

# Probabilistic Uncertainty

CS 121

Lecture 10

July 27, 2009

Chris Archibald

# Announcements

- Homework 4 due today! (the paper)
- Homework 5 due in one week
- Midterm will be returned today. We will go over it at the end of class today
- Homework 6 (final homework!) released on Wednesday

# A brief probability review

- Set of events  $\Omega$
- Set of measurable events  $S$
- Event space must satisfy following:
  - Contains *empty event*  $\emptyset$  and *trivial event*  $\Omega$
  - Closed under union:  
If  $a, b$  in  $S$ , then so is  $a \cup b$
  - Closed under complementation:  
If  $a$  in  $S$ , then  $\Omega - a$  is too

# A brief probability review

- A probability distribution  $P$  over  $(\Omega, S)$  is a mapping from events in  $S$  to real values s.t.
  - $P(a) \geq 0$  for all  $a$  in  $S$
  - $P(\Omega) = 1$
  - If  $a, b$  in  $S$  and  $(a \cap b) = \emptyset$ , then
$$P(a \cup b) = P(a) + P(b)$$

# Interpretations of probability

- Frequentist view: how often repeated event would occur if situation were repeated exactly
  - But what about one time events?
- Subjectivist view: degrees of belief
  - What does it mean to hold a particular degree of belief?
  - Laws of probability = rational believer

# Chain rule and Bayes rule

- Conditional Probability

$$P(b \mid a) = P(a \cap b) / P(a)$$

- Chain rule

$$P(a \cap b \cap c \dots z) = P(a)P(b \mid a) \dots P(z \mid a, b, c, \dots)$$

- Bayes Rule

$$P(a \mid b) = P(b \mid a)P(a) / P(b)$$

# Independence

- Events  $a$  and  $b$  are *independent* if

$$P(a \mid b) = P(a)$$

*or writing a different way, if*

$$P(a \cap b) = P(a)P(b)$$

- Events  $a$  and  $b$  are *conditionally independent* given  $c$  if

$$P(a \mid b, c) = P(a \mid c)$$

# Joint probability distribution

- For a set of random variables (a, b, c . . . ), we can encode the *joint distribution* over these random variables as

$$P(a,b,c,\dots,z)$$

- Should satisfy

$$P(a) = \sum_{b,c,\dots,z} P(a,b,c,\dots,z)$$

- This is called *marginalization*

# Making Decisions under Probabilistic Uncertainty

(Where an agent optimizes what it  
gets on average,  
but it may get more ... or less )

R&N: Chap. 17, Sect. 17.1-5

# General Framework

- An agent operates in some given finite state space
- **No goal state**; instead, states provide **rewards** (positive, negative, or null) that quantify in a single unit system what the agent gets when it visits this state (e.g., a bag of gold, a sunny afternoon on the beach, a speeding ticket, etc...)
- Each action has several possible outcomes, each with some probability; sensing may also be imperfect
- The agent's goal is to plan a strategy (here, it is called a **policy**) to maximize the **expected** amount of rewards collected

# Two Cases

- Uncertainty in action only  
[The world is fully observable]
- Uncertainty in both action and sensing  
[The world is partially observable]

# Action Model

Action  $a$ :

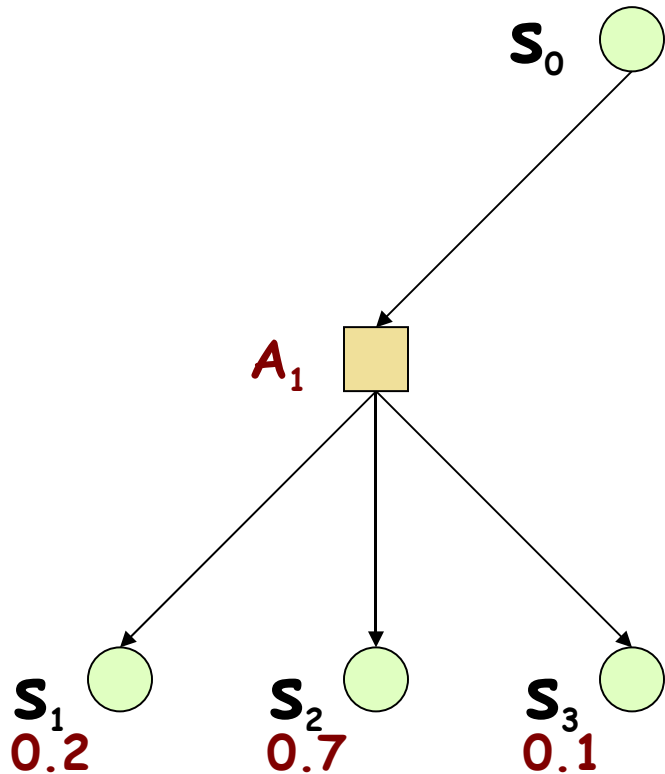
$$s \in S \rightarrow a(s) = \{s_1 (p_1), s_2 (p_2), \dots, s_n (p_n)\}$$

probabilistic distribution  
of possible successor states

$$\left[ \sum_{i=1}^n p_i = 1 \right]$$

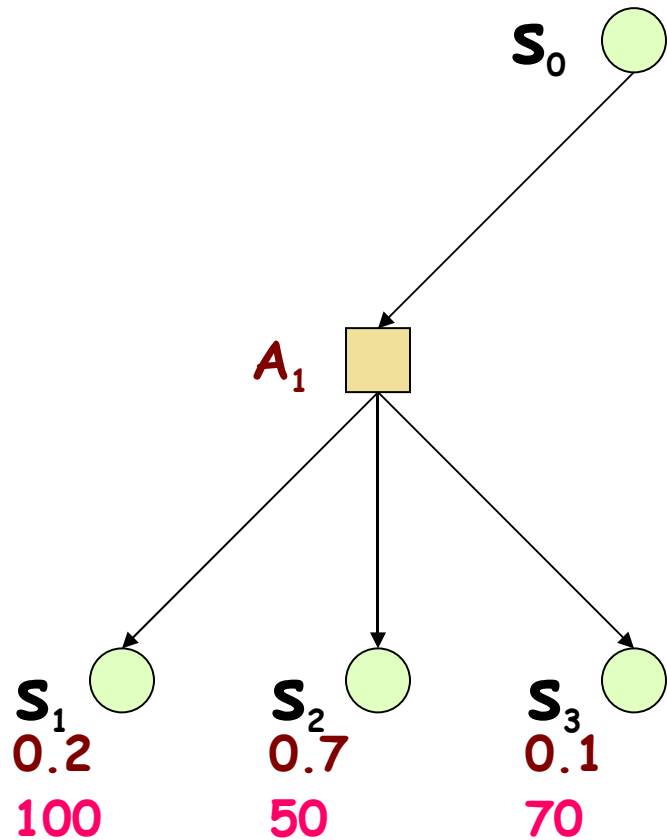
**Markov assumption:** The action model  $a(s)$  does not depend on what happened prior to reaching  $s$

# Starting very simple ...



- $S_0$  describes many actual states of the real world.  $A_1$  reaches  $s_1$  in some states,  $s_2$  in others, and  $s_3$  in the remaining ones
- If the agent could return to  $S_0$  many times in independent ways and if at each time it executed  $A_1$ , then it would reach  $s_1$  20% of the times,  $s_2$  70% of the times, and  $s_3$  10% of the times

# Introducing rewards ...

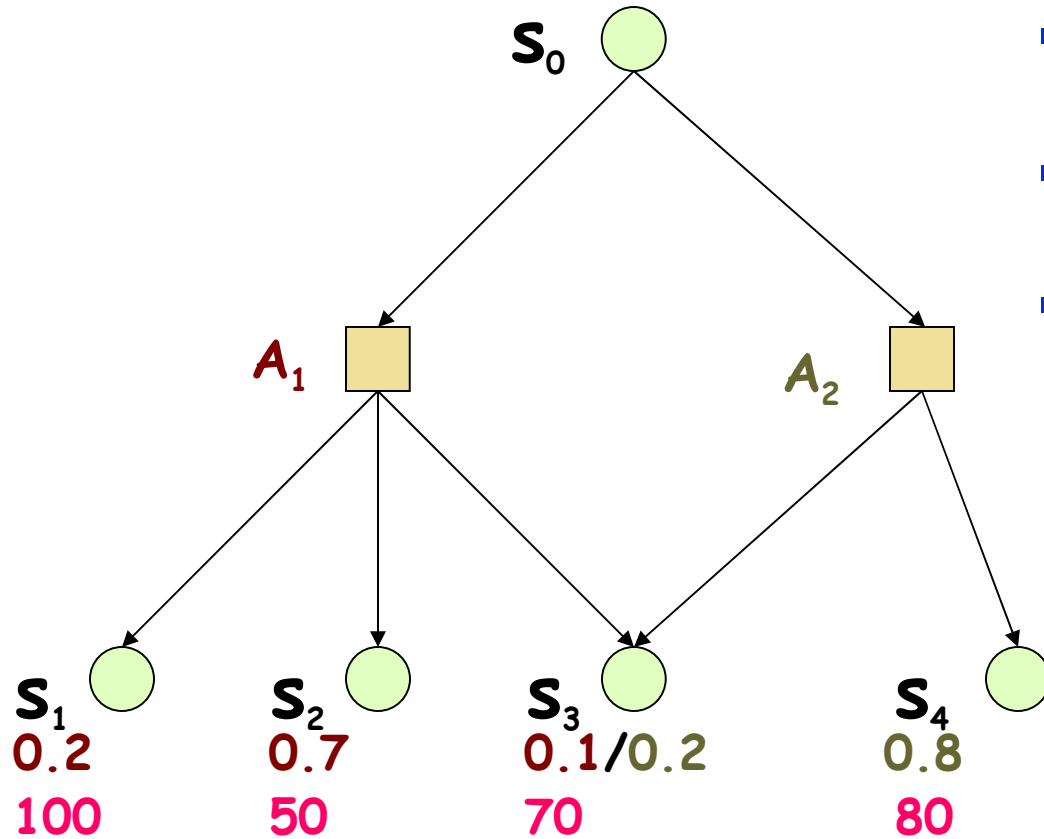


- Assume that the agent receives **rewards** in some states (rewards can be positive or negative)
- If the agent could execute  $A_1$  in  $S_0$  many times, the **average (expected) reward** that it would get is:

$$\begin{aligned}U_1(S_0) &= 100 \times 0.2 + 50 \times 0.7 + 70 \times 0.1 \\ &= 20 + 35 + 7 \\ &= 62\end{aligned}$$

