Sequential Decision Making and Reinforcement Learning

(SDMRL)

Intro, AI Approaches and Formal Models for Decision-Making

Sarah Keren

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Agenda

- Course structure
- Approaches to decision making in AI
- Formal Models for Decision Making

Course structure and objectives

A little bit about me

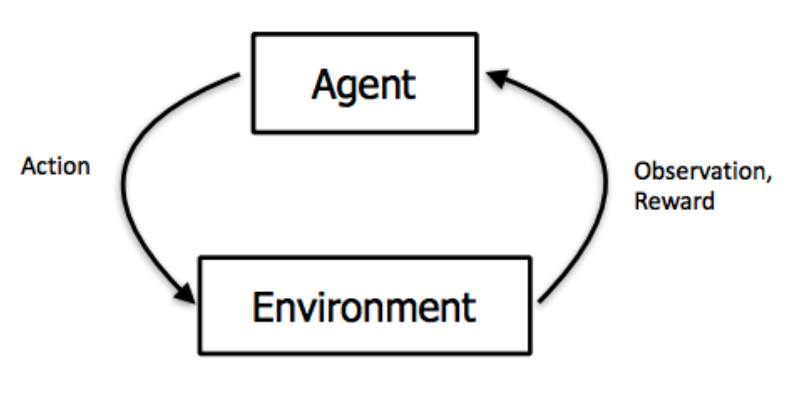
- I am a member of the Computer Science Faculty at the Technion.
- I head the Collaborative AI and Robotics (CLAIR) lab
- My research focuses on multi-agent AI and multi-robot systems.
- My work offers novel approaches to increasing the capabilities of AI agents via the **design** of the environments in which they are intended to act.
- My email: *sarahk@cs.technion.ac.il*

Staff and schedule

- **Teaching:** Sarah Keren
  sarahk@cs.technion.ac.il
- **TA'ing:** Itay Segev
  itaysegev@campus.technion.ac.il
- **Assignment TA:** Yuval Goshen
  yuval.goshen@campus.technion.ac.il
- **Schedule:**
  - Lesson: Tuesday 10:30-12:30
  - Lab: Tuesday 12:30-13:30
  - Reception Hours: per request.

What is this class about ?

Sequential Decision Making: deciding how to act now in the world in order to accomplish a long-term objective



- From observations to decisions
- From low-level to high-level reasoning
- From theory to practice to theory ...
- From single agent to multi-Agent

Our Focus

A wide spectrum of AI research questions.

- Non-adversarial settings
- Complex tasks
- Combinations of approaches (and principled ways to choose the best approach)

Course Structure

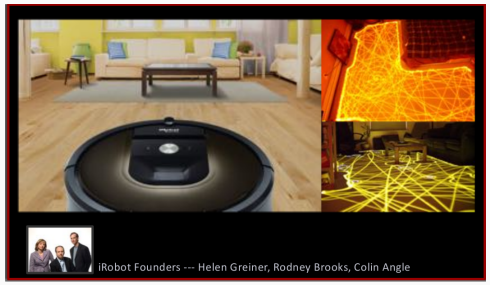
- Weekly 2-hour class followed by a 1-hour hands-on lab.
- In class we will cover the theory behind decision making in AI.
- In the lab we will practice the ideas we discuss in class.

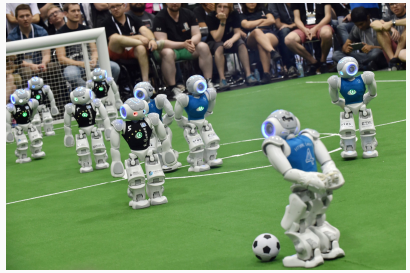
Acknowledgments

- Some slides by Erez Karpas and Carmel Domshlak
- An Introduction to Reinforcement Learning, Sutton and Barto, MIT Press, 2018
- David Silver's course on RL
  https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

