Course 16:198:520: Introduction To Artificial Intelligence
Lecture 13

Decision Making

Abdeslam Boularias

Monday, November 30, 2015

Overview

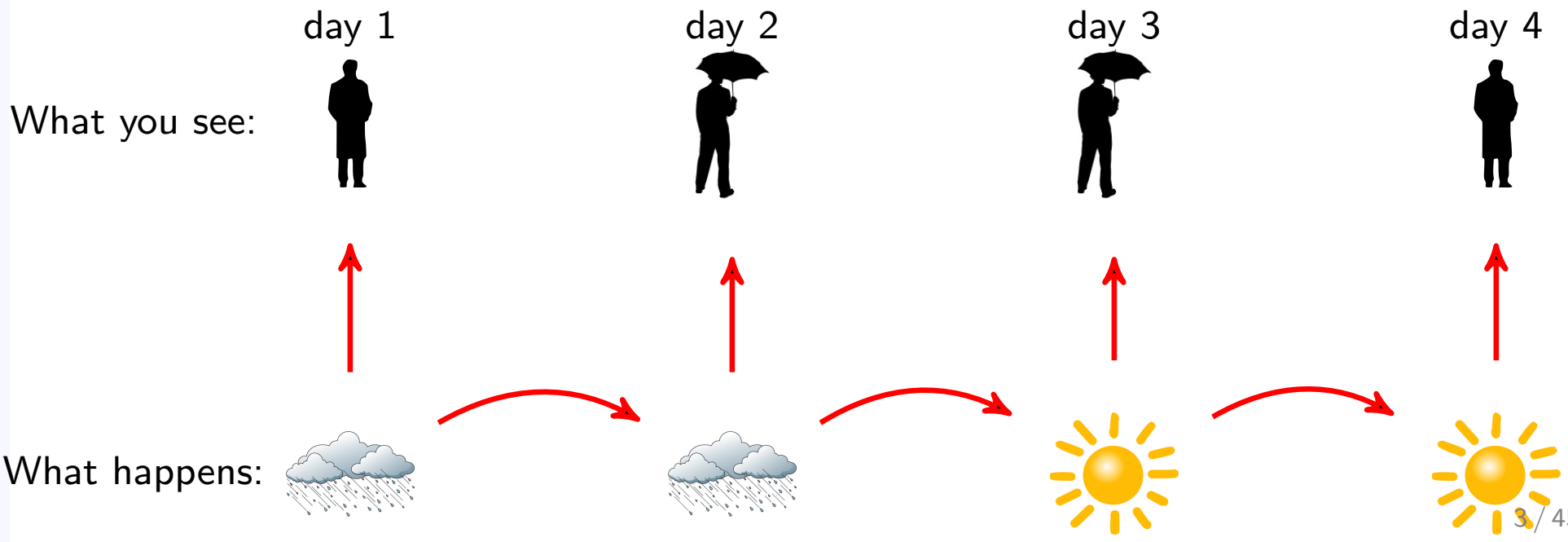
We consider probabilistic temporal models where the values of certain variables are chosen (controlled) by a rational agent.

Outline

1. Simple decision-making
2. Sequential decision-making
3. Markov Decision Process (MDP)
4. Planning algorithms for Markov Decision Processes
5. Learning algorithms for Markov Decision Processes

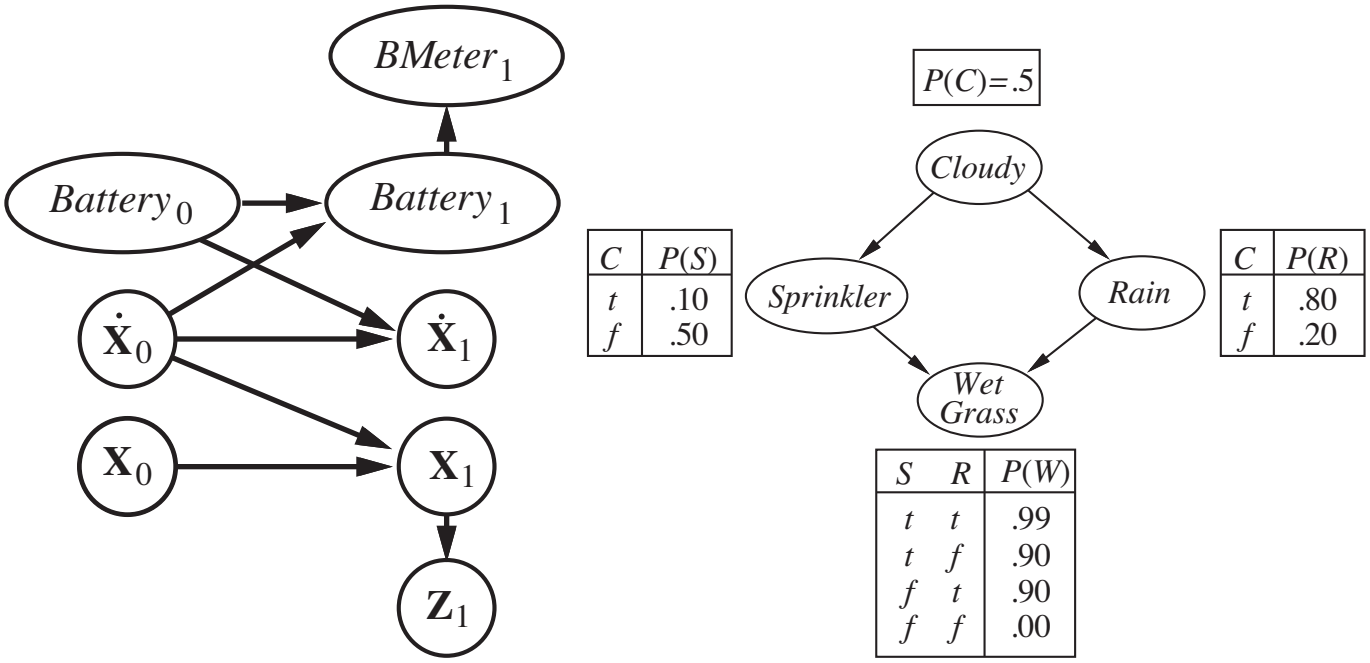
Decision-making

- So far, we have focused on temporal models that describe how a set of random variables, called **state**, changes over time.
- These are models of **uncontrolled systems**: we just observe, predict, estimate, but we do nothing to change how things go.
- In reality, some variables of the system can be **controlled** by a rational agent.



Decision-making

- So far, we have focused on temporal models that describe how a set of random variables, called **state**, changes over time.
- These are models of **uncontrolled systems**: we just observe passively, predict, estimate, but we do nothing to change how things go.
- In reality, some variables of the system can be **controlled** by a rational agent.



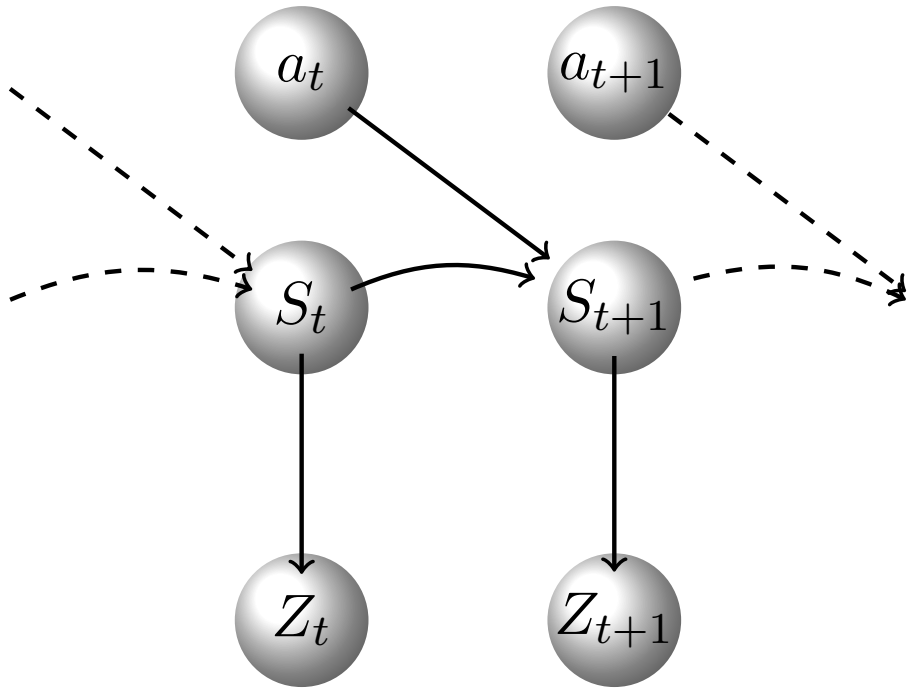
Decision-making

- The set of controlled variables is called an **action**.
- Decision-making: choosing values for the actions.
- The distribution of next states (at $t+1$) depends on the current state and the executed action (at t).