← rewards associated with states  $s_1$ ,  $s_2$ , and  $s_3$

# ... and a second action ...

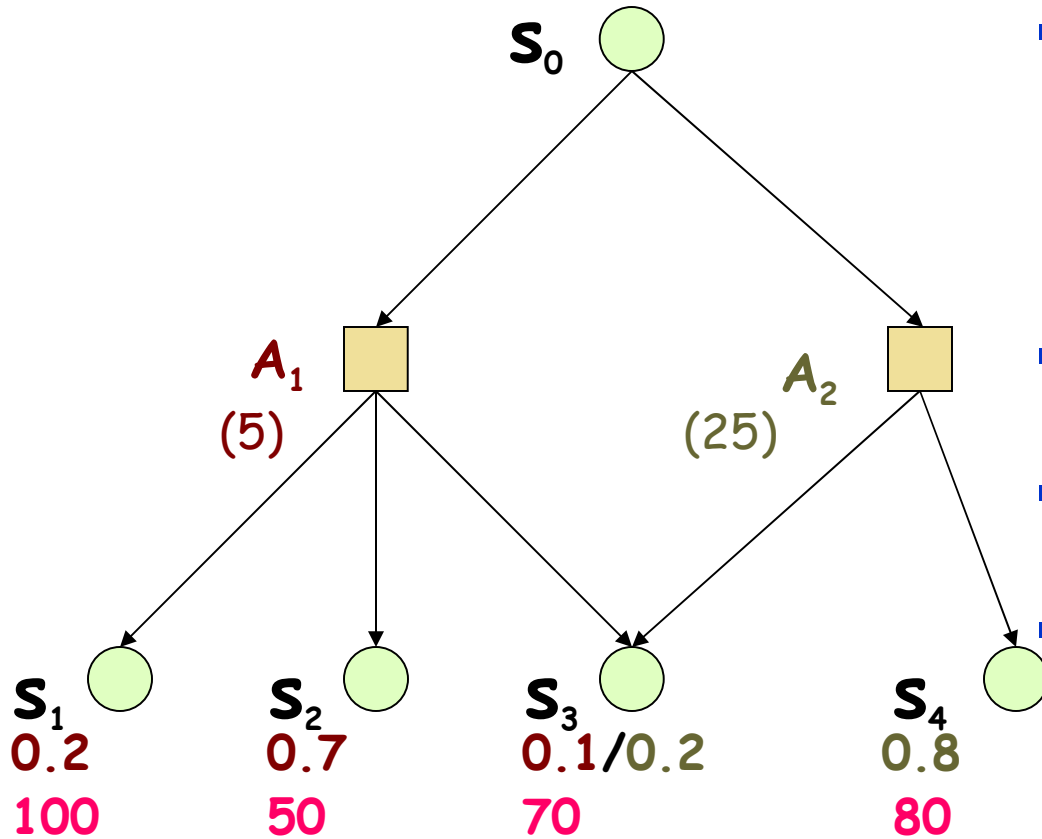


- $U_1(s_0) = 62$

- $U_2(s_0) = 78$

- If the agent chooses to execute  $A_2$ , it will maximize the average collected rewards

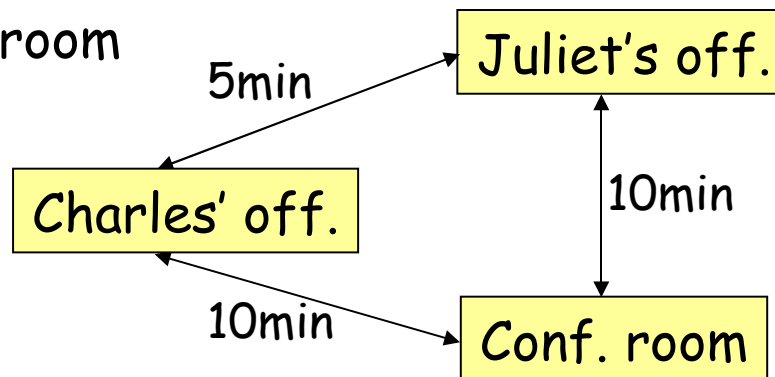
# ... and action costs



- We may associate costs with actions, which are subtracted from the rewards
- $U_1(s_0) = 62 - 5 = 57$
- $U_2(s_0) = 78 - 25 = 53$
- When in  $s_0$ , the agent will now gain more on the average by executing  $A_1$  than by executing  $A_2$

# A complete (but still very simple) example: Finding Juliet

- A robot, Romeo, is in Charles' office and must deliver a letter to Juliet
- Juliet is either in her office, or in the conference room. Each possibility has probability 0.5
- Traveling takes 5 minutes between Charles' and Juliet's office, 10 minutes between Charles' or Juliet's office and the conference room

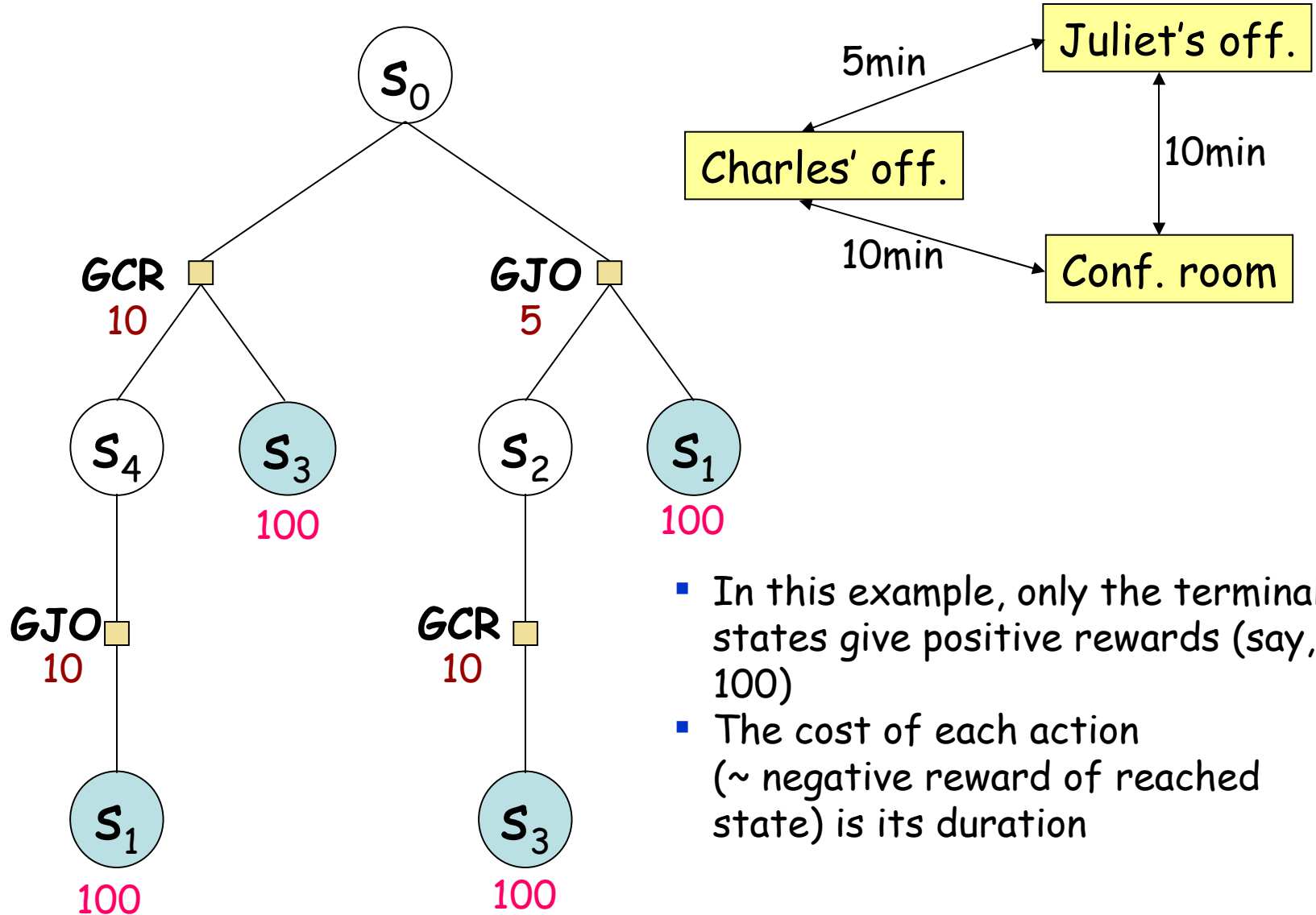


- To perform his duties well and save battery, the robot wants to deliver the letter while minimizing the time spent in transit

# States and Actions in Finding-Juliet Problem

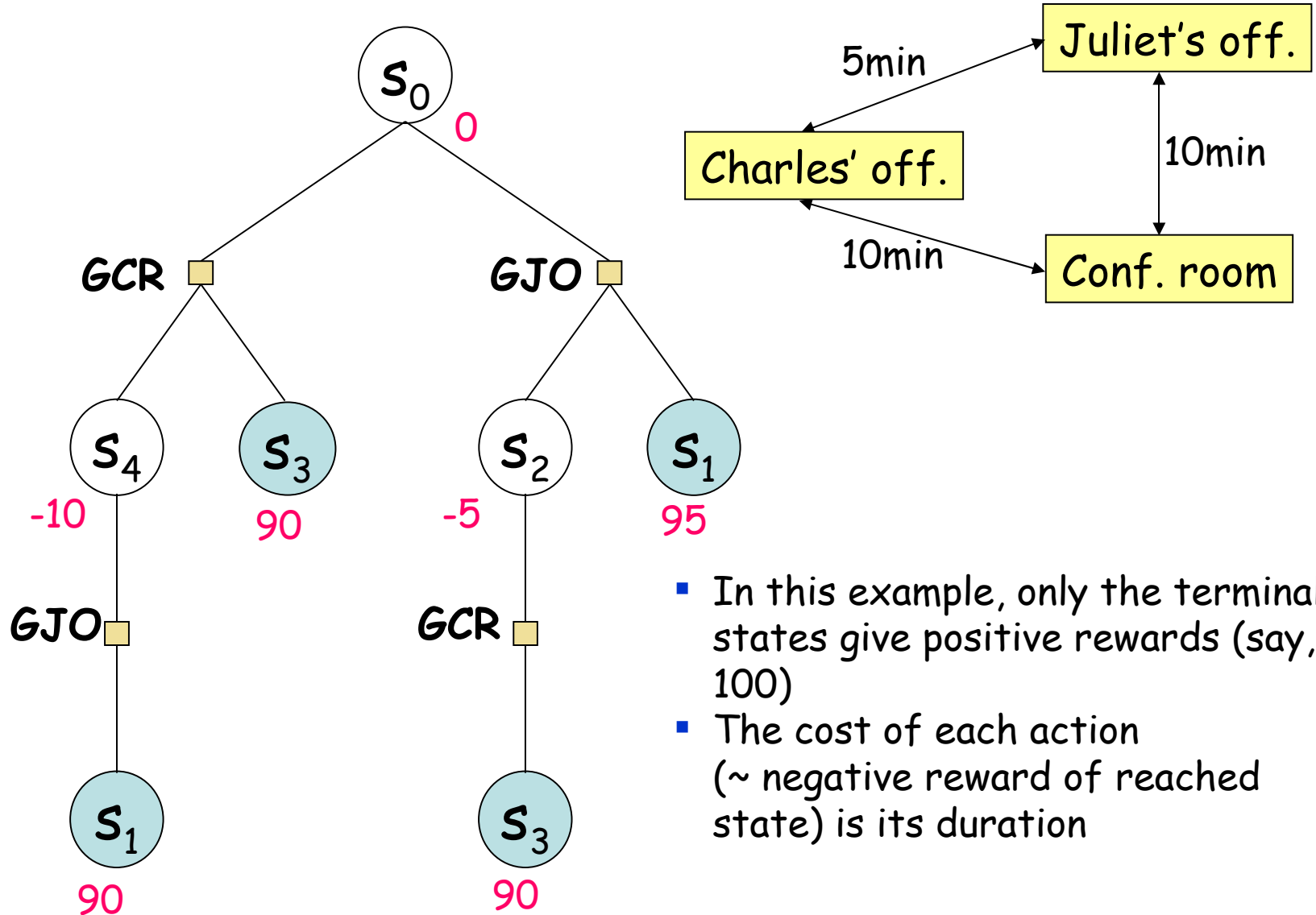
- States:
  - $S_0$ : Romeo in Charles' office
  - $S_1$ : Romeo in Juliet's office and Juliet is there
  - $S_2$ : Romeo in Juliet's office and Juliet is not there
  - $S_3$ : Romeo in conference room and Juliet is there
  - $S_4$ : Romeo in conference room and Juliet is not there
  - In this example,  $S_1$  and  $S_3$  are **terminal** states
- Actions:
  - $GJO$  (go to Juliet's office)
  - $GCR$  (go to conference room)
  - The uncertainty in an action is directly linked to the uncertainty in Juliet's location

