabin automaton $R$ with $\text{Lang}(R) = Paths(\Psi)$,
2. construct product MDP $\mathcal{M} \times R$,
3. by graph algorithms, find in the product MDP all accepting end components,
4. their union is denoted by $X$
5. return TRUE iff $\max_\Theta P_s^\Theta(\Diamond X) \geq p$. 
LTL Verification

Input: MDP $\mathcal{M}$, state $s$, LTL formula $\Psi$, threshold $p \in [0, 1]$
Output: TRUE iff $\exists \Theta : P^\Theta_s(\text{Paths}(\Psi)) \geq p$.

Reducing subcases

We can reduce $\leq$ to $\geq$ by:
$\exists \Theta : P^\Theta_s(\text{Paths}(\Psi)) \leq p \iff \exists \Theta : P^\Theta_s(\text{Paths}(\neg \Psi)) \geq 1 - p$

and necessarily to possibly ($\forall \rightarrow \exists$) by:
$\forall \Theta : P^\Theta_s(\text{Paths}(\Psi)) > p \iff \neg \exists \Theta : P^\Theta_s(\text{Paths}(\Psi)) \leq p$.

Algorithm

The algorithm is conceptually the same as for DTMC:

1. transform $\Psi$ to a deterministic Rabin automaton $R$ with $\text{Lang}(R) = \text{Paths}(\Psi)$,
2. construct product MDP $\mathcal{M} \times R$,
3. by graph algorithms, find in the product MDP all accepting end components, $\leftarrow$ How to do this?!!?
4. their union is denoted by $X$
5. return TRUE iff $\max_{\Theta} P^\Theta_s(\Diamond X) \geq p$. 
End Components

- An end component is a subset of states \( S' \) and actions \( A' \) such that
  - \( \sum_{s' \in S'} P(s, \alpha, s') = 1 \) for each \( s \in S' \) and \( \alpha \in A'(s') \) and
  - it is strongly connected (when considering edges of all actions).
- With probability 1, infinitely often visited states on a run form an end component.
An end component is a subset of states $S'$ and actions $A'$ such that

- $\sum_{s' \in S'} P(s, \alpha, s') = 1$ for each $s \in S'$ and $\alpha \in A'(s')$ and
- it is strongly connected (when considering edges of all actions).

With probability 1, infinitely often visited states on a run form an end component.

It is accepting if for some Rabin pair $(E_i, F_i)$ it contains no state of $E_i$ and some state of $F_i$.

But: there are exponentially many end components.
An end component is a subset of states $S'$ and actions $A'$ such that
- $\sum_{s' \in S'} P(s, \alpha, s') = 1$ for each $s \in S'$ and $\alpha \in A'(s')$ and
- it is strongly connected (when considering edges of all actions).

With probability 1, infinitely often visited states on a run form an end component.

It is accepting if for some Rabin pair $(E_i, F_i)$ it contains no state of $E_i$ and some state of $F_i$.

But: there are exponentially many end components.

The solution: Maximal end components
- Maximal exist as union of two non-disjoint end components is an end component.
- Thus, we can deal with partition, instead.
A partition-refinement algorithm

Start with partition \( \{S\} \). In each iteration for each partition class \( T \).

1. Find in the induced subgraph of \( T \) (when considering edges of all actions) all SCCs that have at least one edge.
2. Repetitively:
   (a) Remove all actions that leave with positive probability its SCC.
   (b) Remove from each SCC all states that have no actions.
3. Replace \( T \) by what is left of each SCC.
4. Newly added classes may be not strongly-connected, repeat.
A partition-refinement algorithm

Start with partition \( \{S\} \). In each iteration for each partition class \( T \).