Action:

State:

Observations:



Interaction between an agent and a dynamical system

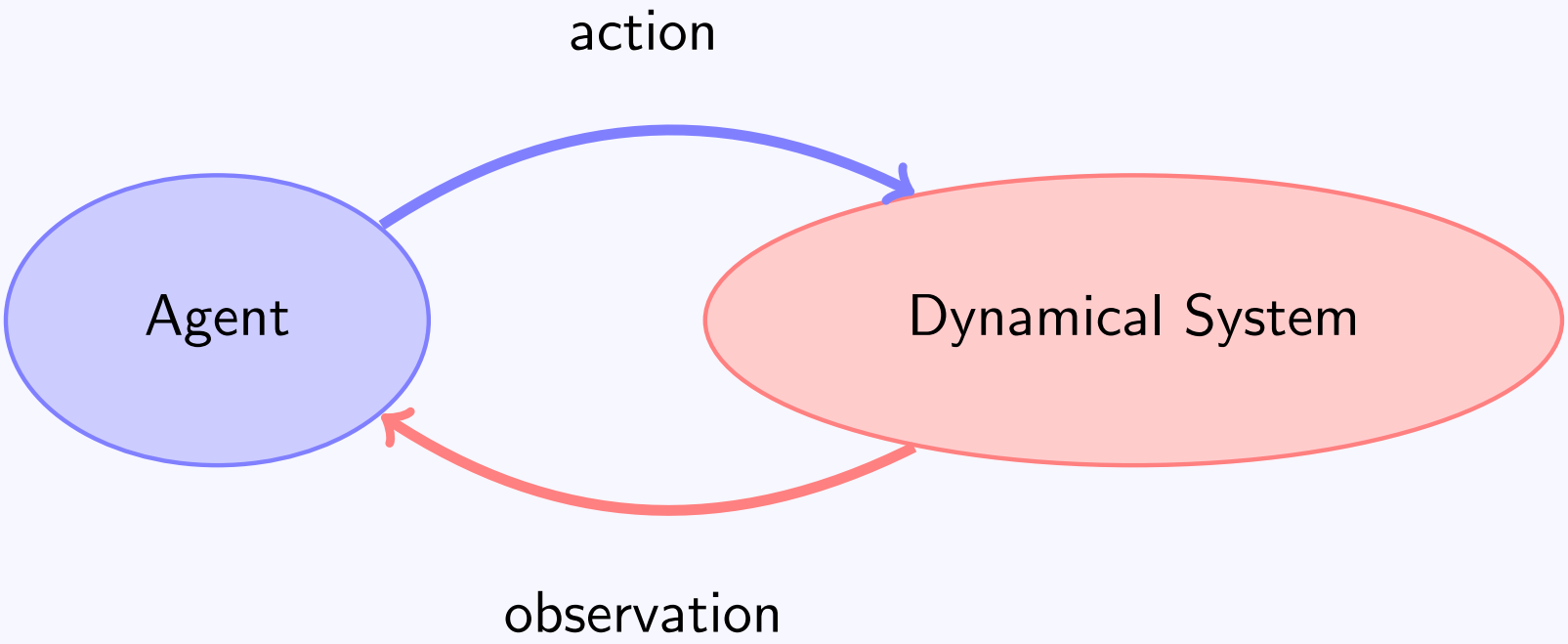


Figure: The process of interaction between an agent and a dynamical system.

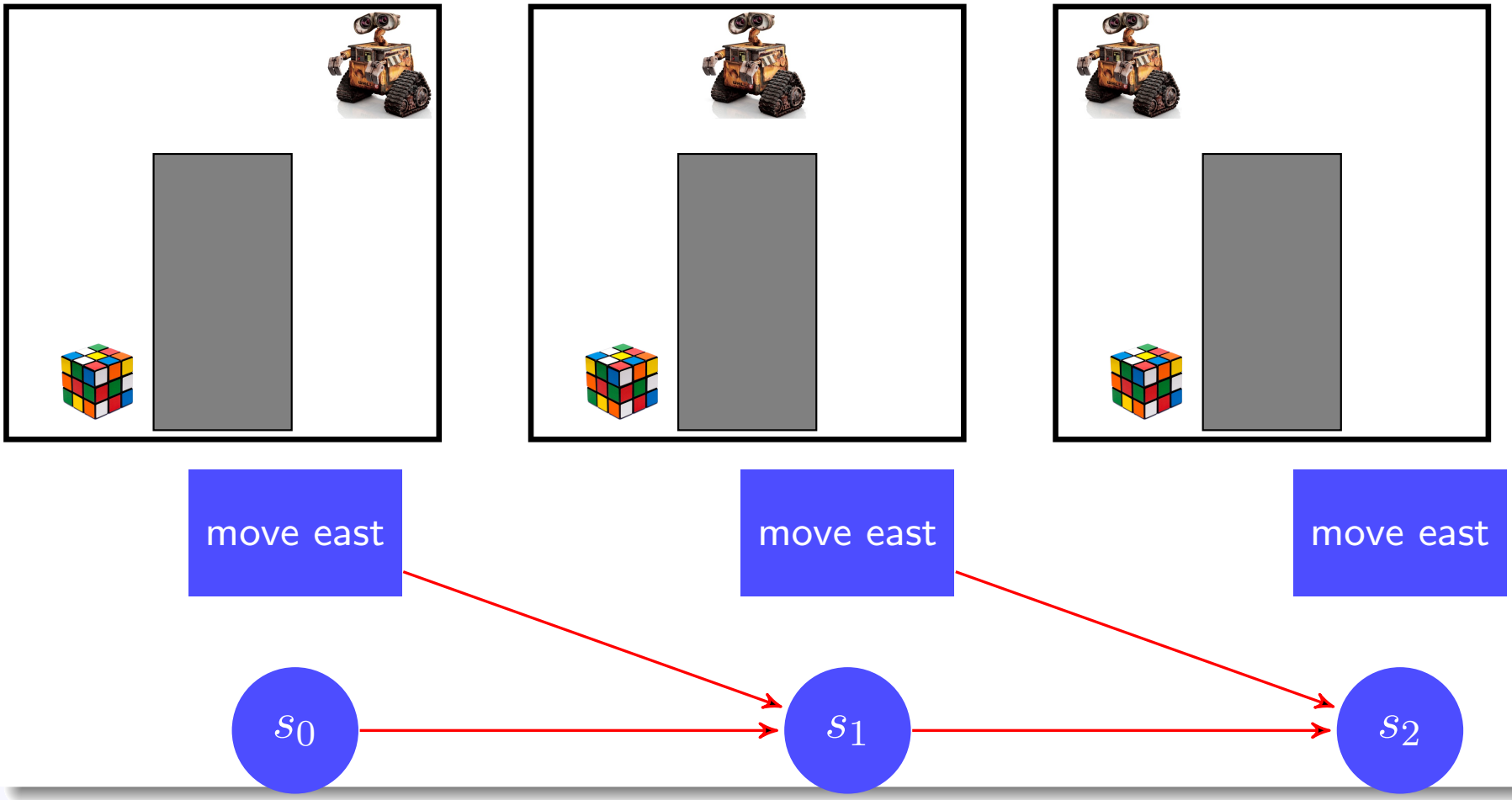
In this lecture, we consider only **fully observable** Markov Decision Processes, where the agent knows the state at any time ($Z_t = S_t$).

Decision-making

- Search problems are decision-making problems: actions correspond to assigning values to variables.
- Navigation problems are decision-making problems: Actions correspond to moving in a particular direction. States correspond to the position of the agent.