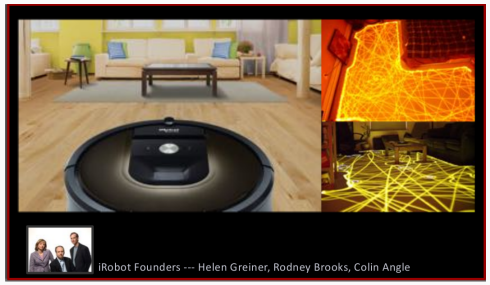
Decision Making in AI

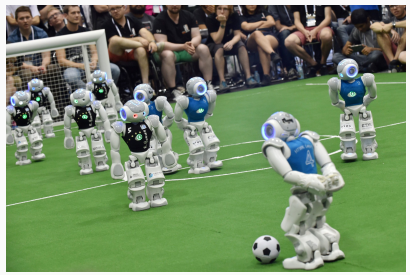
Reactive vs. Rational Agent





Reactive vs. Rational Agent

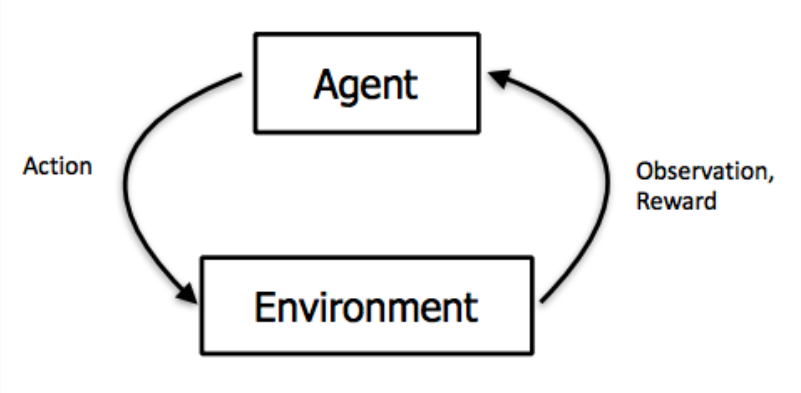




- **Reactive** or **Reflex** agent:
  - simple reactions to stimulus (short-term)
  - mostly used when fast response times are needed or when computational resources are limited
- **Rational agent:**
  - has extended reasoning capabilities and can use its resources and skills to achieve complex objectives autonomously
  - selects actions that maximize its (expected) utility (long term)
  - can learn and gain knowledge from the environment
  - can react to changes in the environment
  - social ability: can negotiate, cooperate, or compete by communicating with other agents

Single Agent Autonomy

What does it mean to be autonomous ?

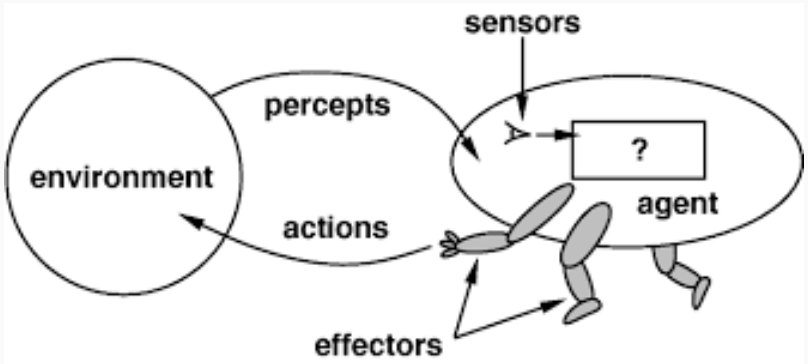


We seek a **policy** from states (or knowledge about the state) to actions.

Components of an Autonomous Agents?

Components of an Autonomous Agents?

- Action
  - Virtual
  - Physical (Actuation)
    - Locomotion
    - Manipulation
    - Interaction
- Perception (Sensors)
  - Internal
  - External
- Cognition (Control)
  - From reactive to proactive
  - From finite state machines to cognitive agents



Sequential Decision Making

We will focus on sequential decision making (a.k.a. planning)

Sequential Decision Making

We will focus on sequential decision making (a.k.a. planning)

Approaches to Decision Making

We seek a **policy**, which we mark as π that will map states or (knowledge about the state) to actions.

Can be deterministic $\pi : \mathcal{S} \mapsto \mathcal{A}$ or probabilistic $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$

More accurately, we seek a mapping from beliefs $\mathcal{B}$, representing knowledge about the current state (formally defined later on), to actions.