# State/Action Tree



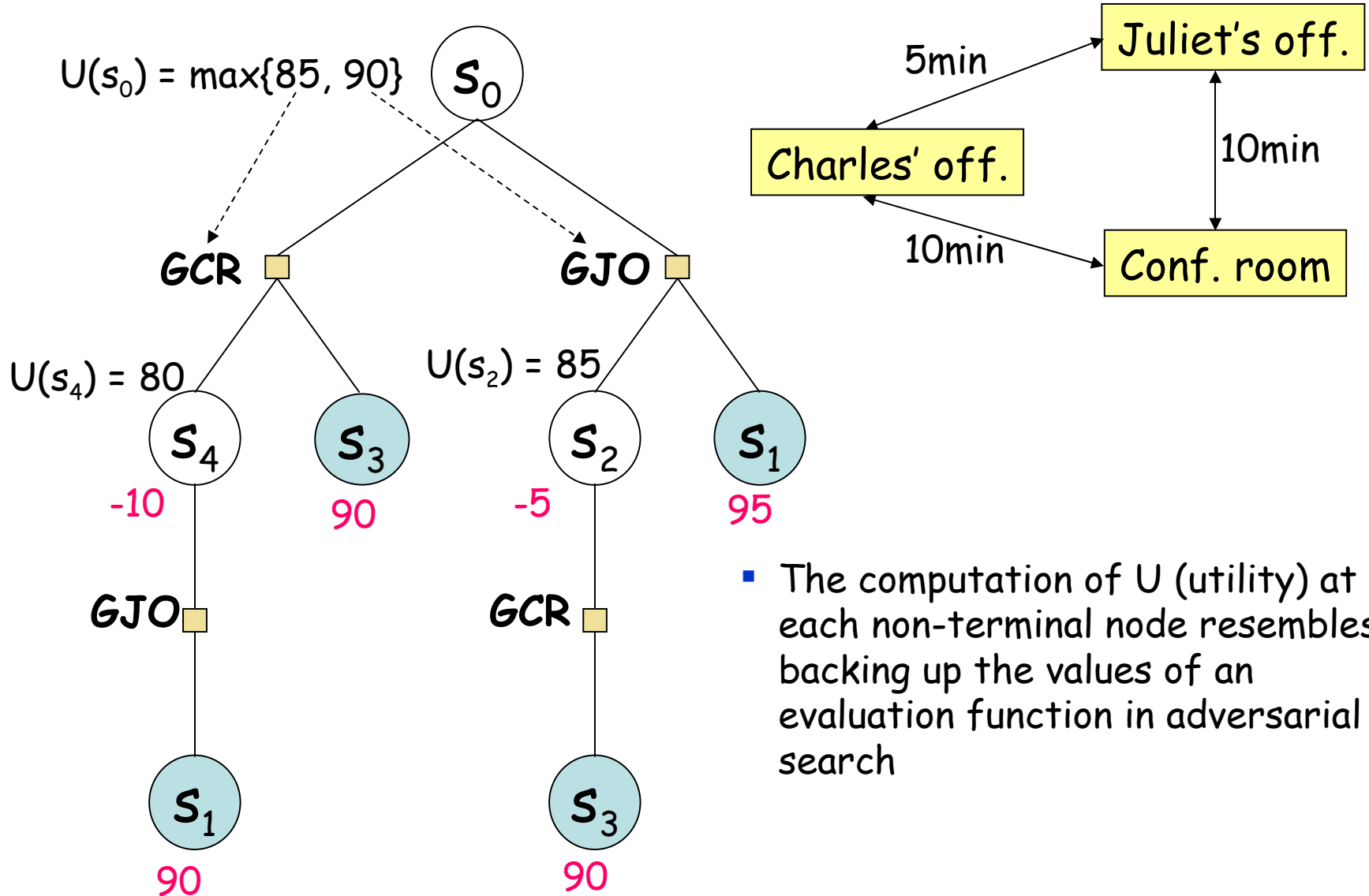
- In this example, only the terminal states give positive rewards (say, 100)
- The cost of each action (~ negative reward of reached state) is its duration

# State/Action Tree

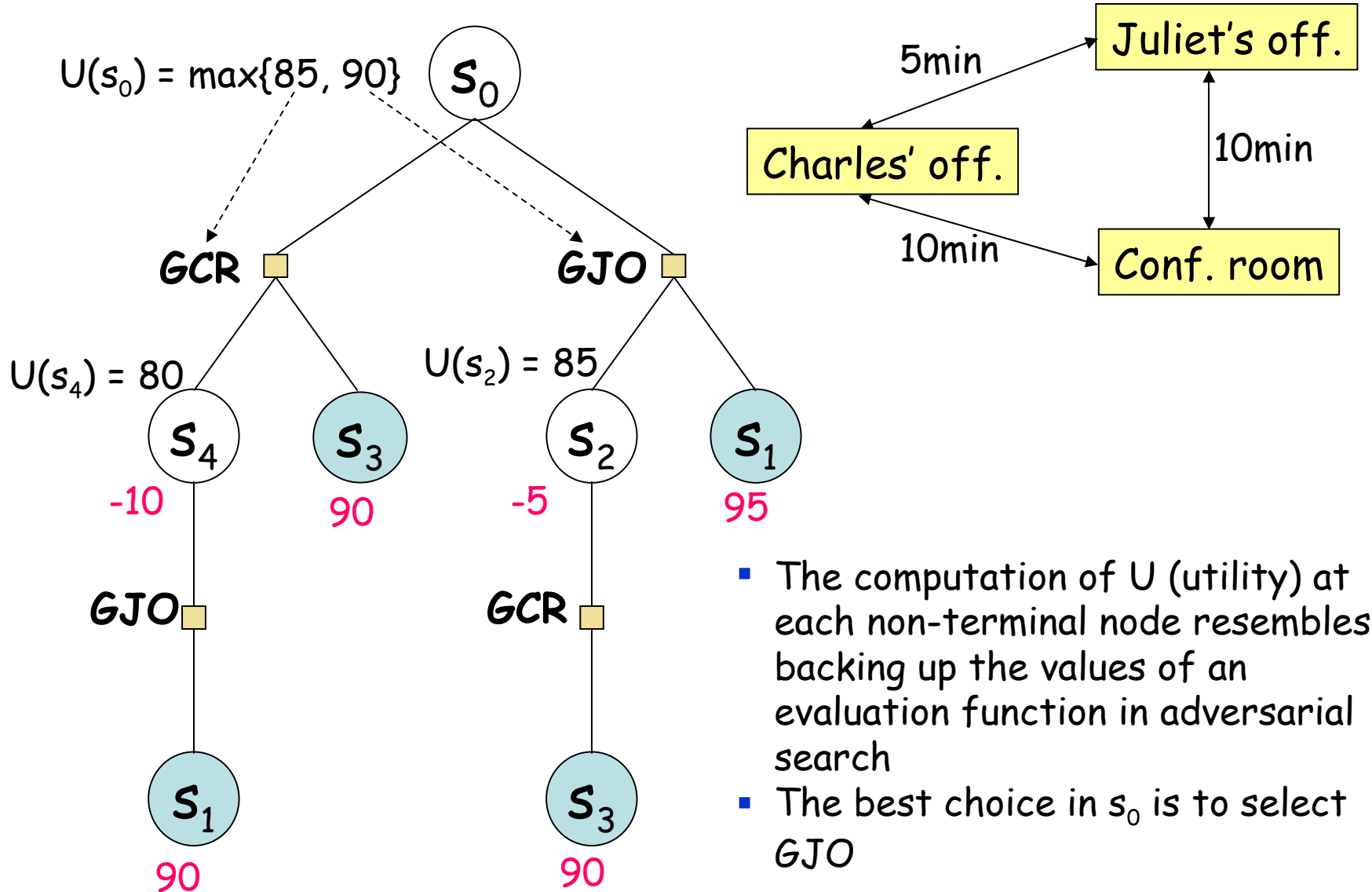


- In this example, only the terminal states give positive rewards (say, 100)
- The cost of each action (~ negative reward of reached state) is its duration

# State/Action Tree



# State/Action Tree



# Generalization: Markov Decision Process (MDP)

- Inputs:
  - Initial state  $s_0$
  - Action model
  - Reward  $R(s)$  collected in each state  $s$
- A state is **terminal** if it has no successor
- Starting at  $s_0$ , the agent keeps executing actions until it reaches a terminal state
- Its goal is to maximize the expected sum of rewards collected (**additive** rewards)
- Assume for a while that the same state can't be reached twice (no cycles)

# Utility of a State

The **utility** of a state  $s$  measures its desirability:

- If  $s$  is terminal:

$$U(s) = R(s)$$

- If  $s$  is non-terminal,

$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')$$

[the reward of  $s$  augmented by the expected sum of rewards collected in future states]

$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$$

$\text{Appl}(s)$  is the set of all actions applicable to state  $s$

$\text{Succ}(s,a)$  is the set of all possible states after applying  $a$  to  $s$

$P(s'|a,s)$  is the probability of being in  $s'$  after executing  $a$  in  $s$

[the reward of  $s$  augmented by the expected sum of rewards collected in future states]

# Utility of a State

The utility of a state  $s$  measures its desirability:

- If  $s$  is terminal:

$$U(s) = R(s)$$

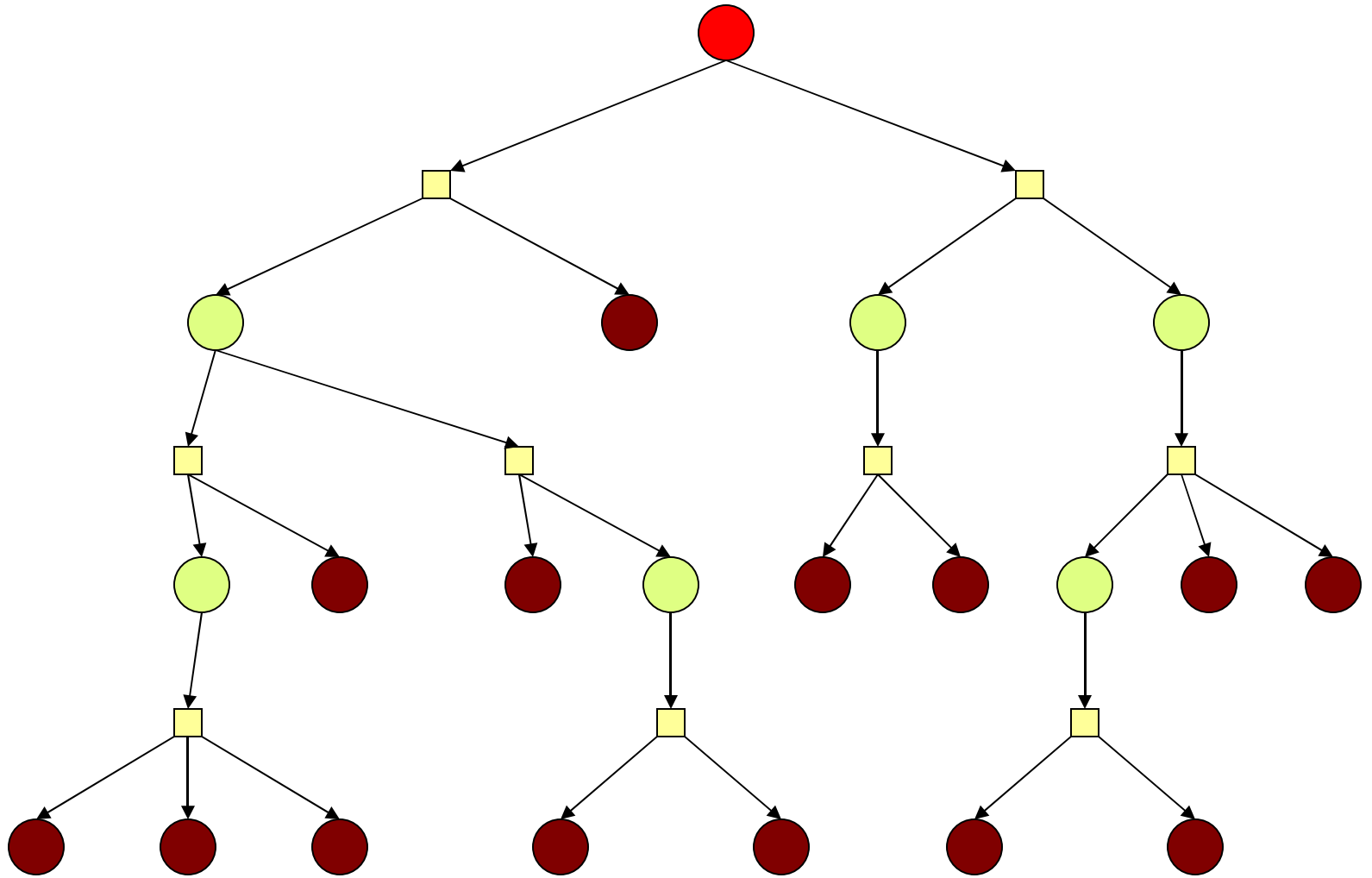
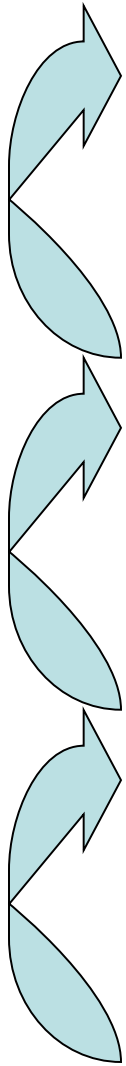
There is no cycle

- If  $s$  is non-terminal,

$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')$$

[the reward of  $s$  augmented by the expected sum of rewards collected in future states]

$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$$



# Utility with Action Costs

$U(s) =$

$$R(s) + \max_{a \in \text{Appl}(s)} [-\text{cost}(a) + \sum_{s' \in \text{Succ}(s,a)} P(s' | a, s) U(s')]$$

# Optimal Policy

- A **policy** is a function that maps each state  $s$  into the action to execute if  $s$  is reached
- The **optimal** policy  $\Pi^*$  is the policy that always leads to maximizing the expected sum of rewards collected in future states  
(Maximum Expected Utility principle)

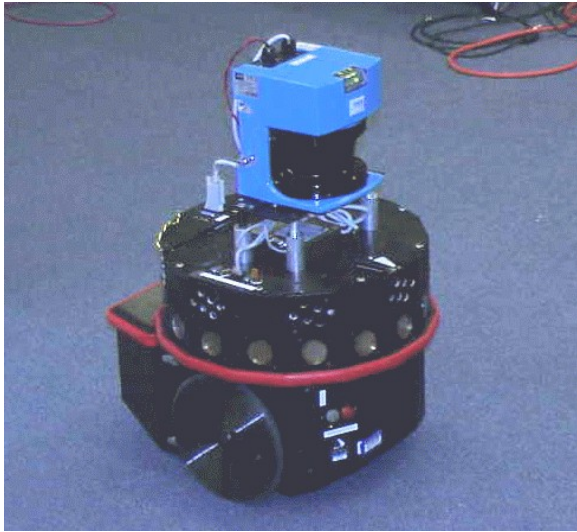
$$\Pi^*(s) = \mathbf{arg} \max_{a \in \text{Appl}(s)} [-\text{cost}(a) + \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')]$$
$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} [-\text{cost}(a) + \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')]$$

# Issues

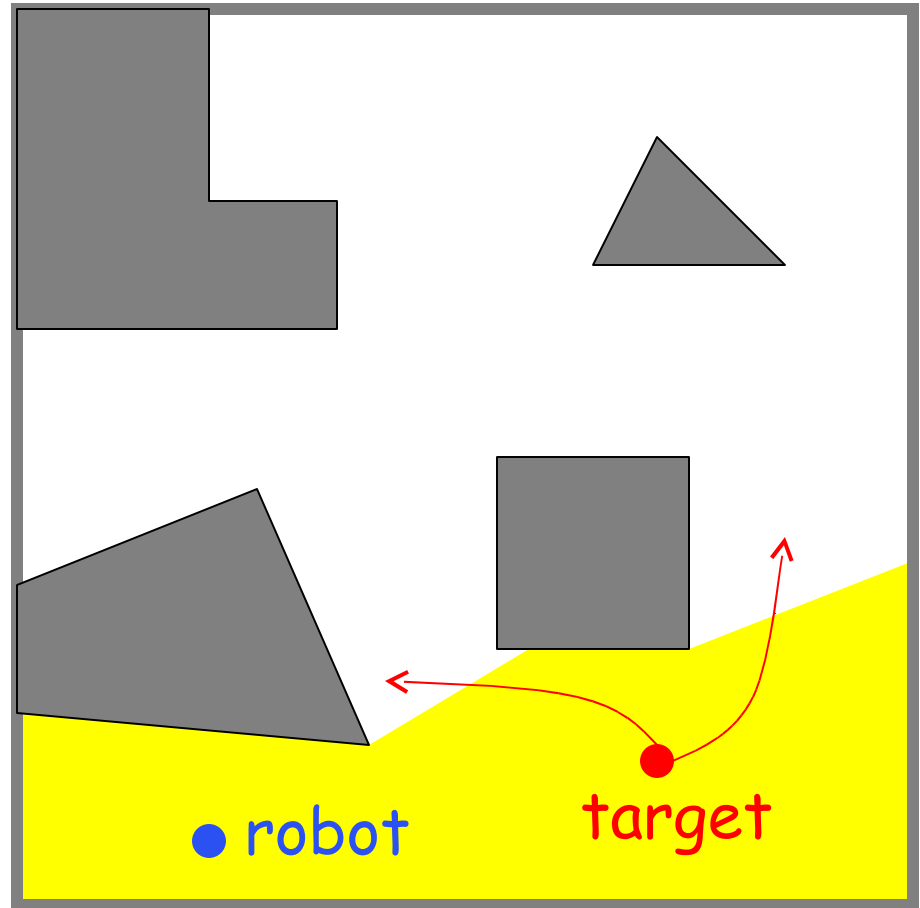
- What if the set of states reachable from the initial state is too large to be entirely generated (e.g., there is a time limit)?
- How to deal with cycles (states that can be reached multiple times)?

- What if the set of states reachable from the initial state is too large to be entirely generated (e.g., there is a time limit)?
  - Expand the state/action tree to some depth  $h$
  - Estimate the utilities of leaf nodes  
[Reminiscent of evaluation function in game trees]
  - Back-up utilities as described before (using estimated utilities at leaf nodes)

# Target-Tracking Example

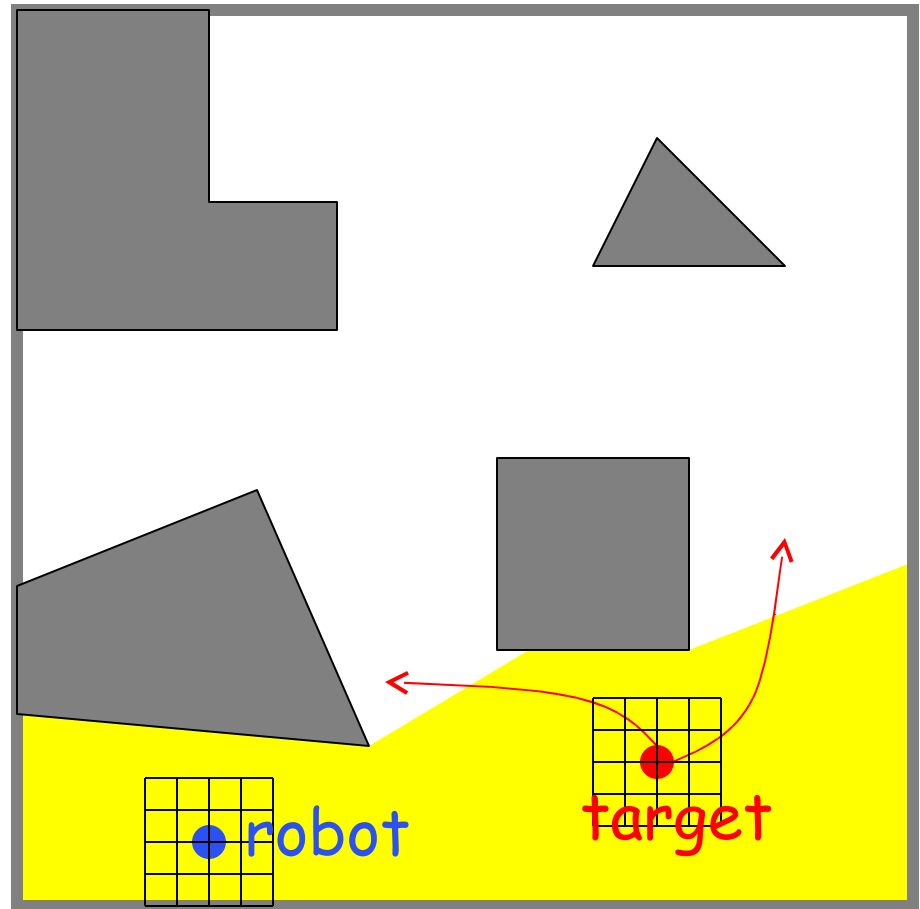


- The robot must keep a target in its field of view
- The robot has a prior map of the obstacles
- But it does not know the target's trajectory in advance