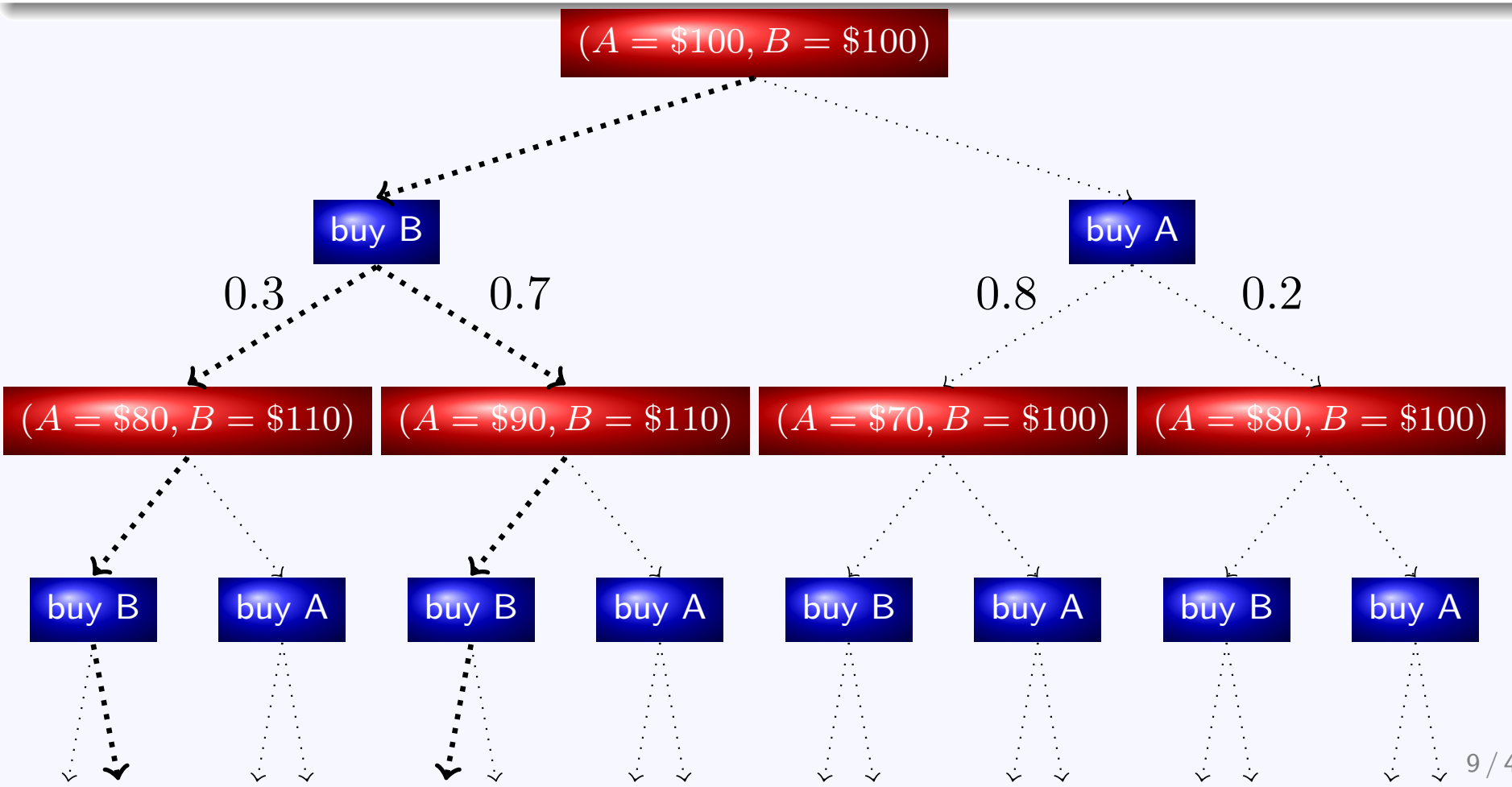
Example of decision-making problems: robot navigation

- State: position of the robot
- Actions: move east, move west, move north, move south.



Example of decision-making problems: buying stocks

- State: prices of stock A and B, how much money we invested so far, and which stocks we have bought.
- Action: buying stock A or B.



Value (Utility)

- An action applied at a given state may lead to different outcomes with varying probabilities.
- Example: If we buy stock A, its price can change to $\$70$ with probability 0.8 and to $\$80$ with probability 0.2.
- Decision theory, in its simplest form, deals with choosing among actions based on the desirability of their immediate outcomes.
- A value (or utility) function V maps every state into a real value that indicates how much we would like to be in that state.
- Example:

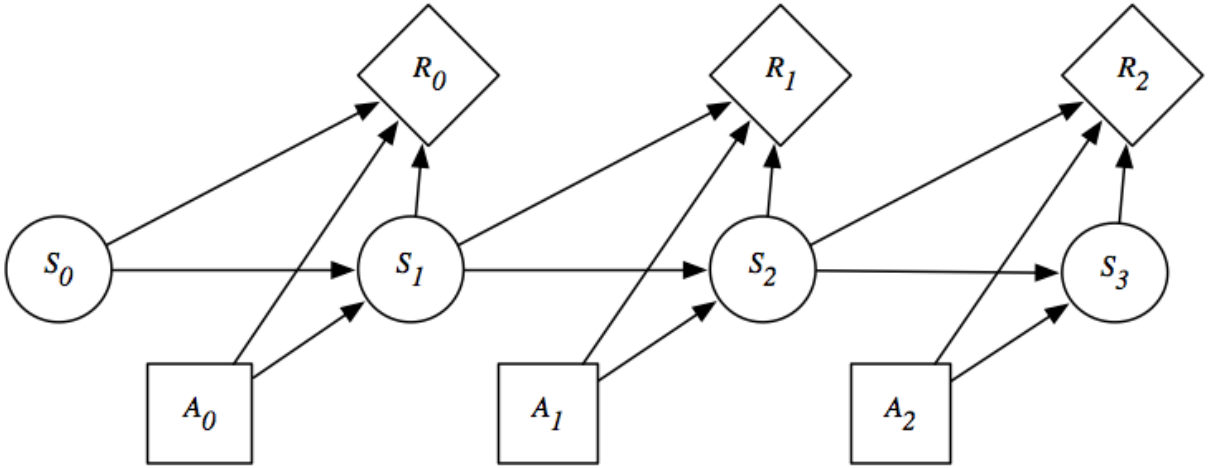
$$V(A = \$90, B = \$110, \text{stocks we have} = \{A\}, \text{invested money} = \$100) = -10$$

- Note: In Russell and Norvig textbook, the term utility (denoted by U) is used instead of value (denoted by U). The two terms are equivalent.

Markov Decision Process (MDP)

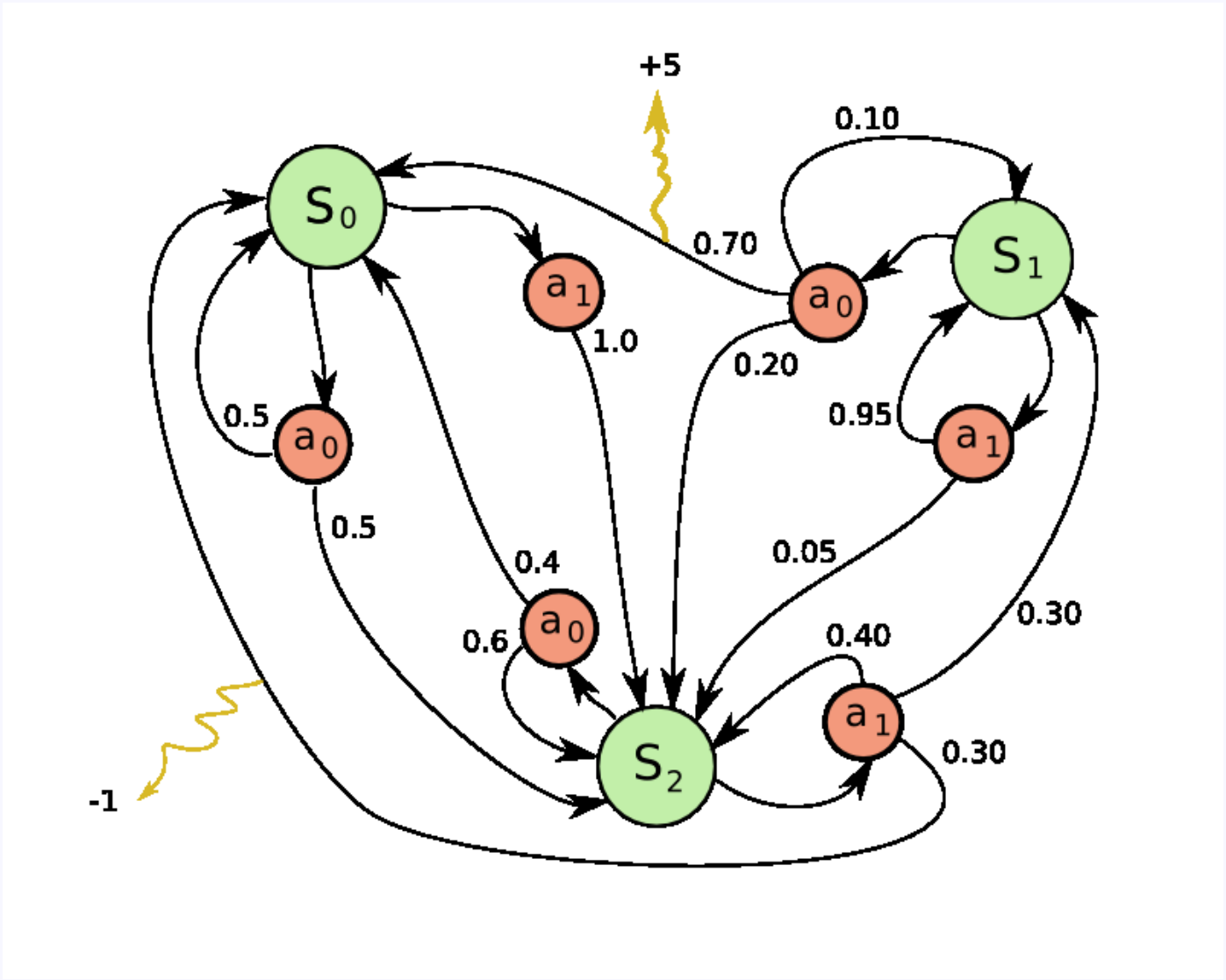
Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where:

- $\mathcal{S}$: is the space of state values.
- $\mathcal{A}$: is the space of action values.
- T: is the transition matrix.
- R: is a reward function.