Basic approaches:

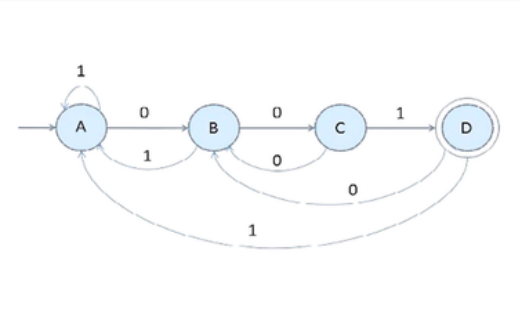
- Programming/reactive
- Model-Based Planning (using a domain-independent descriptive language)
- Machine Learning (from data and experience)
- Reinforcement Learning

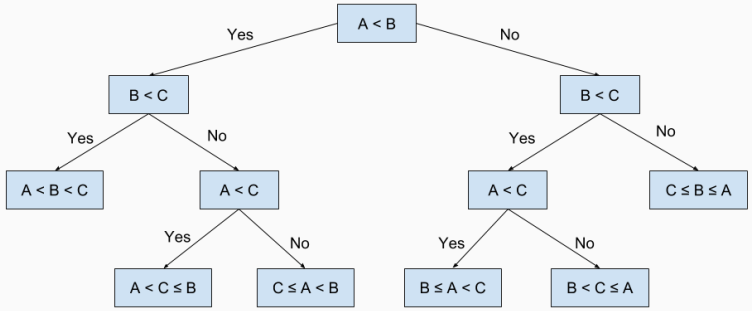
Example

You are in Kibutz Ein Dor and you want to get to Austin Texas.

How do you decide how to get to Austin?

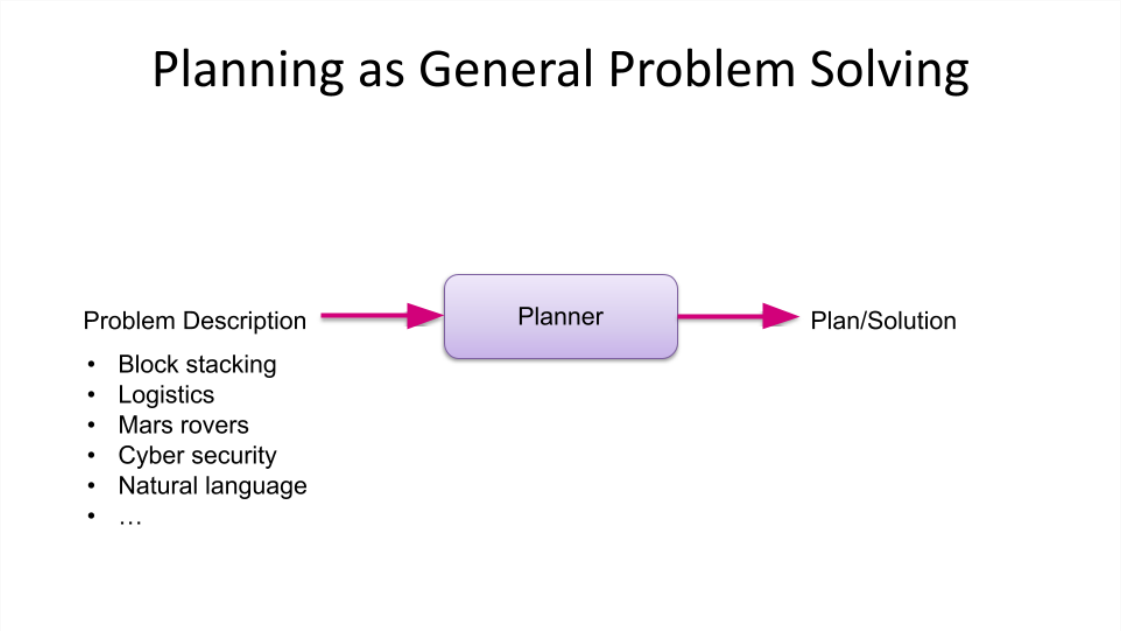
Programming/reactive





Very practical in many applications, but lacks flexibility.

Model-Based Planning



From Erez Karpas's course on automated planning **Model-based Planning** to autonomous behavior.

Model-Based Planning

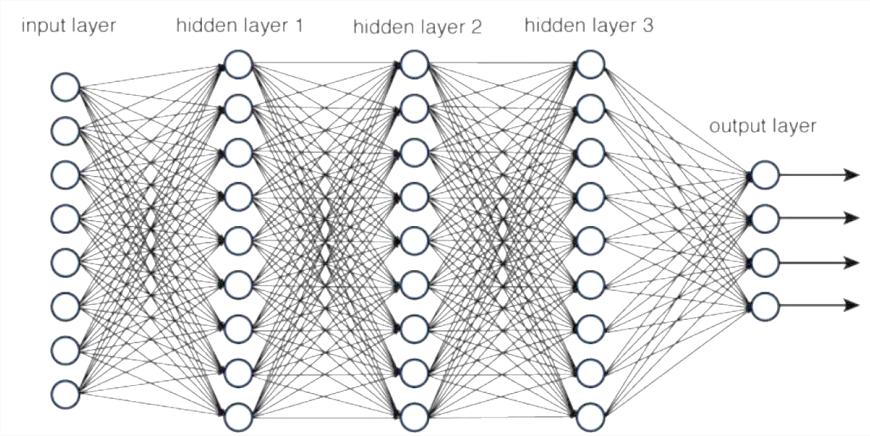
Characterized by:

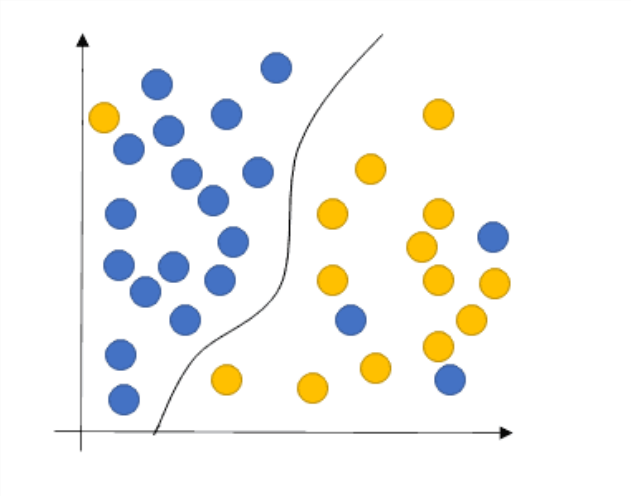
- A set of **states** $\mathcal{S}$ a system can be in.
  - a state is a full assignment to the set of **variables** (features) $\mathcal{X}$.
- **Actions** $\mathcal{A}$ change the values of certain variables.
- **Reward Function** $\mathcal{R}$ sets a numeric signal passed from the environment (can represent cost)
  - used to signal the objective
  - some domains have **goal conditions** such that the agent should reach **goal states** that satisfy is(e.g., 'be at Austin').
- **Objective:** find a **policy** that drives the initial state into a goal state or that maximizes the expected accumulated reward.
- Language is **generic** and not domain specific.
- **Complexity:** Even in the simplest setting it is NP-hard; i.e., exponential in the number of variables in the worst case.

Machine Learning

Use statistics to find patterns in massive amounts of data.

The input vector x (e.g., an image) is mapped to an output $f(x)$, which represents a classification label, a probability distribution over the possible labels, or object labels with suitable bounding boxes.





Machine Learning

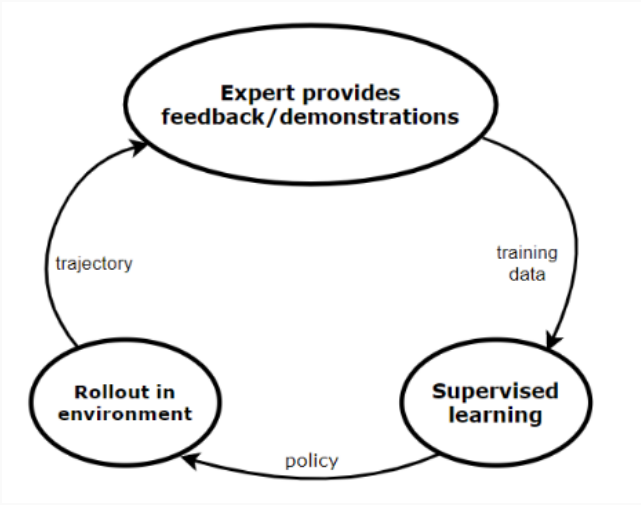
Template algorithm (for supervised learning)*:

- **Input:**
  - Sample set: $S = \{(x_i, y_i)_{i=1}^m\}$
  - Model calls H (candidate classifiers)
- **Output:**
  - Classifier with lowest **expected error** (e.g. loss).

* the formulation used in Intro to ML course

Machine Learning: Imitation Learning

- Learning from demonstrations
- Useful when it is easier for an expert to demonstrate the desired behavior rather than to specify a reward function.
- Simplest form: **behaviour cloning (BC)**, learning the expert's policy using supervised learning.



What's the sample set here? What does loss represent?