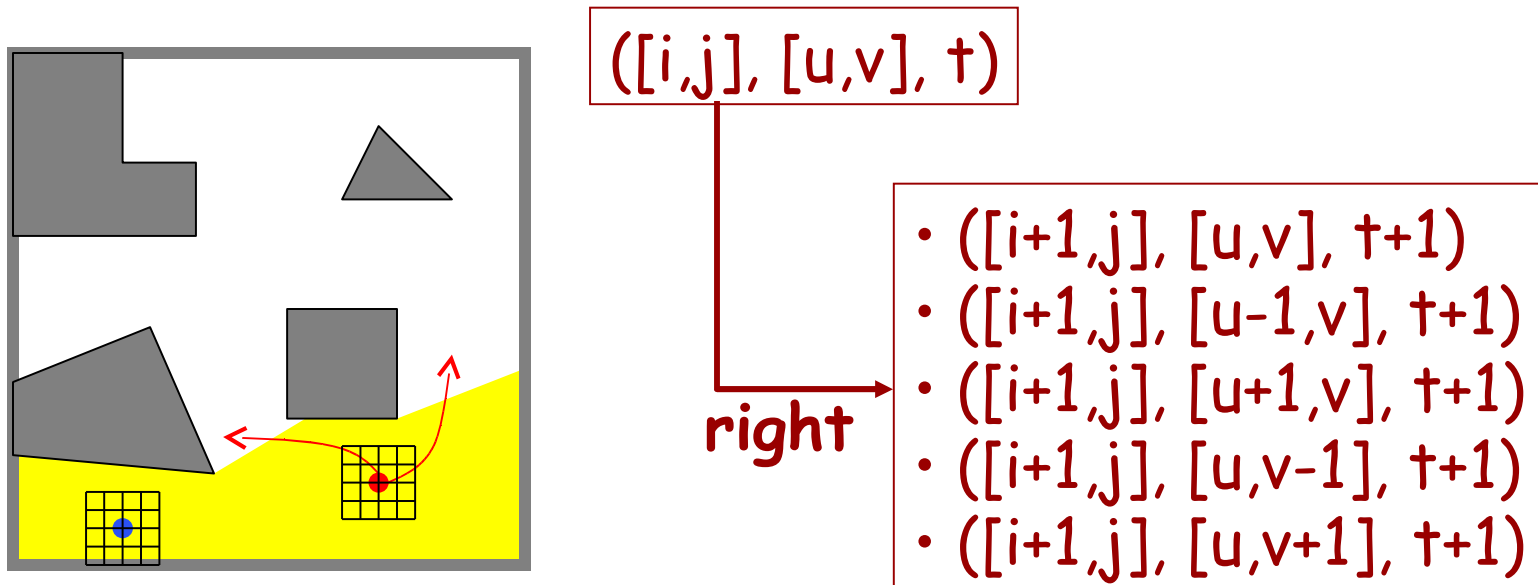


# Target-Tracking Example

- Time is discretized into small steps of unit duration
- At each time step, each of the two agents moves by at most one increment along a single axis
- The two moves are **simultaneous**
- The robot senses the new position of the target at each step
- The target is not influenced by the robot (**non-adversarial, non-cooperative** target)



# Time-Stamped States (no cycles possible)



- State = (robot-position, target-position, time)
  - In each state, the robot can execute 5 possible actions :  
{stop, up, down, right, left}
  - Each action has 5 possible outcomes (one for each possible action of the target), with some probability distribution
- [Potential collisions are ignored for simplifying the presentation]

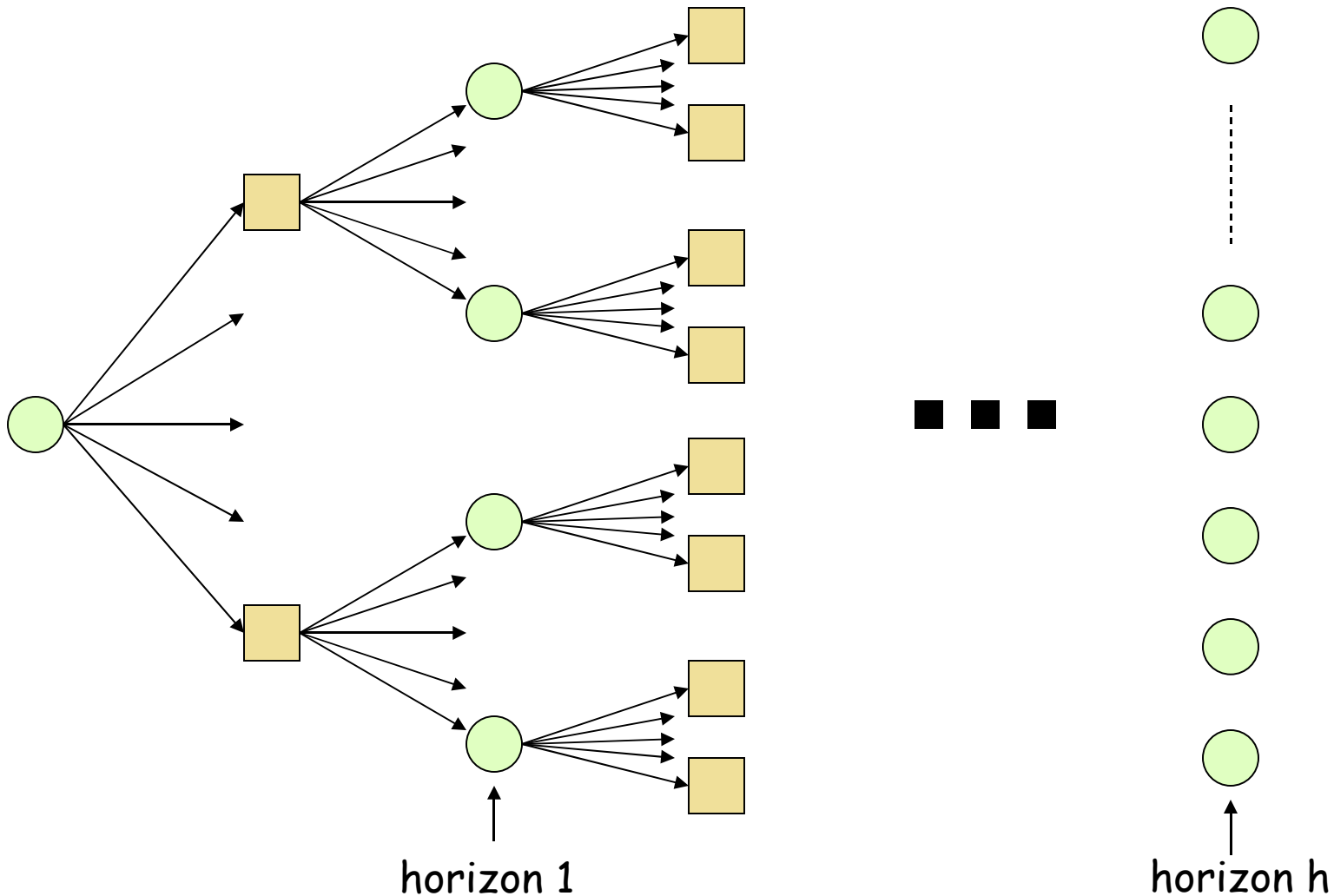
# Rewards and Costs

The robot must keep seeing the target as long as possible

- Each state where it does not see the target is terminal
- The reward collected in every non-terminal state is 1; it is 0 in each terminal state

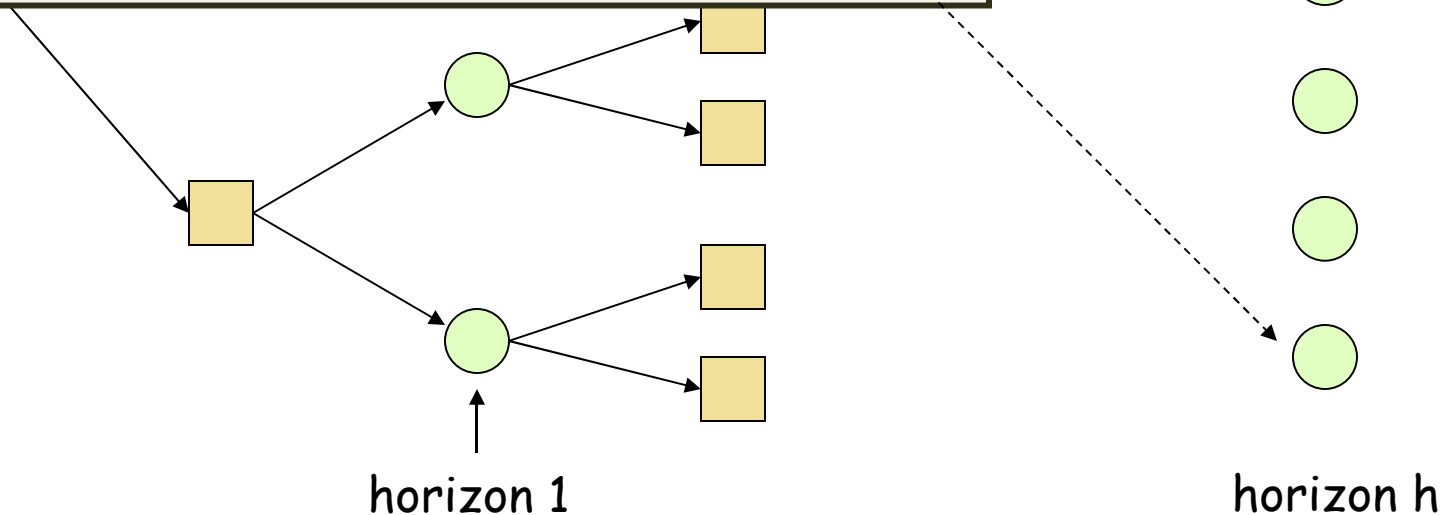
[→ The sum of the rewards collected in an execution run is exactly the amount of time the robot sees the target]