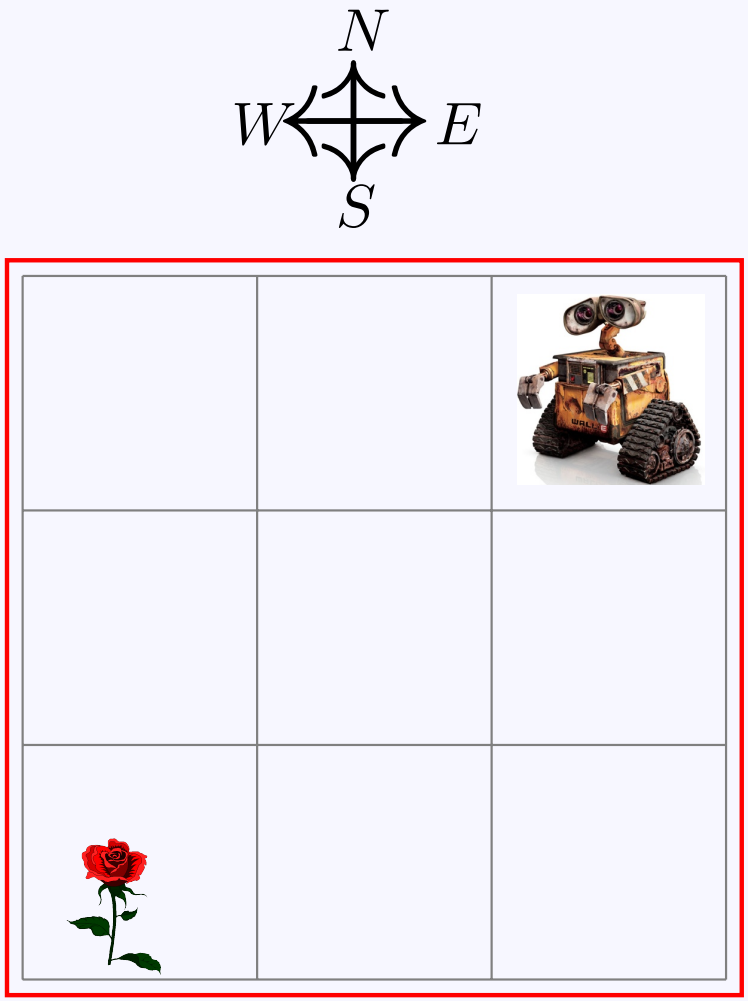
from http://artint.info

Example of a Markov Decision Process with three states and two actions

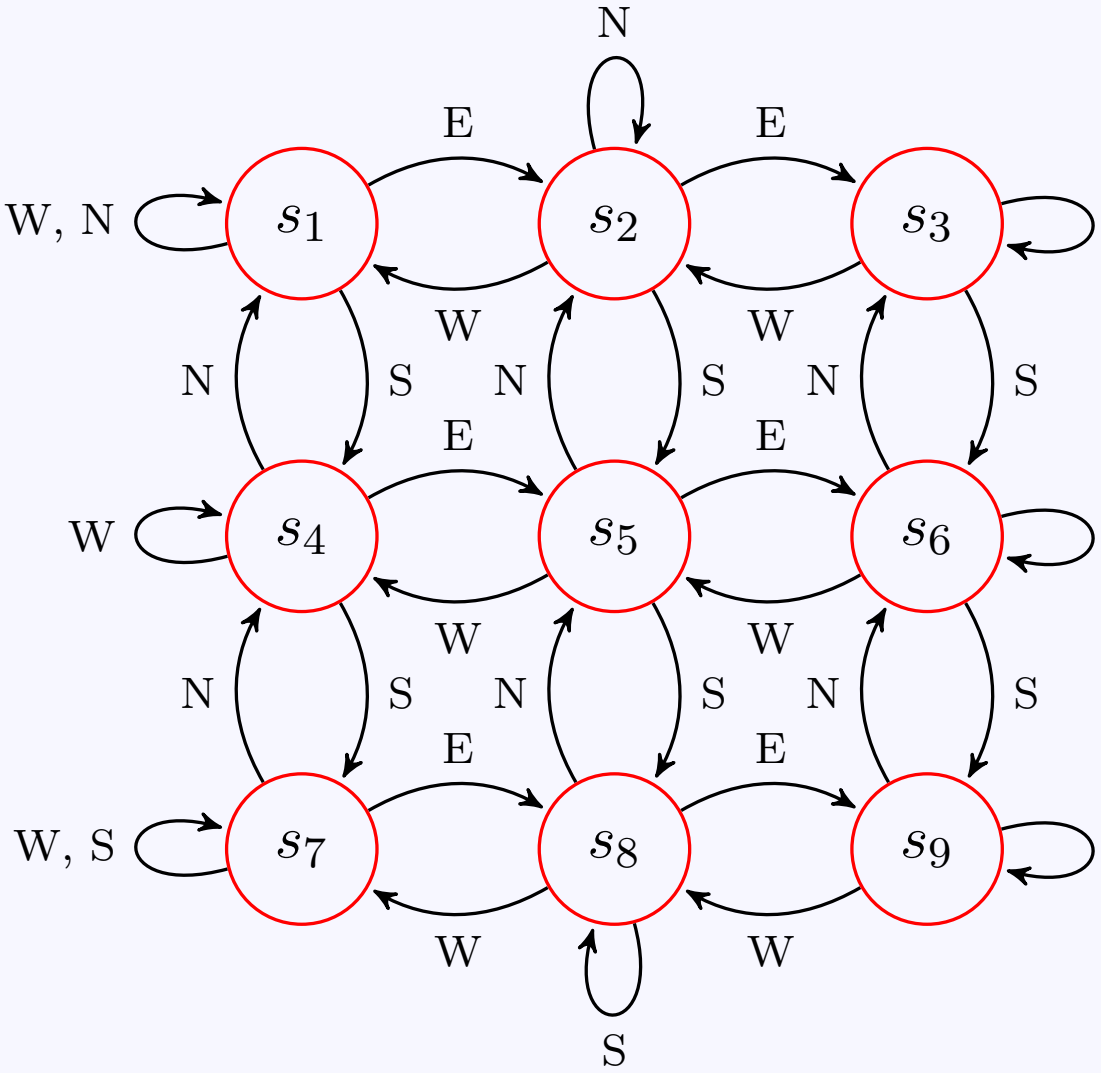


from Wikipedia

Example of a Markov Decision Process



(a) A simple navigation problem



(b) MDP representation

State space $\mathcal{S}$:

- The definition of a state is recursive: A state is a representation of all the relevant information for predicting future states, in addition to all the information relevant for the related decision problem.
- In the example of robot navigation, the state space $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$ corresponds to the set of the robot's locations on the grid.
- The state space may be finite, countably infinite, or continuous. We will focus on models with a finite set of states. In our example, the states correspond to different positions on a discretized grid.

Action space $\mathcal{A}$:

- The states of the system are modified by the actions executed by an agent.
- The goal of the agent is to choose actions that will influence the future of the system so that the more desirable states will occur more frequently.
- The actions space can be finite, infinite or continuous, but we will consider only the finite case.
- In our example, the actions of the robot might be move north, move south, move east, move west, or do not move, so $\mathcal{A} = \{N, S, E, W, nothing\}$.
- In the MDP framework, the state changes only after an agent executes an action.

Transition function T:

- When an agent tries to execute an action in a given state, the action does not always lead to the same result, this is due to the fact that the information represented by the state is not sufficient for determining precisely the outcome of the actions.
- $T(s_t, a_t, s_{t+1})$ returns the probability of transitioning to state s_{t+1} after executing action a_t in state s_t.

$$T(s_t, a_t, s_{t+1}) = P(s_{t+1} \mid s_t, a_t)$$

- In our example, the actions can be either deterministic, or stochastic if the floor is slippery, and the robot might ends up in a different position while trying to move toward another one.