1. Find in the induced subgraph of \( T \) (when considering edges of all actions) all SCCs that have at least one edge.
2. Repetitively:
   (a) Remove all actions that leave with positive probability its SCC.
   (b) Remove from each SCC all states that have no actions.
3. Replace \( T \) by what is left of each SCC.
4. Newly added classes may be not strongly-connected, repeat.
Accepting Maximal End Components

Accepting MEC for Rabin condition \((E_i, F_i)_{i \in I}\)

- For each \(i \in I\), run the algorithm with initial “partitioning” \(S \setminus E_i\)
i.e. construct an MDP \(M_i\) by removing states \(E_i\) and repetitively removing
(a) actions that lead with positive probability to some removed state and
(b) states with no actions,
then run the algorithm
- Accepting MEC in each \(M_i\) are those containing some state of \(F_i\).
Analysis questions

- Reachability: LP or VI
- Probabilistic logics: combination of the techniques
- Transient analysis: preference over $S$ needed
- Steady-state analysis: preference over $S$ needed
- Rewards: solves transient and steady-state analysis

For best/worst transient/steady-state distribution, a preference over $S$ needed

- Step bounded reachability $\Diamond \leq n B$ is one approach to distribution after $n$ steps (preferred are exactly the states in $B$).
- A more fine tuned preference can be specified by rewards
MDP – Rewards

- expected instantaneous reward
- expected mean payoff
MDP – Rewards

**Instantaneous rewards**
What is the maximal expected number of my pieces in the play area after 50 rounds?

**Step-bounded cumulative rewards**
What is the maximal expected number of times I kick out a piece of the opponent within the first 100 steps?

**Cumulative rewards to reach a target**
What is the minimal expected number of steps before the game ends?

**Mean payoff (long-run average reward)**
What is the average number of pieces on board? (restart after game end ⇒ infinite run)
Definition
\[ \sup_{\Theta} \mathbb{E}^{\Theta}[l_r^k] \text{ where } l_r^k(\xi(s_0s_1\ldots)) = r(s_k) \]
**Definition**

$\sup_{\Theta} \mathbb{E}_{\Theta}^s[I_r^k] \text{ where } I_r^k(\xi(s_0s_1 \ldots)) = r(s_k)$

**Theorem**

For an MDP with reward $r$, the vector $x(s) = \sup_{\Theta} \mathbb{E}_{\Theta}^s[I_r^k]$ equals to $x^k(s)$ where

$$x^\ell(s) = \begin{cases} r(s) & \text{if } \ell = 0 \\
\max_{\alpha \in \text{Act}(s)} \sum_{s' \in S} P(s, \alpha, s') \cdot x^{\ell-1}(s') & \text{otherwise}
\end{cases}$$
**Definition**

\[
\sup_{\Theta} E^\Theta[I_r^{=k}] \text{ where } I_r^{=k}(\xi(s_0s_1\ldots)) = r(s_k)
\]

**Theorem**

For an MDP with reward \( r \), the vector \( x(s) = \sup_{\Theta} E^\Theta_s[I_r^{=k}] \) equals to \( x^k(s) \) where

\[
x^\ell(s) = \begin{cases} 
    r(s) & \text{if } \ell = 0 \\
    \max_{\alpha \in \text{Act}(s)} \sum_{s' \in S} P(s, \alpha, s') \cdot x^{\ell-1}(s') & \text{otherwise}
\end{cases}
\]

**Corollary**

There are optimal deterministic schedulers for \( \max E^\Theta_s[I_r^{=k}] \) (and similarly min).
**Definition**

\[ \sup_\Theta E^\Theta[I_r^{=k}] \text{ where } I_r^{=k}(\xi(s_0s_1 \ldots)) = r(s_k) \]

**Theorem**

For an MDP with reward \( r \), the vector \( x(s) = \sup_\Theta E_s^\Theta[I_r^{=k}] \) equals to \( x^k(s) \) where

\[
x^\ell(s) = \begin{cases} r(s) & \text{if } \ell = 0 \\
\max_{\alpha \in \text{Act}(s)} \sum_{s' \in S} P(s, \alpha, s') \cdot x^{\ell-1}(s') & \text{otherwise}
\end{cases}
\]

**Corollary**

There are optimal deterministic schedulers for \( \max E_s^\Theta[I_r^{=k}] \) (and similarly min).