# Expanding the state/action tree

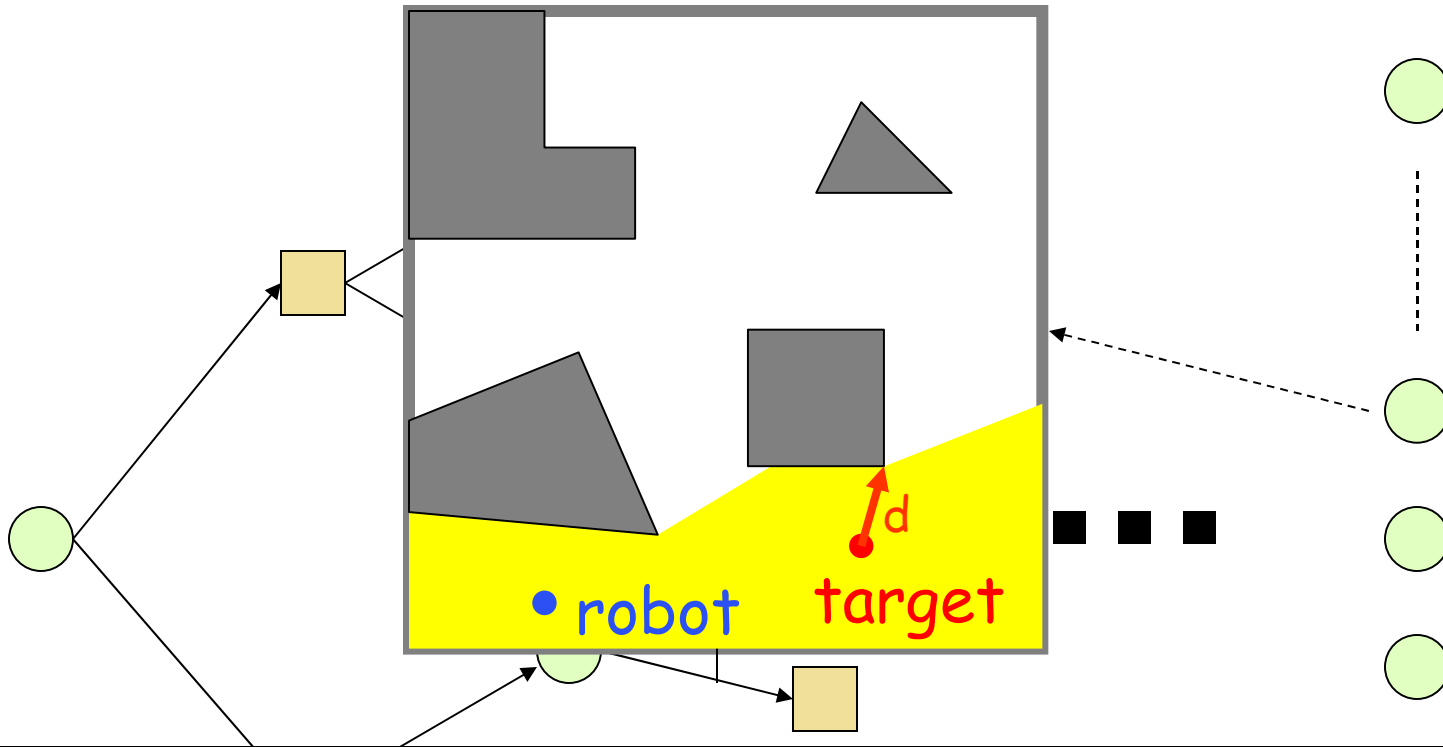


# Assigning rewards

- Terminal states: states where the target is not visible
- Rewards: 1 in non-terminal states; 0 in others
- But how to estimate the utility of a leaf at horizon  $h$ ?

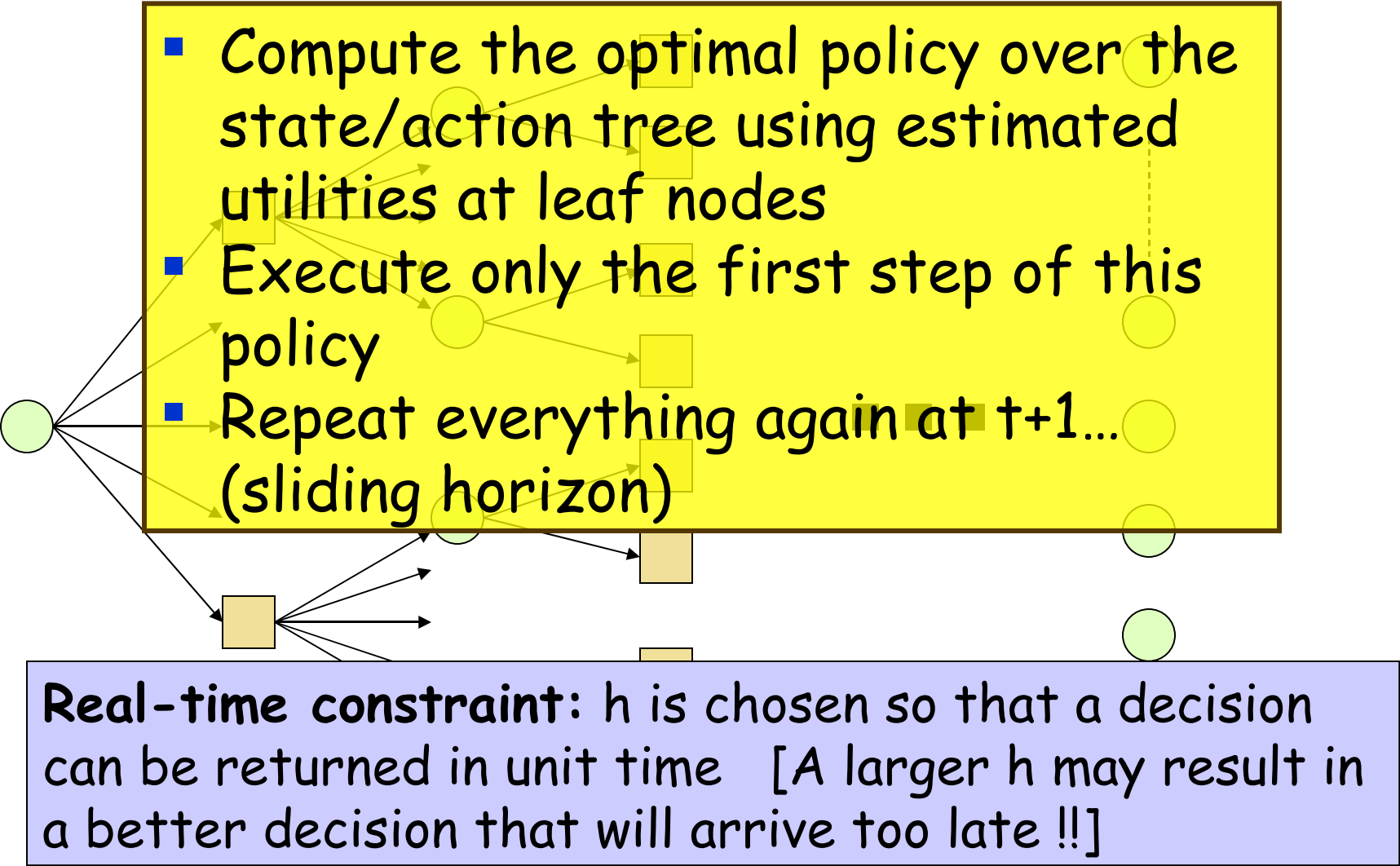


# Estimating the utility of a leaf



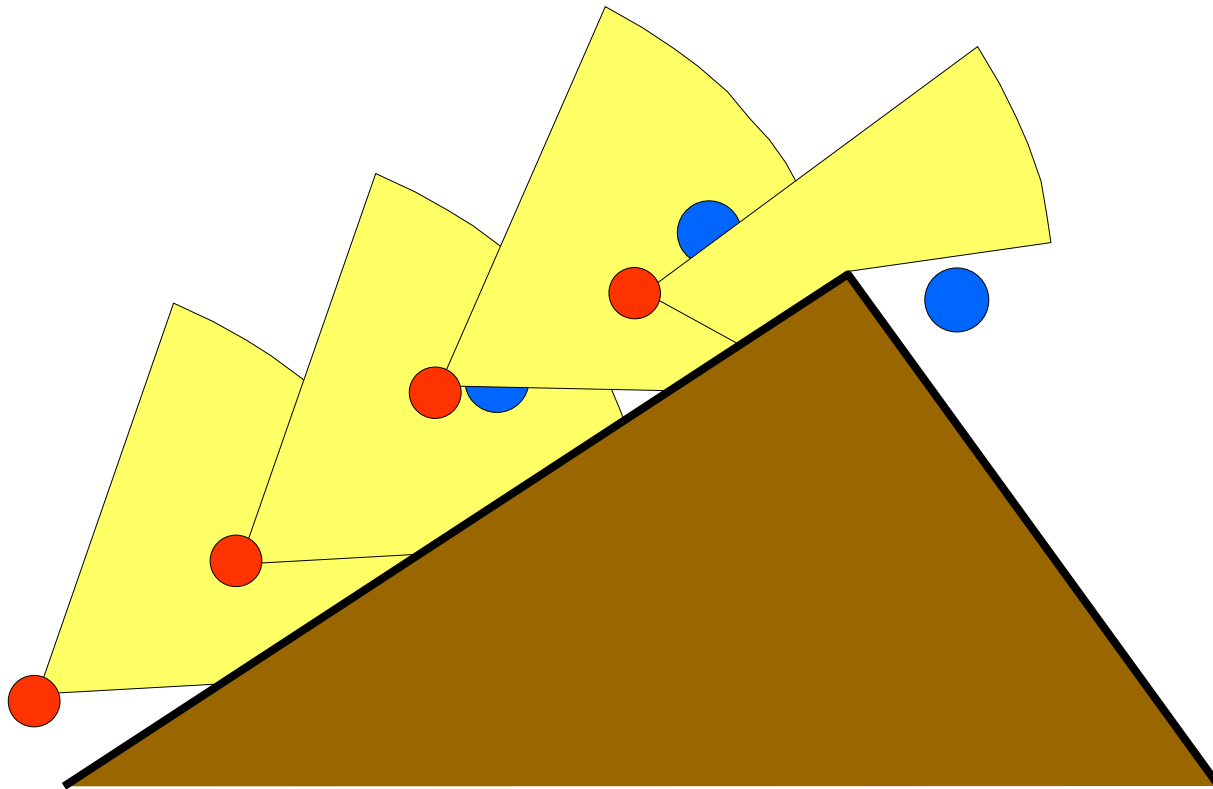
- Compute the shortest distance  $d$  for the target to escape the robot's current field of view
- If the maximal velocity  $v$  of the target is known, estimate the utility of the state to  $d/v$  [conservative estimate]