Reward function R:

- The preferences of the agent are defined by the reward function R.
- This function directs the agent towards desirable states and keeps it away from unwanted ones. $R(s_t, a_t)$ returns a reward (or a penalty) to the agent for executing action a_t in state s_t.
- The goal of the agent is then to choose actions that maximize its cumulated reward.
- The elegance of the MDP framework comes from the possibility of modeling complex concurrent tasks by simply assigning rewards to the states.
- In our example, one may consider a reward of $+100$ for reaching the goal state, a -2 for any movement (consumption of energy), and a -1 for not doing anything (waste of time).

Finite or Infinite Horizons, Discount Factors

Given a reward function, the goal of the agent is to maximize the expected cumulated reward over some number H of steps, called the *horizon*.

- The horizon can be either finite or infinite.
- If the horizon is finite, then the optimal actions of the agent will depend not only on the states, but also on the remaining number of steps until the end.
- In our example, if the agent is at the top right position on the grid, and only two steps are left before the end, then it would better to do nothing and receive an average reward of -1 per step, than to move and receive an average reward of -2 per step, since the goal cannot be reached within two steps.

Finite or Infinite Horizons, Discount Factors

Given a reward function, the goal of the agent is to maximize the expected cumulated reward over some number H of steps, called the *horizon*.

- If the horizon is infinite ($H = \infty$), then the optimal actions depend only on the state. In our case, the optimal action at any step is to move toward the goal.
- A *discount factor* $\gamma \in [0, 1)$ is also used to indicate how the importance of the earned rewards decreases for every time-step delay. A reward that will be received k time-steps later is scaled down by a factor of γ^k.
- The discount factor can also be interpreted as the probability that the process continues after any step.

Policies and Value Functions

- The agent selects its actions according to a *policy* (or a *strategy*).
- A deterministic stationary policy π is a function that maps every state s into an action a.

$$\pi : \mathcal{S} \rightarrow \mathcal{A}.$$

- A stochastic stationary policy is a function that maps each state into a probability distribution over the different possible actions.
- We consider only deterministic policies here.
- The value function of a policy π is a function V^π that associates to each state the sum of expected rewards that the agent will receive if it starts executing policy π from that state. In other terms:

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t} \left[R(s_t, \pi(s_t)) | \pi, s_0 = s \right]$$

where $\pi(s_t)$ is the action chosen in state s.

Bellman Equation

The value function of a stationary policy can also be recursively defined as:

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s\right]$$

$$= R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s\right]$$

$$= R(s, \pi(s)) + \gamma \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 \sim T(s, \pi(s), .)\right]$$

(the next state is the new s_0, distributed as $T(s, \pi(s), .)$)

$$= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s'\right]$$

$$= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^{\pi}(s')$$

Bellman Equation

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^{\pi}(s')$$

$$\text{value} = \text{immediate reward} + \gamma(\text{expected value of next state})$$

This equation plays a central role in **dynamic programming**, a family of methods for solving a complex problem by breaking it down into a collection of simpler subproblems.

In dynamic programming, invented by Richard Bellman in 1957, sub-problems are nested recursively inside larger problems.

Richard Bellman
(1920-1984)

Bellman Equation

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^{\pi}(s')$$

Optimal policies

- An optimal policy π^* is one that satisfies:

$$\forall s \in \mathcal{S} : \pi^* \in \arg\max_{\pi} V^{\pi}(s)$$

- The value function of an optimal policy is called **the optimal value function**, it is defined as:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V^*(s') \right]$$

Optimal policies

- In his seminal work on dynamic programming, Richard Bellman proved that a stationary deterministic optimal policy exists for any discounted infinite horizon MDP.
- If the value function V^π of a given policy π satisfies

$$V^\pi(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \right],$$

  then $V^\pi = V^*$ and π is an optimal policy.
- The equation above is a necessary and sufficient optimality condition.

Planning with Markov Decision Processes

Planning: finding an optimal policy π^* given an MDP $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$.

Most of planning algorithms for MDPs fall in one of the two categories:

- Policy iteration
- Value iteration

Policy Iteration

- Start with a randomly chosen policy π_t at $t = 0$
- Alternate between the **policy evaluation** and the **policy improvement** operations until convergence.

Policy evaluation

- Randomly initialize the value function V_k, for $k = 0$.
- Repeat the operation:

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$ for a predefined error threshold ϵ.

Policy Iteration

- Start with a randomly chosen policy π_t at $t = 0$
- Alternate between the **policy evaluation** and the **policy improvement** operations until convergence.

Policy improvement

Find a *greedy* policy π_{t+1} given the value function V_k (computed in the policy evaluation phase):

$$\forall s \in \mathcal{S} : \pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

The policy iteration process stops when $\pi_t = \pi_{t-1}$, in which case π_t is an optimal policy, i.e. $\pi_t = \pi^*$.

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;

`/* Initialization */ ;`

$t = 0$, $k = 0$;

$\forall s \in \mathcal{S}$: Initialize $\pi_t(s)$ with an arbitrary action;

$\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;

repeat

`/* Policy evaluation */;`

repeat

$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$;

$k \leftarrow k + 1$;

until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;

`/* Policy improvement */;`

$\forall s \in \mathcal{S} : \pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$;

$t \leftarrow t + 1$;

until $\pi_t = \pi_{t-1}$;

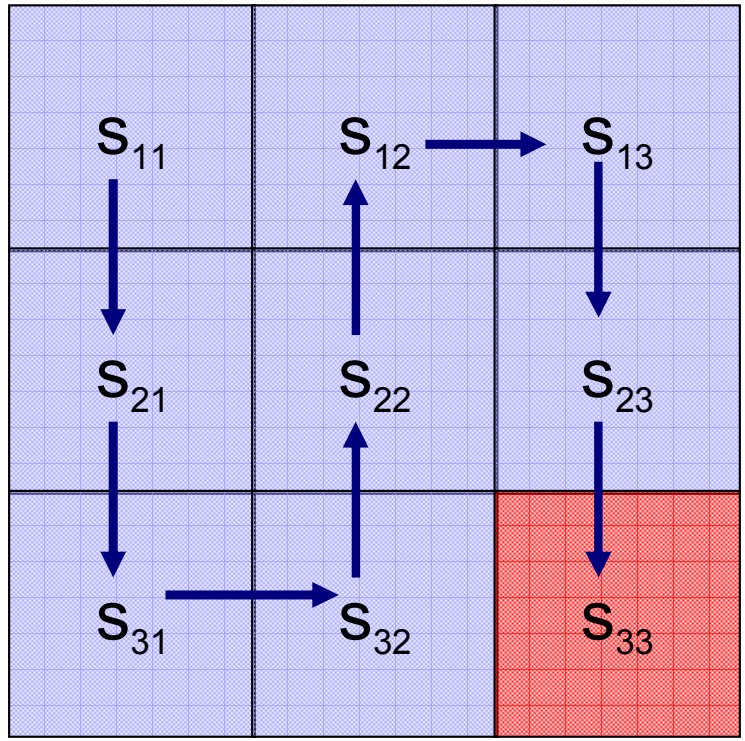
$\pi^* = \pi_t$;

Output: An optimal policy π^*;

Algorithm 1: The policy iteration algorithm.

Example

- State space $\mathcal{S} = \{s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, s_{31}, s_{32}, s_{33}\}$
- Action space $\mathcal{A} = \{\leftarrow, \rightarrow, \uparrow, \downarrow, \text{do nothing}\}$
- Deterministic transition function
- Reward function $\forall a : R(s_{33}, a) = 1$, $\forall a, \forall s \neq s_{33} : R(s, a) = 0$
- Discount factor $\gamma = 0.9$.