Image by SmartLab AI

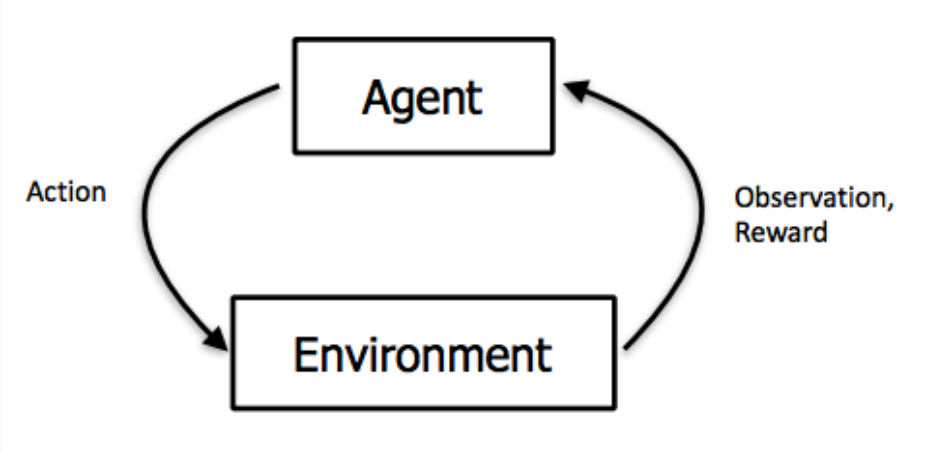
Are we done?

Figure 1: NVIDIA DRIVE

We have very advanced planning and imitation learning methods - what's missing?

Reinforcement Learning (RL)

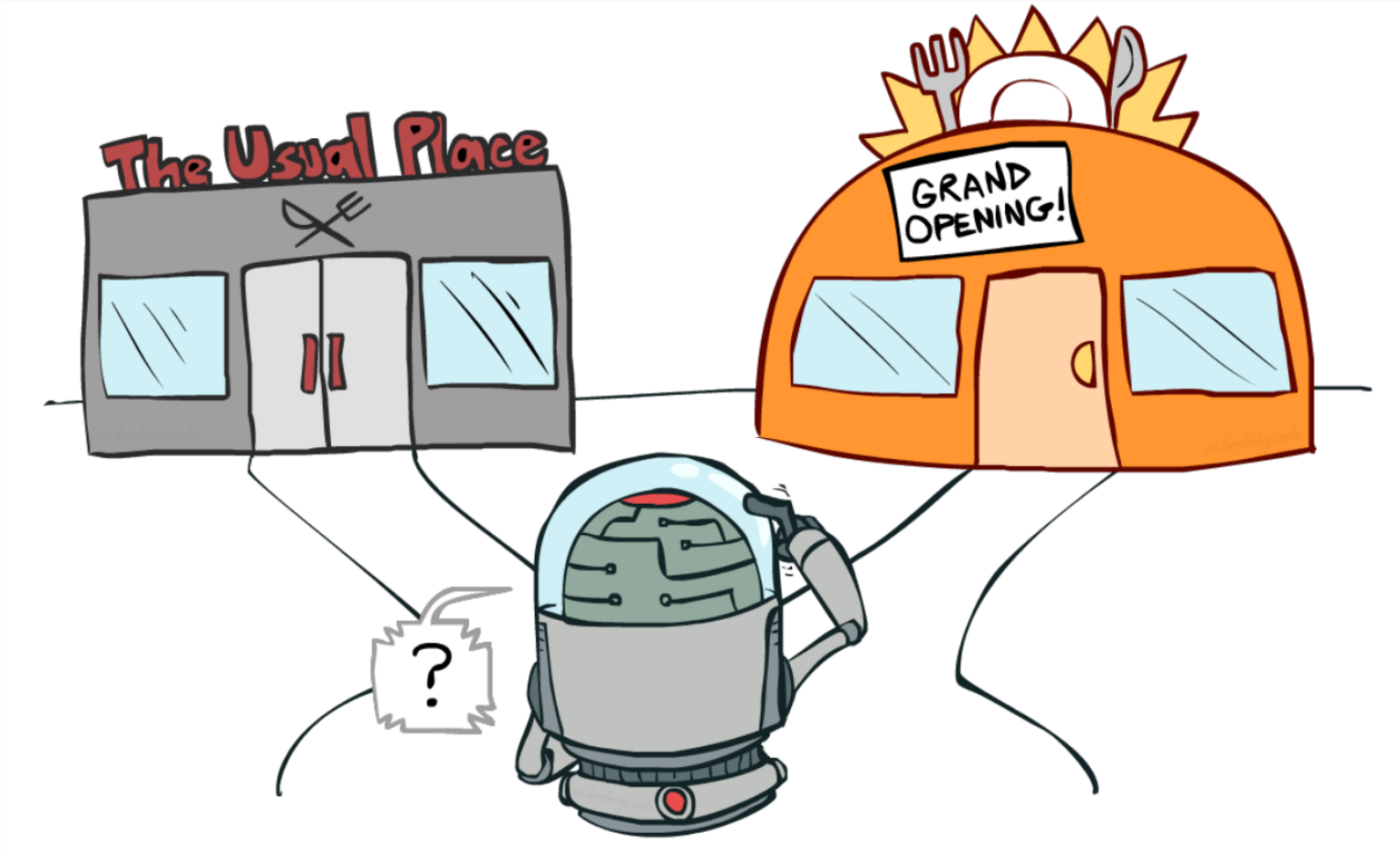
- **Learning from interaction** - how to behave to maximize the accumulated reward/achieve a goal.
- The learner is not told which actions to take, but instead **must discover** which actions yield the highest total **reward** by trying them.