**What about step-bounded cumulative reward?**

\[
x^\ell(s) = \begin{cases} 0 & \text{if } \ell = 0 \\
r(s) + \max_{\alpha \in \text{Act}(s)} \sum_{s' \in S} P(s, \alpha, s') \cdot x^{\ell-1}(s') & \text{otherwise}
\end{cases}
\]
Recall mean payoff (long-run average reward):

$$R_1 R_2 \cdots = 42 \ 1 \ 2 \ 1 \ 2 \ 1 \ 2 \cdots$$

$$\lim_{n \to \infty} \frac{\sum_{i=1}^{n} R_i}{n} = 1.5$$

Example: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff
Recall mean payoff (long-run average reward):

$$R_1 R_2 \cdots = 42 \ 1 \ 2 \ 1 \ 2 \ 1 \ 2 \ \cdots$$

$$\lim_{n \to \infty} \frac{\sum_{i=1}^{n} R_i}{n} = 1.5$$

Example: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff

Limit may not exist:

$$0 \ (1)^{10} \ (0)^{1000} \ (1)^{1000000} \ \cdots$$

$$\lim_{n \to \infty} \text{lim inf } \frac{\sum_{i=1}^{n} R_i}{n} = 0$$

Definition

$$\sup_{\Theta} \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} E[\Theta [r(A_i)]]$$

where $A_i$ is (random variable for) $i$th action
For ergodic systems, extensible to general but more complicated

Value vector $\vec{v}$ found by successive approximation. $\vec{w}^t$ is the optimal total reward in time $t$.

1. Choose $\epsilon > 0$, and take $\vec{w}^0 := \vec{0} \in \mathbb{R}^{|S|}$
2. Compute iteration:

$$\vec{w}_s^{t+1} := \max_{a \in \text{Act}(s)} r(a) + \sum_{s' \in S} \delta(a)(s') \vec{w}_{s'}, \text{ for } s \in S \quad (1)$$
For ergodic systems, extensible to general but more complicated

Value vector $\vec{v}$ found by successive approximation

$\vec{w}^t$ is the optimal total reward in time $t$

1. Choose $\varepsilon > 0$, and take $\vec{w}^0 := \vec{0} \in \mathbb{R}^{|S|}$

2. Compute iteration:

   $$\vec{w}^{t+1}_s := \max_{a \in \text{Act}(s)} r(a) + \sum_{s' \in S} \delta(a)(s') \vec{w}_{s'}, \text{ for } s \in S$$ (1)

3. Compute error

   $\text{upper} := \max_{s \in S}(\vec{w}^{t+1}_s - \vec{w}^t_s)$

   $\text{lower} := \min_{s \in S}(\vec{w}^{t+1}_s - \vec{w}^t_s)$

   If $\text{upper} - \text{lower} > \varepsilon$: go to step 2. with $t := t+1$

   else $\frac{\text{upper} + \text{lower}}{2}$ is a $\frac{\varepsilon}{2}$-approximation of the value $\vec{v}$ (Stop)

   Optimal strategy: pick maximum in (1)

$\text{upper}$ and $\text{lower}$ approximate $\vec{v}$ from above and below, respectively

$\vec{w}^{t+1} - \vec{w}^t$, converges to $\vec{v}$
Sequence $f^0, f^1, \ldots$ of strategies such that $\bar{v}(f^{t+1}) \geq \bar{v}(f^t)$ and converging to an optimal strategy

Finitely many strategies $\Rightarrow$ termination

\[
\begin{align*}
\text{for all } s \in S: \quad \vec{x}_s &= \sum_{s' \in S} \delta(f(s), s') \vec{x}_{s'} \\
\text{for all } s \in S: \quad \vec{x}_s + \vec{y}_s &= \sum_{s' \in S} \delta(f(s), s') \vec{y}_{s'} + r(f(s)) \\
\text{for all } s \in S: \quad \vec{y}_s + \vec{z}_s &= \sum_{s' \in S} \delta(f(s), s') \vec{z}_{s'}
\end{align*}
\] (2)

\( \vec{x} \) is equal to $\mathbb{E}^f[MP]$

\( \vec{y} \) is the difference between total and long-run rewards

\( \vec{z} \) is used in the algorithm to prevent cycling
MDP – Mean payoff – Strategy iteration II