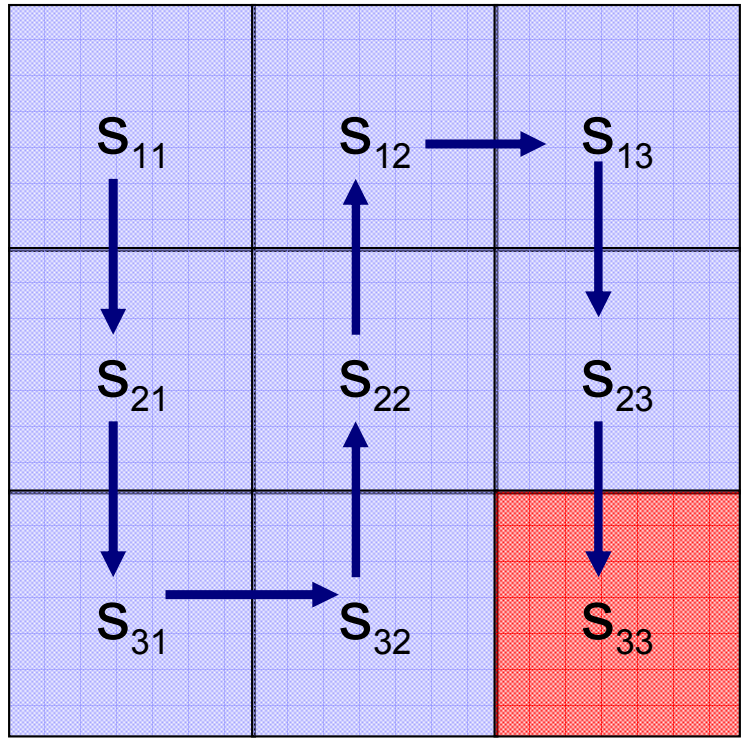


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0
s_{11}	0
s_{12}	0
s_{13}	0
s_{21}	0
s_{22}	0
s_{23}	0
s_{31}	0
s_{32}	0
s_{33}	0

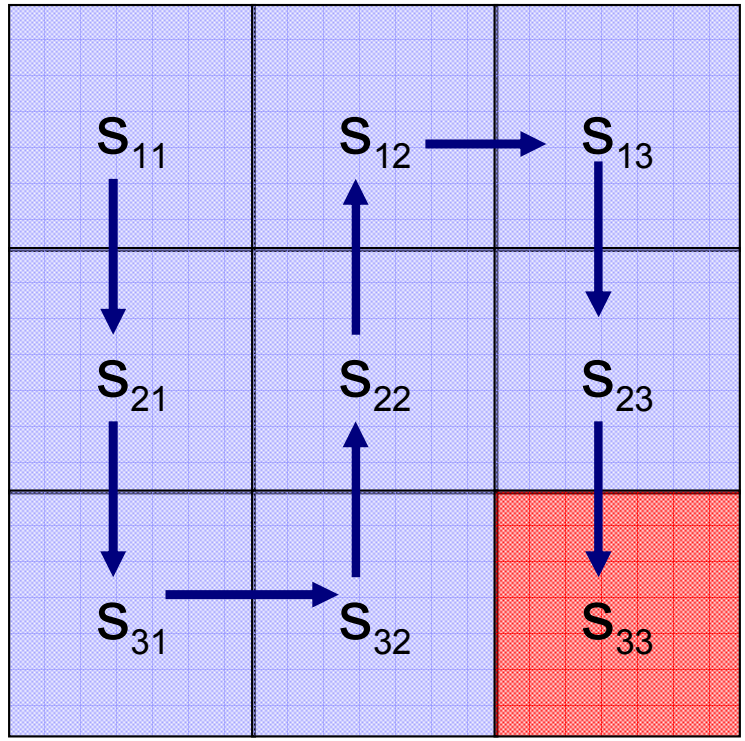


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1
s_{11}	0	$0 + 0.9 \times 0$
s_{12}	0	$0 + 0.9 \times 0$
s_{13}	0	$0 + 0.9 \times 0$
s_{21}	0	$0 + 0.9 \times 0$
s_{22}	0	$0 + 0.9 \times 0$
s_{23}	0	$0 + 0.9 \times 0$
s_{31}	0	$0 + 0.9 \times 0$
s_{32}	0	$0 + 0.9 \times 0$
s_{33}	0	$1 + 0.9 \times 0$

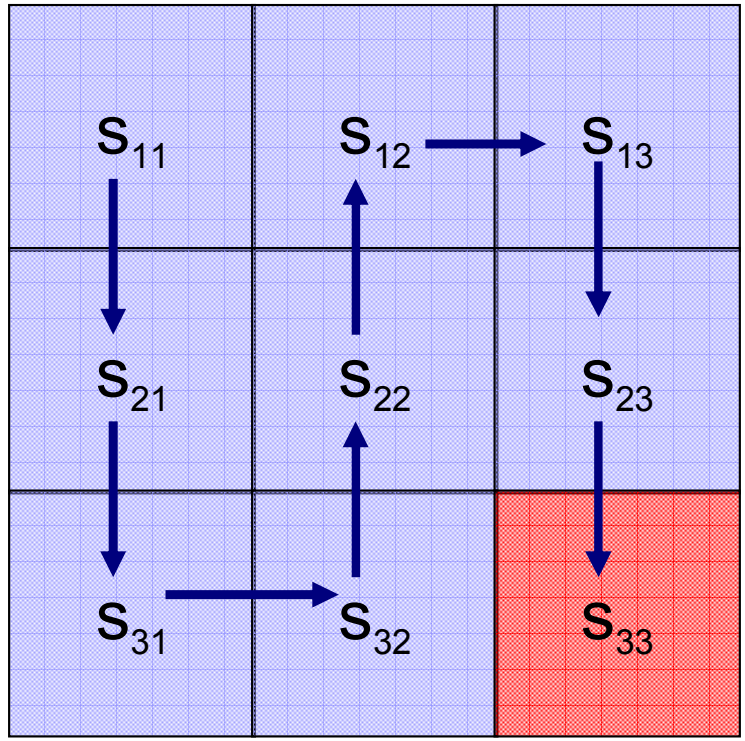


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1
s_{11}	0	0
s_{12}	0	0
s_{13}	0	0
s_{21}	0	0
s_{22}	0	0
s_{23}	0	0
s_{31}	0	0
s_{32}	0	0
s_{33}	0	1

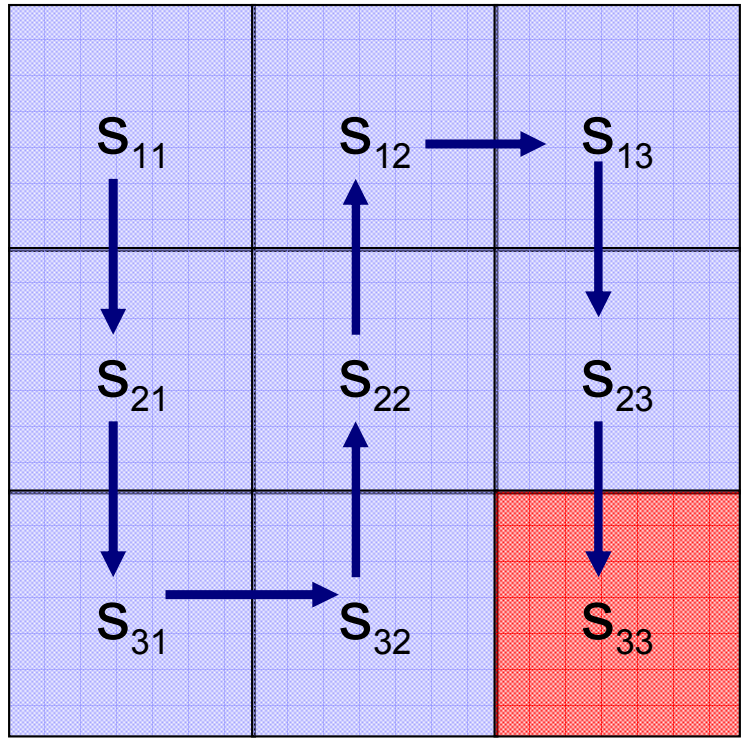


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2
s_{11}	0	0	$0 + 0.9 \times 0$
s_{12}	0	0	$0 + 0.9 \times 0$
s_{13}	0	0	$0 + 0.9 \times 0$
s_{21}	0	0	$0 + 0.9 \times 0$
s_{22}	0	0	$0 + 0.9 \times 0$
s_{23}	0	0	$0 + 0.9 \times 1$
s_{31}	0	0	$0 + 0.9 \times 0$
s_{32}	0	0	$0 + 0.9 \times 0$
s_{33}	0	1	$1 + 0.9 \times 1$