# Selecting the next action

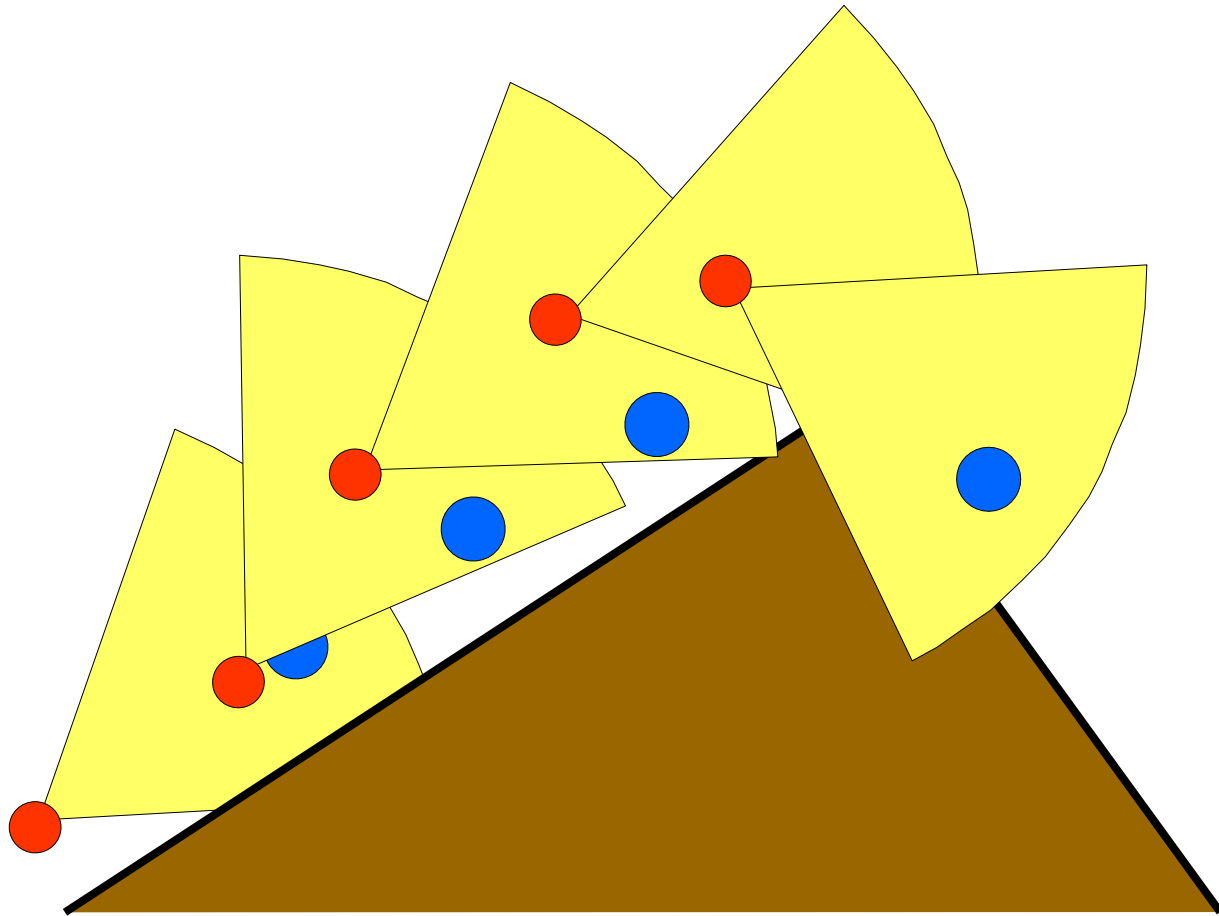
- 
- Compute the optimal policy over the state/action tree using estimated utilities at leaf nodes
  - Execute only the first step of this policy
  - Repeat everything again at  $t+1$ ... (sliding horizon)

**Real-time constraint:**  $h$  is chosen so that a decision can be returned in unit time [A larger  $h$  may result in a better decision that will arrive too late !!]

# Pure Visual Servoing

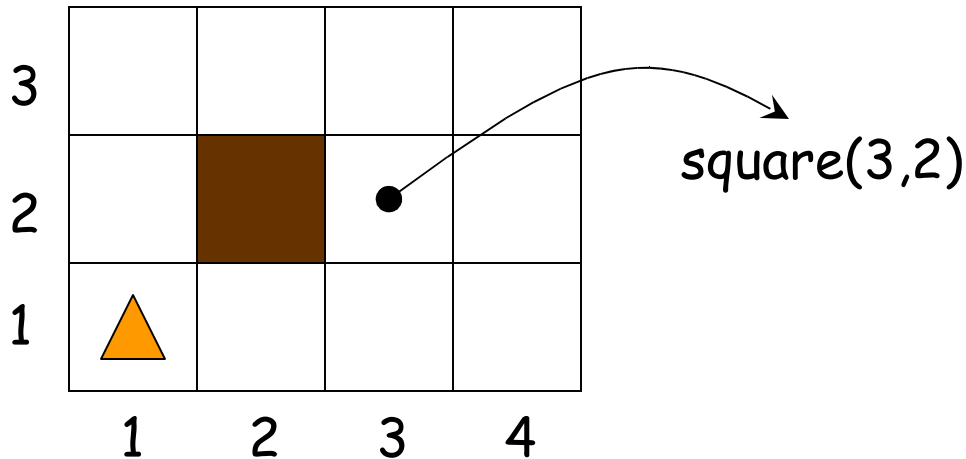



# Computing and Using a Policy



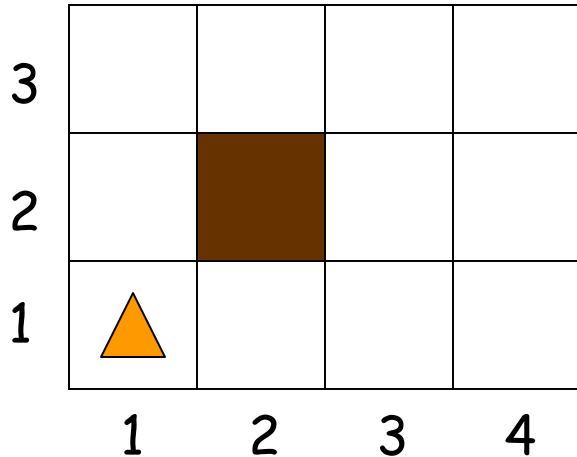
- In this target-tracking example stamping states by time is a “trick” to avoid cycles
- If the probabilistic distribution of the target’s possible moves at each time step does not depend on past history (Markov assumption), then the best move of the robot at each time step should only depend on the robot’s and target’s current positions, not on time
- To find this best move, we must remove the time stamp and handle cycles appropriately

# Robot Navigation Example



- The robot (shown ) lives in a world described by a 4x3 grid of squares with square (2,2) occupied by an obstacle
- A state is defined by the square in which the robot is located:  
(1,1) in the above figure  
→ 11 states

# Action (Transition) Model

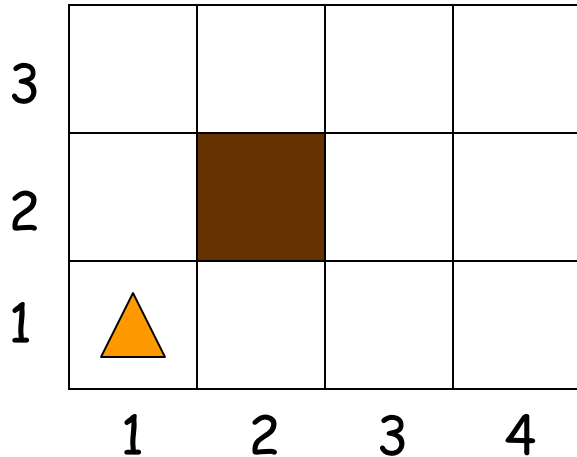


U brings the robot to:

- (1,2) with probability 0.8
- (2,1) with probability 0.1
- (1,1) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
  - For each action:
    - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
    - With probability 0.1 it moves in a direction perpendicular to the intended one
    - If the robot can't move, it stays in the same square
- [This model satisfies the Markov condition]

# Action (Transition) Model



L brings the robot to:

- (1,1) with probability  $0.8 + 0.1 = 0.9$
- (1,2) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
  - For each action:
    - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
    - With probability 0.1 it moves in a direction perpendicular to the intended one
    - If the robot can't move, it stays in the same square
- [This model satisfies the Markov condition]

# Terminal States, Rewards, and Costs

|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | -0.04 | -0.04 | -0.04 | +1    |
| 2 | -0.04 |       | -0.04 | -1    |
| 1 | -0.04 | -0.04 | -0.04 | -0.04 |
|   | 1     | 2     | 3     | 4     |

- Two terminal states: (4,2) and (4,3)
- Rewards:
  - $R(4,3) = +1$  [The robot finds gold]
  - $R(4,2) = -1$  [The robot gets trapped in quick sands]
  - $R(s) = -0.04$  in all other states
- Actions have zero cost  
[actually, they are encoded in the negative rewards of non-terminal states]

# State Utilities

|   |      |      |      |      |
|---|------|------|------|------|
| 3 | 0.81 | 0.87 | 0.92 | +1   |
| 2 | 0.76 |      | 0.66 | -1   |
| 1 | 0.71 | 0.66 | 0.61 | 0.39 |
|   | 1    | 2    | 3    | 4    |

- The utility of a state  $s$  is the maximal expected amount of reward that the robot will collect from  $s$  and future states by executing some action in each encountered state, until it reaches a terminal state (**infinite horizon**)
- Under the Markov and infinite horizon assumptions, **the utility of  $s$  is independent of when and how  $s$  is reached**  
[It only depends on the possible sequences of states after  $s$ , not on the possible sequences before  $s$ ]

# (Stationary) Policy

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | → | ↑ | ←  |
|   | 1 | 2 | 3 | 4  |

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

- A **stationary policy** is a complete map  $\Pi$ : state  $\rightarrow$  action
- For each non-terminal state it recommends an action, independent of when and how the state is reached
- Under the Markov and infinite horizon assumptions, **the optimal policy  $\Pi^*$  is necessarily a stationary policy**  
[The best action in a state does not depends on the past]

# (Stationary) Policy

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | → | ↑ | ←  |
|   | 1 | 2 | 3 | 4  |

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

- A **stationary policy** is a complete map  $\Pi$ : state  $\rightarrow$  action
- For each non-terminal state it recommends an action, independent of when an
- Under the **optimal policy  $\Pi^*$  is necessarily a stationary policy** [The best action in a state does not depends on the past]
- Finding  $\Pi^*$  is called an **observable Markov Decision Problem (MDP)**

The optimal policy tries to avoid "dangerous" state (3,2)

the optimal

# Execution of Optimal Policy

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

Repeat:

- $s \leftarrow$  sensed state
  - If  $s$  is terminal then exit
  - $a \leftarrow \Pi^*(s)$
  - Perform  $a$
- (Reactive agent algorithm)

- Let the robot execute  $\Pi^*$  many times from the same state  $s$
- Each run produces a sequence of states with some probability
- Each possible sequence collects a certain amount of reward
- The utility of  $s$  is the average amount of reward collected over the successive executions of  $\Pi^*$
- It is maximal over all policies

# Optimal Policies for Various $R(s)$

|   |   |   |    |
|---|---|---|----|
| → | → | → | +1 |
| ↑ |   | ↑ | -1 |
| ↑ | ← | ← | ←  |

$R(s) = -0.04$

|   |   |   |    |
|---|---|---|----|
| → | → | → | +1 |
| ↑ |   | → | -1 |
| → | → | → | ↑  |

$R(s) = -2$

|   |   |   |    |
|---|---|---|----|
| → | → | → | +1 |
| ↑ |   | ← | -1 |
| ↑ | ← | ← | ↓  |

$R(s) = -0.01$

|   |   |   |    |
|---|---|---|----|
| ↕ | ↕ | ← | +1 |
| ↕ |   | ← | -1 |
| ↕ | ↕ | ↕ | ↓  |

$R(s) > 0$

# Defining Equations

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

- If  $s$  is terminal:  
 $U(s) = R(s)$
- If  $s$  is non-terminal:  
 $U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')$   
[Bellman equation]
- $\Pi^*(s) = \arg \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s)U(s')$

# Defining Equations

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

The utility of  $s$  depends on the utility of other states  $s'$  (possibly, including  $s$ ), and vice versa

The equations are non-linear

- If  $s$  is terminal:  
 $U(s) = R(s)$

- If  $s$  is non-terminal:

$$U(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$$

[Bellman equation]

- $\Pi^*(s) = \arg \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$

# Value Iteration Algorithm

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 |   | 0 | -1 |
| 1 | 0 | 0 | 0 | 0  |
|   | 1 | 2 | 3 | 4  |

→

|   |      |      |      |      |
|---|------|------|------|------|
| 3 | 0.81 | 0.87 | 0.92 | +1   |
| 2 | 0.76 |      | 0.66 | -1   |
| 1 | 0.71 | 0.66 | 0.61 | 0.39 |
|   | 1    | 2    | 3    | 4    |

- Initialize the utility of each non-terminal states to  $U_0(s) = 0$
- For  $t = 0, 1, 2, \dots$  do

$$U_{t+1}(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U_t(s')$$

for each non-terminal state  $s$

# Value Iteration Algorithm

|   |      |      |      |      |
|---|------|------|------|------|
| 3 | 0.81 | 0.87 | 0.92 | +1   |
| 2 | 0.76 |      | 0.66 | -1   |
| 1 | 0.71 | 0.66 | 0.61 | 0.39 |
|   | 1    | 2    | 3    | 4    |

→

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
| 2 | ↑ |   | ↑ | -1 |
| 1 | ↑ | ← | ← | ←  |
|   | 1 | 2 | 3 | 4  |

- Initialize the utility of each non-terminal states to  $U_0(s) = 0$
- For  $t = 0, 1, 2, \dots$  do

$$U_{t+1}(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U_t(s')$$

for each non-terminal state  $s$

- For each non-terminal state  $s$  do

$$\Pi^*(s) = \mathbf{arg} \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$$

# Value Iteration Algorithm

|   |      |      |      |    |
|---|------|------|------|----|
| 3 | 0.81 | 0.87 | 0.92 | +1 |
|---|------|------|------|----|

|   |   |   |   |    |
|---|---|---|---|----|
| 3 | → | → | → | +1 |
|---|---|---|---|----|

Value iteration is essentially the same as computing the best move from each state using a state/action tree expanded to a large depth  $h$  (with estimated utilities of the non-terminal leaf nodes set to 0)

By doing the computation for all states simultaneously, it avoids much redundant computation

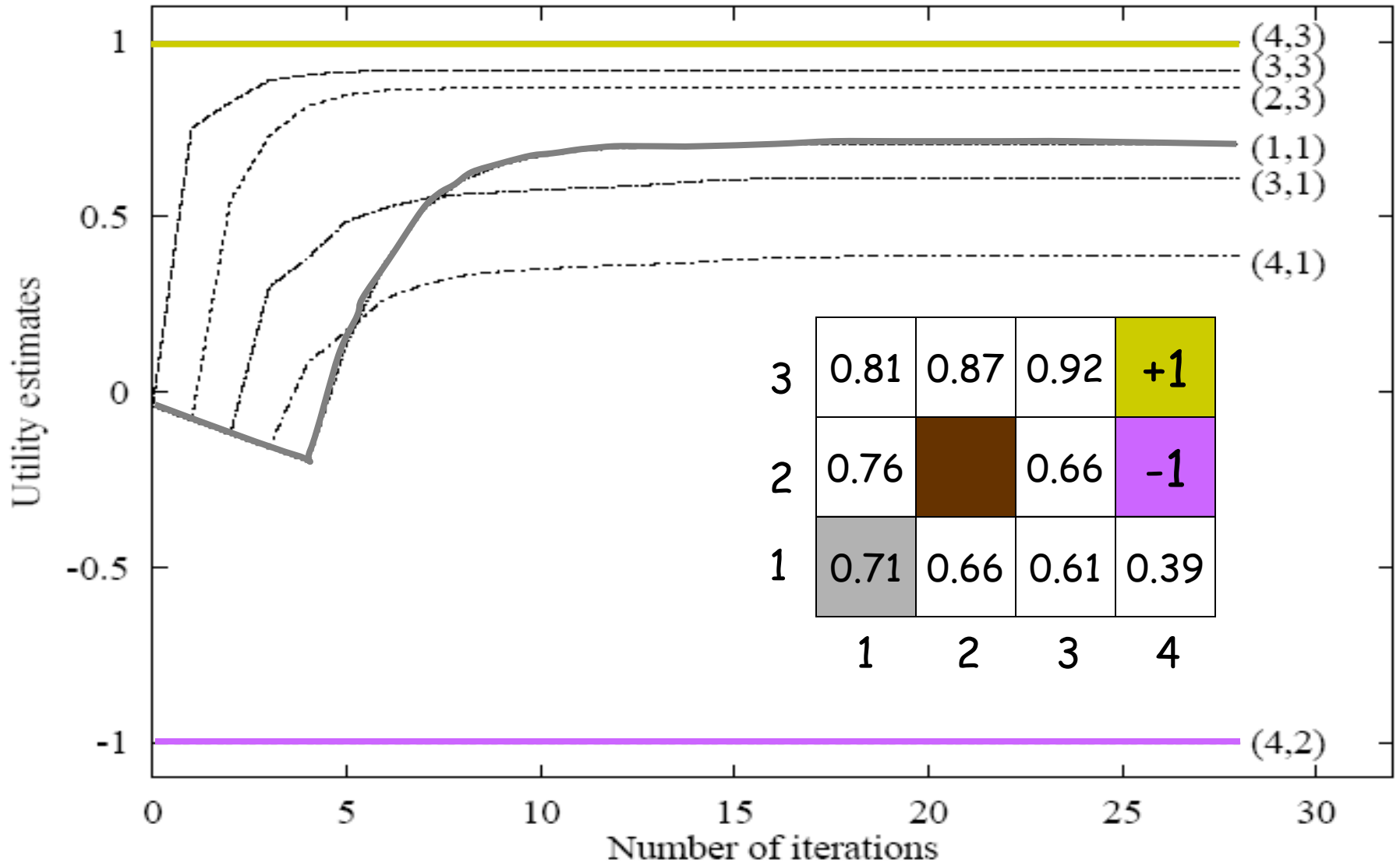
$$U_{t+1}(s) = R(s) + \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a.s) U_t(s')$$

for each non-terminal state  $s$

- For each non-terminal state  $s$  do

$$\Pi^*(s) = \arg \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a.s) U(s')$$

# Convergence of Value Iteration



# Convergence of Value Iteration

- If:
  - The number of states is finite
  - There exists at least one terminal state that gives a positive reward and is reachable with non-zero probability from every other non-terminal state (connectivity of the state space)
  - $R(s) \leq 0$  at every non-terminal state
  - The cost of every action is  $\geq 0$
- Then value iteration converges toward an optimal policy *if we wait long enough*
- But what if the above conditions are not verified?  
*[E.g. a "taxi-driver" example where there is no*

# Discount Factor

- Idea: Prefer short-term rewards over long-term ones
- If the execution of a policy from a state  $s_0$  produces the sequence  $s_0, s_1, \dots, s_n$  of states, then the amount of collected reward is

$$R(s_0) + \gamma R(s_1) + \dots + \gamma^n R(s_n) = \sum_{i=0, \dots, n} \gamma^i R(s_i)$$

where  $0 < \gamma < 1$  is a constant called the **discount factor**

- The defining equation of the utilities becomes:

$$U(s) = R(s) + \gamma \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s,a)} P(s'|a,s) U(s')$$

- Using a discount factor guarantees the convergence of V.I. in finite state spaces, even if there is no terminal states and reward/cost values for each state/action are arbitrary (but finite) [Intuition: The nodes in the state/action tree get less and less important as we go deeper in the tree]

# Remarks

- Value iteration may give the optimal policy  $\Pi^*$  long before the utility values have converged
- There is only a finite number of possible policies

→ Policy iteration algorithm

# Policy Iteration Algorithm

- Pick any policy  $\Pi$
- $\text{changed?} \leftarrow \text{true}$
- Repeat until  $\neg \text{changed?}$

often a sparse one

- [Policy evaluation] Solve the linear system

$$U_{\Pi}(s) = R(s) + \gamma \sum_{s' \in \text{Succ}(s, \Pi(s))} P(s' | \Pi(s), s) U_{\Pi}(s')$$

- $\text{changed?} \leftarrow \text{false}$
- [Policy improvement] For each state  $s$  do

- If  $\max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s, a)} P(s' | a, s) U_{\Pi}(s') >$   
 $\sum_{s' \in \text{Succ}(s, \Pi(s))} P(s' | a, s) U_{\Pi}(s')$

then

- $\Pi(s) \leftarrow \arg \max_{a \in \text{Appl}(s)} \sum_{s' \in \text{Succ}(s, a)} P(s' | a, s) U_{\Pi}(s')$
- $\text{changed?} \leftarrow \text{true}$

# Finite vs. Infinite Horizon

If the agent must maximize the expected amount of collected award within a maximum number of steps  $h$  (finite horizon), the optimal policy is no longer stationary

→ We stamp states by time (step number) just as in the target-tracking example

We compute the optimal policy over the state/action tree expanded to depth  $h$ , in which all leaf states are now terminal

[E.g. taxi driver at the end of his/her work day]

# Summary

- MDPs
- Policy / Stationary Policy
- Types of rewards : additive, discounted
- Value Iteration
- Policy Iteration