Using \((\vec{x}, \vec{y})\)

\[
B(s, f) = \left\{ a \in \text{Act}(s) \right\} \begin{cases} \sum_{s'} \delta(a)(s') \vec{x}_{s'} > \vec{x}_s \text{ or } \\ \sum_{s'} \delta(a)(s') \vec{x}_{s'} = \vec{x}_s \text{ and } \\ r(a) + \sum_{s'} \delta(a)(s') \vec{y}_{s'} > \vec{x}_s + \vec{y}_s \end{cases}
\] (3)

1. Start with any \(f \in F\).
2. Determine unique \((\vec{x}, \vec{y})\)-part in a solution of the linear system (2)
3. For every \(s \in S\): determine \(B(s, f)\) as defined in (3) using the values \(\vec{x}\) and \(\vec{y}\) from step 2
4. If \(B(s, f) = \emptyset\) for every \(s \in S\): go to step 6
   Otherwise: take any \(g \neq f\) such that \(g(s) \in B(s, f)\) if \(g(s) \neq f(s)\)
5. \(f := g\) and go to step 2
6. \(f\) is an average optimal strategy
\( \vec{v} \) the smallest solution of LP, strategy derived from its dual LP

Primary linear program:

Minimize:

\[
\sum_{s \in S} \vec{\mu}_s \vec{x}_s
\]

Subject to:

for all \( s \in S, a \in \text{Act}(s) \):

\[
\vec{x}_s \geq \sum_{s' \in S} \delta(a)(s') \vec{x}_{s'}
\]

for all \( s \in S, a \in \text{Act}(s) \):

\[
\vec{x}_s \geq r(a) + \sum_{s' \in S} \delta(a)(s') \vec{y}_{s'} - \vec{y}_s
\]

where \( \vec{\mu}_s > 0 \) arbitrarily chosen
Dual linear program:

Maximize:

$$\sum_{a \in A} r(a)\vec{x}_a$$

Subject to:

for all $s \in S$:

$$\vec{\mu}_s + \sum_{a \in A} \delta(a)(s)\vec{y}_a = \sum_{a \in \text{Act}(s)} \vec{y}_a + \sum_{a \in \text{Act}(s)} \vec{x}_a$$

for all $s \in S$:

$$\sum_{a \in A} \delta(a)(s)\vec{x}_a = \sum_{a \in \text{Act}(s)} \vec{x}_a$$

$\vec{x}$: occupation measure in the limit
$\vec{y}_a$: expected number of taking action $a$ during the transient phase

both flows subject to Kirchhoff’s law
Optimal strategy: $f$ such that

- $\bar{x}_{f(s)} > 0$ if $s \in S_{\bar{x}}$
- $\bar{y}_{f(s)} > 0$ if $s \notin S_{\bar{x}}$

where $S_{\bar{x}} := \{s \in S \mid \sum_{a \in \text{Act}(s)} \bar{x}_a > 0\}$
Optimize multiple mean payoffs $MP_i, i \in \{1, \ldots, n\}$, in MDP:

- **expectation**
  $$\bigwedge_i \mathbb{E}[MP_i] \geq \exp_i$$

- **satisfaction** (quantiles, percentiles)
  - **conjunctive**
    $$\bigwedge_i \mathbb{P}[MP_i \geq \text{sat}_i] \geq \text{prob}_i$$
  - **joint**
    $$\mathbb{P}[\bigwedge_i MP_i \geq \text{sat}_i] \geq \text{prob}$$

- **conjunctions** thereof [CKK15,CR15]
Examples

Example 1: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff $\mathbb{E}[MP]$
Examples

Example 1: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff $\mathbb{E}[MP]$
- maximize probability $\mathbb{P}[MP \geq 0]$
Examples

Example 1: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff $\mathbb{E}[MP]$
- maximize probability $\mathbb{P}[MP \geq 0]$
- maximize $\mathbb{E}[MP]$ while ensuring $\mathbb{P}[MP \geq 0] \geq 0.95$

“risk-averse” strategies
Examples

Example 1: Money investment

- $> 0$ earning, $< 0$ losing
- maximize expected mean payoff $\mathbb{E}[MP]$
- maximize probability $\mathbb{P}[MP \geq 0]$
- maximize $\mathbb{E}[MP]$ while ensuring $\mathbb{P}[MP \geq 0] \geq 0.95$