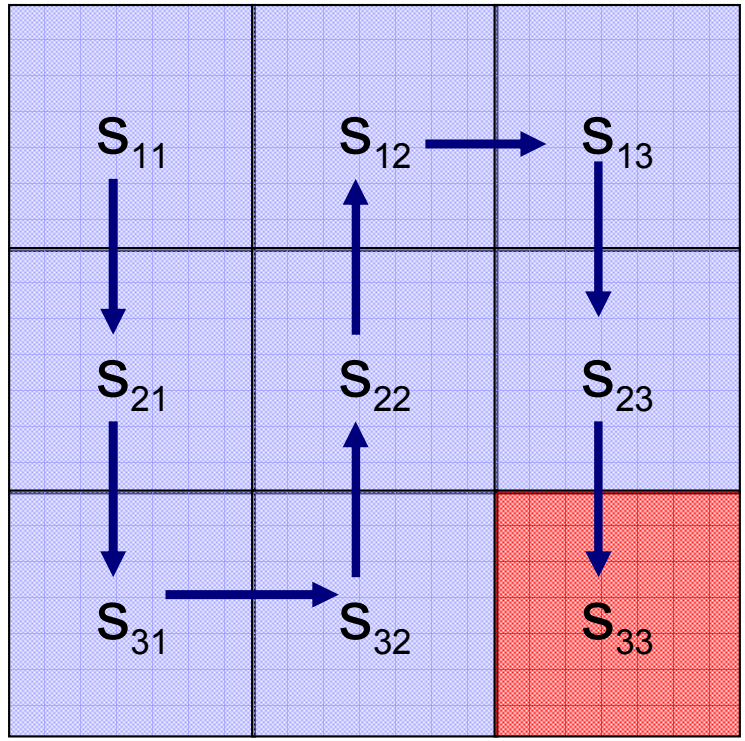


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2
s_{11}	0	0	0
s_{12}	0	0	0
s_{13}	0	0	0
s_{21}	0	0	0
s_{22}	0	0	0
s_{23}	0	0	0.9
s_{31}	0	0	0
s_{32}	0	0	0
s_{33}	0	1	1.9

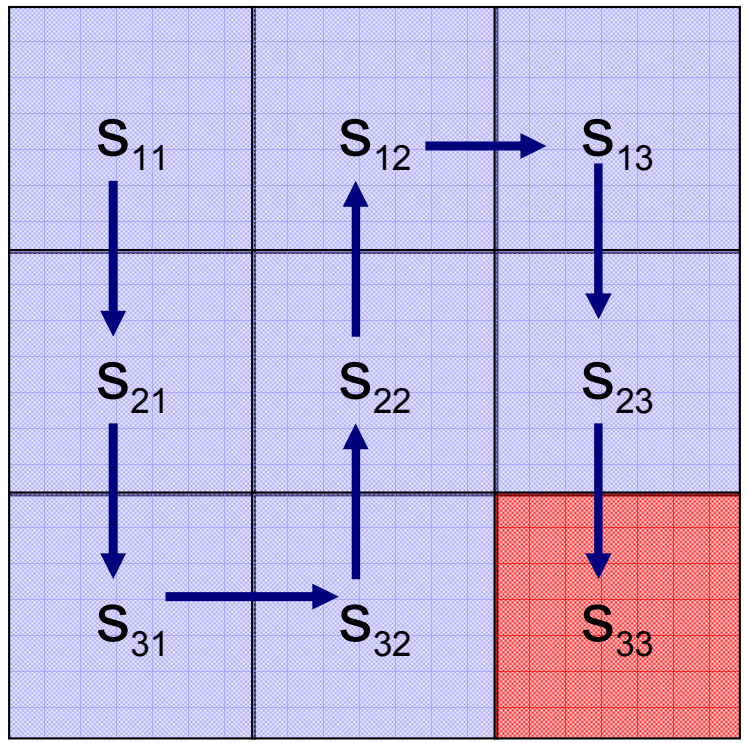


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3
s_{11}	0	0	0	$0 + 0.9 \times 0$
s_{12}	0	0	0	$0 + 0.9 \times 0$
s_{13}	0	0	0	$0 + 0.9 \times 0.9$
s_{21}	0	0	0	$0 + 0.9 \times 0$
s_{22}	0	0	0	$0 + 0.9 \times 0$
s_{23}	0	0	0.9	$0 + 0.9 \times 1.9$
s_{31}	0	0	0	$0 + 0.9 \times 0$
s_{32}	0	0	0	$0 + 0.9 \times 0$
s_{33}	0	1	1.9	$1 + 0.9 \times 1.9$

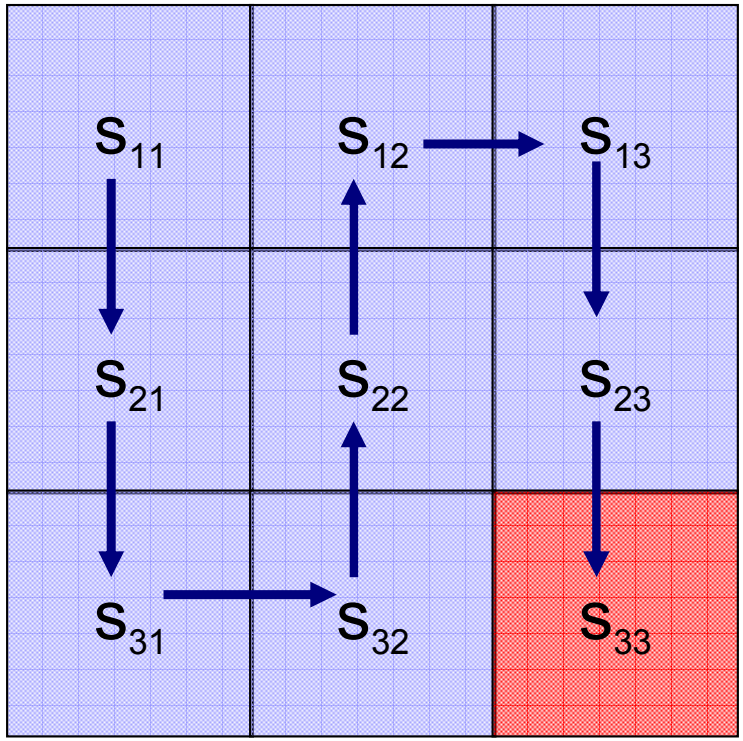


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3
s_{11}	0	0	0	0
s_{12}	0	0	0	0
s_{13}	0	0	0	0.81
s_{21}	0	0	0	0
s_{22}	0	0	0	0
s_{23}	0	0	0.9	1.71
s_{31}	0	0	0	0
s_{32}	0	0	0	0
s_{33}	0	1	1.9	2.71

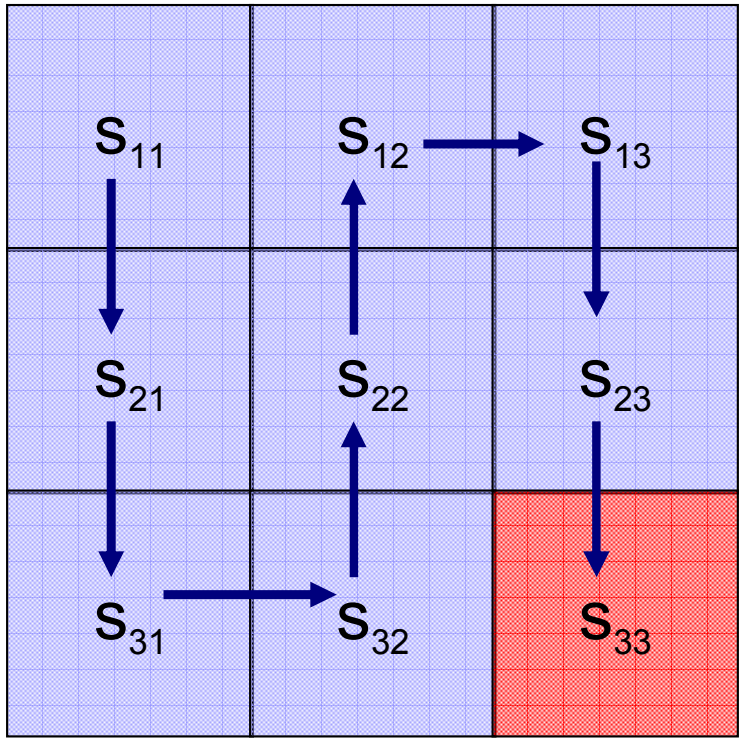


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3	$\dots$	V_{1000}
s_{11}	0	0	0	0		4.3
s_{12}	0	0	0	0		7.3
s_{13}	0	0	0	0.81		8.1
s_{21}	0	0	0	0		4.8
s_{22}	0	0	0	0		6.6
s_{23}	0	0	0.9	1.71		9
s_{31}	0	0	0	0		5.3
s_{32}	0	0	0	0		5.9
s_{33}	0	1	1.9	2.71		10