“risk-averse” strategies

Example 2: Downloading service (multiple mean payoffs)

- gratis service: expected throughput $MP_1 \geq 1Mbps$
- premium service: $\mathbb{E}[MP_2] \geq 10Mbps$ and $\geq 95\%$ connections run on $\geq 5Mbps$; sold at $p_2$ per $Mb$
- need to hire $MP_3$ resources from a cloud each at price $p_3$
- while satisfying the guarantees, maximize $\mathbb{E}[p_2 \cdot MP_2 - p_3 \cdot MP_3]$
Example

\[ \exp = (1.1, 0.5), \text{sat} = (0.5, 0.5), \text{prob} = (0.8, 0.8) \]
Example

\[ a, (4, 0) \quad b, (1, 0) \quad d, (0, 1) \]

\[ s \]

\[ \ell, 0.5 \quad r, 0.5 \]

\[ v \quad c, (0, 0) \quad e, (0, 0) \]

\[ exp = (1.1, 0.5), \ sat = (0.5, 0.5), \ prob = (0.8, 0.8) \]
Example

\[ \text{exp} = (1.1, 0.5), \quad \text{sat} = (0.5, 0.5), \quad \text{prob} = (0.8, 0.8) \]

- linear programming
- feasible and practically useful
Model Construction Principles
The setting

- “Real” parallel system: $P = P_1 \parallel \ldots \parallel P_n$. 
The setting

- “Real” parallel system: $P = P_1 \parallel \ldots \parallel P_n$.
- Transition system: $T = T_1 \parallel \ldots \parallel T_n$. 
The setting

- “Real” parallel system: $P = P_1 \parallel \ldots \parallel P_n$.
- Transition system: $T = T_1 \parallel \ldots \parallel T_n$.

Our goal: Define semantic parallel operators on transition systems to model “real” parallel operators.
The setting

- “Real” parallel system: \( P = P_1 \parallel \ldots \parallel P_n \).
- Transition system: \( T = T_1 \parallel \ldots \parallel T_n \).

**Our goal:** Define semantic parallel operators on transition systems to model “real” parallel operators.

In the following we:

1. recall the notions **without** randomness
2. observe how to **add** the randomness
A transition system in a tuple

\[ T = (S, Act, \rightarrow, s_0, AP, L) \]

- \( S \) is the state space, i.e., set of states,
- \( Act \) is a set of actions,
- \( \rightarrow \subseteq S \times Act \times S \) is the transition relation of the form \( s \xrightarrow{\alpha} s' \)
  where \( s, s' \in S \) and \( \alpha \in Act \).
- \( s_0 \in S \) is the initial state,
- \( AP \) is a set of atomic propositions,
- \( L : S \rightarrow 2^{AP} \) is the labelling function.
1. **Pure concurrency**: Interleaving operator, no communication, no dependencies
2. **Synchronous product**: For hardware systems with a shared clock
3. **Synchronous message passing**
4. **Communication via shared variables**
5. **Channel systems**: Shared variables + communication via channels
\[ \mathcal{T}_1 = (S_1, \text{Act}_1, \rightarrow_1, s_{01}, \text{AP}_1, L_1) \]

\[ \mathcal{T}_2 = (S_2, \text{Act}_2, \rightarrow_2, s_{02}, \text{AP}_2, L_2) \]

The composite transition system \( \mathcal{T}_1 \| \mathcal{T}_2 \) is given by:

\[ \mathcal{T}_1 \| \mathcal{T}_2 = (S_1 \times S_2, \text{Act}_1 \cup \text{Act}_2, \rightarrow, \langle s_{01}, s_{02} \rangle, \text{AP}, L) \]

where \( \rightarrow \) is given by:

\[
\begin{array}{c}
\frac{s_1 \xrightarrow{\alpha} s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \quad \frac{s_2 \xrightarrow{\alpha} s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}
\end{array}
\]

atomic propositions: \( \text{AP} = \text{AP}_1 \cup \text{AP}_2 \)
labelling function: \( L(\langle s_1, s_2 \rangle) = L(s_1) \cup L(s_2) \)
The composite transition system $\mathcal{T}_1 \otimes \mathcal{T}_2$ is given by:

\[ \mathcal{T}_1 \otimes \mathcal{T}_2 = (S_1 \times S_2, \text{Act}, \rightarrow, \langle s_{01}, s_{02} \rangle, \text{AP}, L) \]

where $\rightarrow$ is given by:

\[
\langle s_1, s_2 \rangle \xrightarrow{\alpha \ast \beta} \langle s'_1, s'_2 \rangle
\]