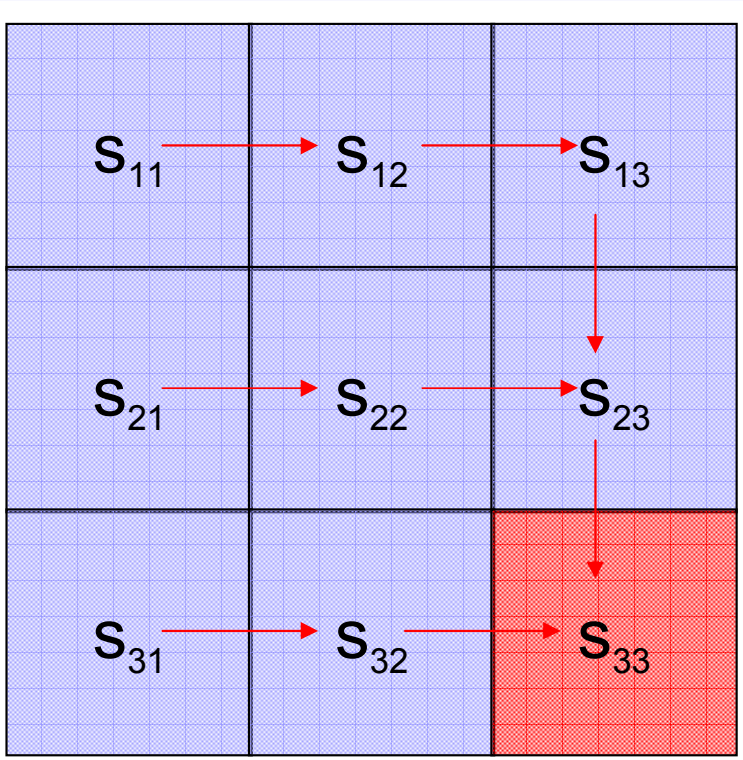


Initial policy

Example

Now, we improve the previous policy based on the calculated values

State	V_0	V_1	V_2	V_3	$\dots$	V_{1000}
s_{11}	0	0	0	0		4.3
s_{12}	0	0	0	0		7.3
s_{13}	0	0	0	0.81		8.1
s_{21}	0	0	0	0		4.8
s_{22}	0	0	0	0		6.6
s_{23}	0	0	0.9	1.71		9
s_{31}	0	0	0	0		5.3
s_{32}	0	0	0	0		5.9
s_{33}	0	1	1.9	2.71		10



Improved policy

Value Iteration

Value iteration can be written as a simple backup operation:

$$\forall s \in \mathcal{S} : V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

This operation is repeated until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$, in which case the optimal policy is simply the greedy policy with respect to the value function V_k

$$\forall s \in \mathcal{S} : \pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

Value Iteration

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;
$k = 0$;
$\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;
repeat

$$\forall s \in \mathcal{S} : V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right];$$

$k \leftarrow k + 1$;
until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;

$$\forall s \in \mathcal{S} : \pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right];$$

Output: An optimal policy π^*;

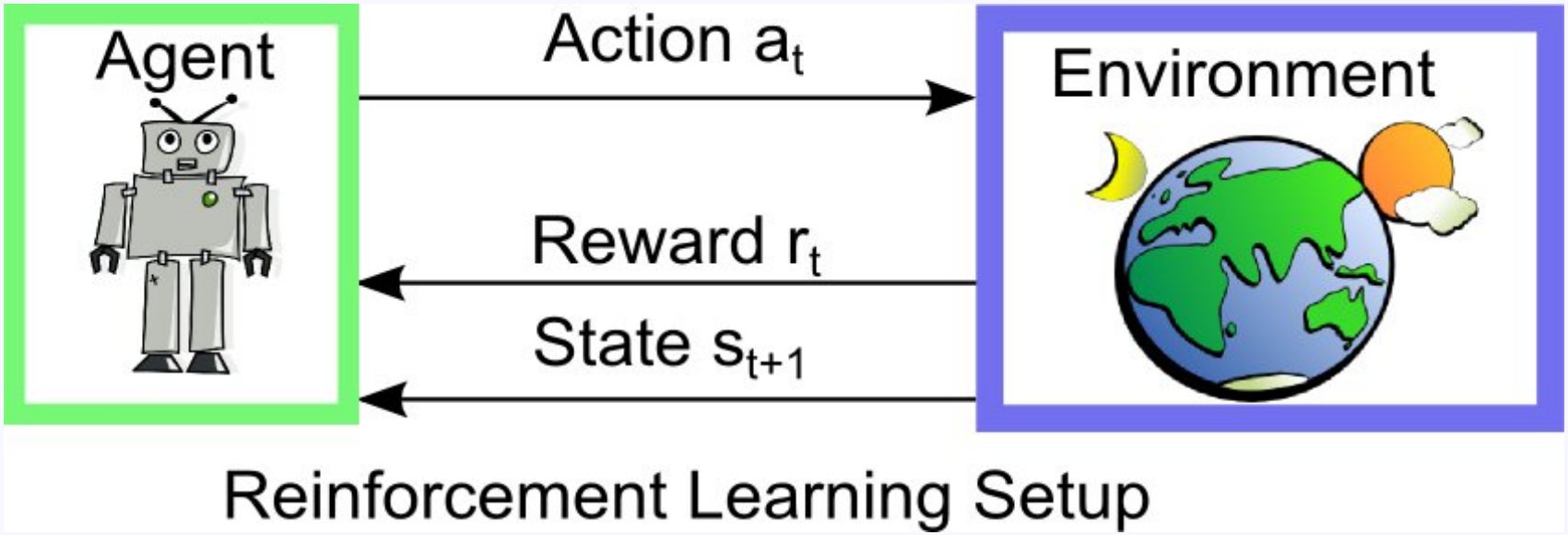
Algorithm 2: The value iteration algorithm.

How can we find an optimal policy when we do not know the transition function T?

Reinforcement Learning

- Learning with MDPs generally refers to the problem of finding an optimal policy π^* for an MDP with unknown transition function T.
- The agent learns the best actions from experience, by acting and observing the received rewards.
- This problem is better known under the name of **Reinforcement Learning** (a.k.a *trial-and-error* learning).

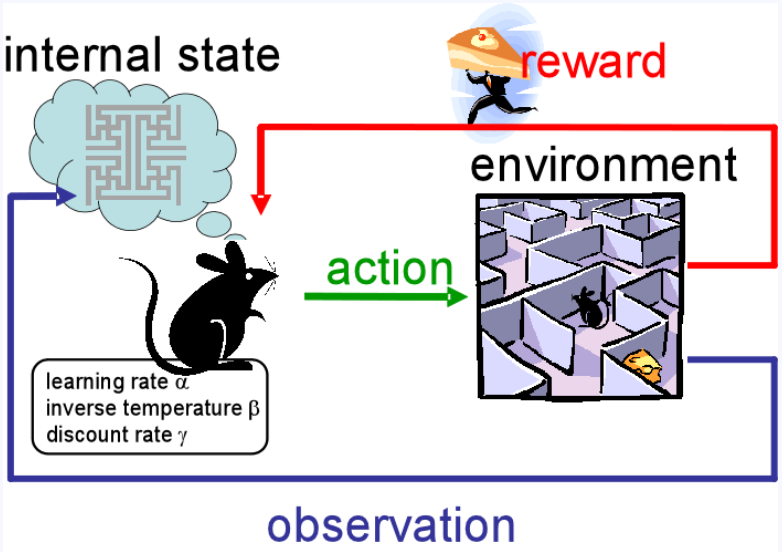
Reinforcement Learning



http://www.ausy.tu-darmstadt.de/Research/Research

Reinforcement learning in behavioral psychology

- Reinforcement: strengthening an organism's future behavior whenever that behavior is preceded by a specific antecedent stimulus.
- Behaviour will likely be repeated if it has been reinforced (with positive rewards).
- Behaviour will likely not be repeated if it has been punished (with negative rewards).



http://www.cs.utexas.edu/ eladlieb/RLRG.html

Before presenting some learning algorithms, we will first need to introduce the *Q-value function*.

Q-value

A *Q-value* is the expected sum of rewards that an agent will receive if it executes action a in state s then follows a policy π for the remaining steps.

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s')$$

a can be any action, it is not necessarily $\pi(s)$.

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, R \rangle$ with unknown transition function;
$t = 0$, s_0 is an initial state;
$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$: Initialize $Q^t(s, a)$ with an arbitrary value;
repeat
$\pi(s_t) = \arg\max_{a \in \mathcal{A}} Q^t(s_t, a)$;
Choose action a_t as $\pi(s_t)$ with probability $1 - \epsilon_t$ (for exploitation), and as a random action (for exploration) with probability ϵ_t;
Execute action a_t and observe the received reward $R(s_t, a_t)$ and the next state s_{t+1};
$Q^{t+1}(s_t, a_t) =$
$$(1 - \alpha_t) Q^t(s_t, a_t) + \alpha_t \left[R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q^t(s_{t+1}, a') \right] ;$$
$t \leftarrow t + 1$;
until *the end of learning* ;
Output: A learned policy π;

Algorithm 3: The Q-learning algorithm.