$\ast : \text{Act}_1 \times \text{Act}_2 \rightarrow \text{Act}$
$T_1 = (S_1, \text{Act}_1, \rightarrow_1, s_{01}, \text{AP}_1, L_1)$
$T_2 = (S_2, \text{Act}_2, \rightarrow_2, s_{02}, \text{AP}_2, L_2)$

Concurrent execution with synchronization over all actions in $\text{Syn} \subseteq \text{Act}_1 \cap \text{Act}_2$:

$T_1 \parallel_{\text{syn}} T_2 = (S_1 \times S_2, \text{Act}_1 \cup \text{Act}_2, \rightarrow, \langle s_{01}, s_{02} \rangle, \text{AP}, L)$

- Interleaving for $\alpha \notin \text{Syn}$:

$\frac{s_1 \rightarrow_1 s'_1}{\langle s_1, s_2 \rangle \rightarrow_\alpha \langle s'_1, s_2 \rangle}$
$\frac{s_2 \rightarrow_2 s'_2}{\langle s_1, s_2 \rangle \rightarrow_\alpha \langle s_1, s'_2 \rangle}$

- Handshaking for $\alpha \in \text{Syn}$:

$\frac{s_1 \rightarrow_1 s'_1 \land s_2 \rightarrow_2 s'_2}{\langle s_1, s_2 \rangle \rightarrow_\alpha \langle s'_1, s'_2 \rangle}$
1. **Pure concurrency**: Interleaving operator, no communication, no dependencies
2. **Synchronous product**: For hardware systems with a shared clock
3. **Synchronous message passing**: Interleaving + synchronization
4. **Communication via shared variables**
   - Encode possible variable values as states
   - Transition system describes possible updates and lookups
   - Resort to synchronous message passing
5. **Channel systems**: Shared variables + communication via channels
   - Communication over **shared variables**
   - synchronous message passing (channels of capacity 0)
   - asynchronous message passing (channels of capacity $\geq 1$)

**can be encoded into**
- transition systems using only
- synchronous message passing
Given $n$ different processes $i = 1, \ldots, n$

To model variable $x$ with values $V = \{v_1, \ldots, v_m\}$

Introduce another process and new actions

$T_x = (S_x, Act_x, \rightarrow_x, \ldots)$

- $S_x = \{v_1, \ldots, v_m\}$
- $Act_x = \{get_{x,i,v}, set_{x,i,v} \mid i \in \{1, \ldots, n\}, v \in V\}$
- $\rightarrow_x = \{(v, get_{x,i,v}, v), (v, set_{x,i,v}, v') \mid i \in \{1, \ldots, n\}, v \in V, v' \in V\}$
- $Act$ of process $i$ is extended by $Act_x$ to get and set the variable $x$

Mathematical operations can be derived
- Extension similar to shared variables
- Use transition system to model channel
  - parallel composition
  - rename actions as needed
- Pure concurrency and Synchronous product are special cases of synchronous message passing
- Communication via shared variables and Channel systems can be encoded by synchronous message passing
Model Construction Principles
The Stochastic Case
The composite transition system $D_1 \parallel D_2$ is given by:

$D_1 \parallel D_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$

where $\rightarrow$ is given by:

\[
\begin{align*}
\langle s_1, s_2 \rangle &\stackrel{\alpha_{1}}{\rightarrow_{1}} \langle \mu_1, s_2 \rangle, \\
\langle s_1, s_2 \rangle &\stackrel{\alpha_{2}}{\rightarrow_{2}} \langle s_1, \mu_2 \rangle,
\end{align*}
\]

where $\langle \mu_1, s_2 \rangle(\langle s'_1, s'_2 \rangle) = \mu_1(s'_1)$ if $s'_2 = s_2$ and 0 otherwise, and $\langle s_1, \mu_2 \rangle(\langle s'_1, s'_2 \rangle) = \mu_2(s'_2)$ if $s'_1 = s_1$ and 0 otherwise.
Recall:

\[ T_1 = (S_1, \text{Act}_1, \rightarrow_1, \ldots) \quad T_2 = (S_2, \text{Act}_2, \rightarrow_2, \ldots) \]

Concurrent execution with synchronization over all actions in \( \text{Syn} \subseteq \text{Act}_1 \cap \text{Act}_2 \):

\[ T_1 \parallel_{\text{Syn}} T_2 = (S_1 \times S_2, \text{Act}_1 \cup \text{Act}_2, \rightarrow, \ldots) \]

- **Interleaving for** \( \alpha \notin \text{Syn} \):

  \[
  \frac{s_1 \xrightarrow{\alpha} s'_1}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s_2 \rangle}
  \]

  \[
  \frac{s_2 \xrightarrow{\alpha} s'_2}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s'_2 \rangle}
  \]

- **Handshaking for** \( \alpha \in \text{Syn} \):

  \[
  \frac{s_1 \xrightarrow{\alpha} s'_1 \land s_2 \xrightarrow{\alpha} s'_2}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s'_2 \rangle}
  \]
\( \mathcal{D}_1 = (S_1, \text{Act}_1, \rightarrow_1, \ldots) \quad \mathcal{D}_2 = (S_2, \text{Act}_2, \rightarrow_2, \ldots) \)

Concurrent execution with synchronization over all actions in \( \text{Syn} \subseteq \text{Act}_1 \cap \text{Act}_2 \):

\( \mathcal{D}_1 \parallel_{\text{Syn}} \mathcal{D}_2 = (S_1 \times S_2, \text{Act}_1 \cup \text{Act}_2, \rightarrow, \ldots) \)

- **Interleaving for \( \alpha \notin \text{Syn} \):**

  \[
  \begin{align*}
  \frac{s_1 \xrightarrow{\alpha_1} \mu_1 \quad s_2 \xrightarrow{\alpha_2} \mu_2}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle \mu_1, s_2 \rangle}
  \end{align*}
  \]

- **Handshaking for \( \alpha \in \text{Syn} \):**

  \[
  \begin{align*}
  \frac{s_1 \xrightarrow{\alpha_1} \mu_1 \wedge s_2 \xrightarrow{\alpha_1} \mu_2}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle \mu_1, \mu_2 \rangle}
  \end{align*}
  \]

where \( \langle \mu_1, \mu_2 \rangle(\langle s'_1, s'_2 \rangle) = \mu_1(s'_1) \cdot \mu_2(s'_2) \).
What is $s_0 \parallel \{\alpha\} t_0$?
Probabilistic automata - Example

\[
\begin{align*}
\langle s_1, t_1 \rangle & \xrightarrow{\alpha} s_0 \parallel \{\alpha\} t_0 \\
\langle s_2, t_1 \rangle & \xrightarrow{\beta, 0.2} s_0 \parallel \{\alpha\} t_0 \\
\langle s_1, t_2 \rangle & \xrightarrow{\alpha} \langle s_1, t_0 \rangle \\
\langle s_2, t_2 \rangle & \xrightarrow{\alpha} \langle s_2, t_0 \rangle
\end{align*}
\]
Pure concurrency
- Synchronous product
- Synchronous message passing
- Communication via shared variables
- Channel systems

What is the difference pf PA to MDPs, actually?
Probabilistic automata - Parallelism & Communication

- Pure concurrency
- Synchronous product
- Synchronous message passing
- Communication via shared variables
- Channel systems

What is the difference pf PA to MDPs, actually?
MDP: each state has at most one transition for a given action.
PA: each state can have several transitions for a given action.
Outlook

Further models

- PTA, Attack trees
- STA
- CTMC, CTMDP, fault trees (transient, steady-state, CSL)
- hybrid automata (reachability)
- corresponding games