Is reward enough?
https://deepmind.com/research/publications/2021/Reward-is-Enough

Reinforcement Learning (RL)

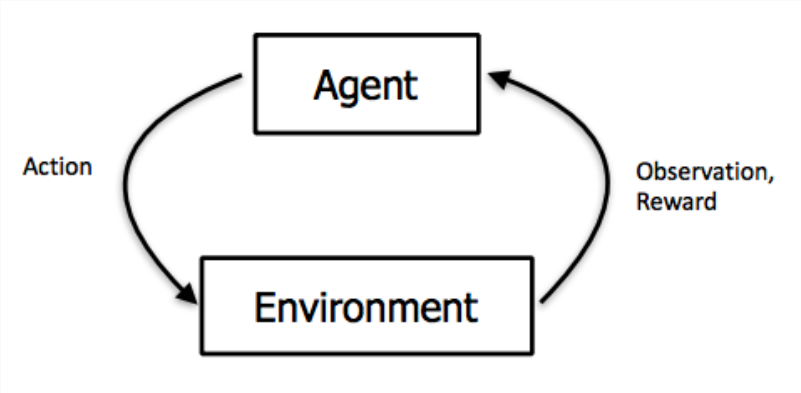
In RL there's a tradeoff between **exploration** (choosing an action for which the outcome is unknown or choosing a random action) and **exploitation** (choosing an action based on learned values).



Reinforcement Learning (contd.)

The process of Reinforcement Learning involves:

- Observing the environment
- Deciding how to act using some strategy
- Acting accordingly
- Receiving a reward (or penalty) and an observation
- Learning from the experiences and refining the policy
- Iteratively improving until converting to an optimal policy



- RL vs. Model-Based Planning ?
- RL vs. ML ?
- RL vs. IL ?

RL vs. Model-Based Planning

Both are fundamental problems in sequential decision-making

- **Model-Based Planning:** Involves creating a sequence of actions to achieve a specific goal based on a known model of the environment. It relies on a predefined representation of the environment and the dynamics governing it.
- **Reinforcement Learning:** The model of the environment is initially unknown. The agent improves its policy by interacting with the environment.

The distinction is very fuzzy...

RL vs. ML

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data): agent's actions affect the subsequent reward it receives

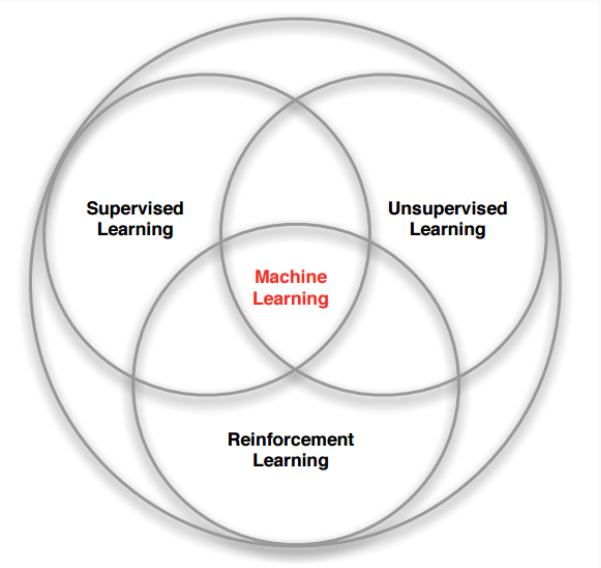


Image by David Silver.

RL vs. IL

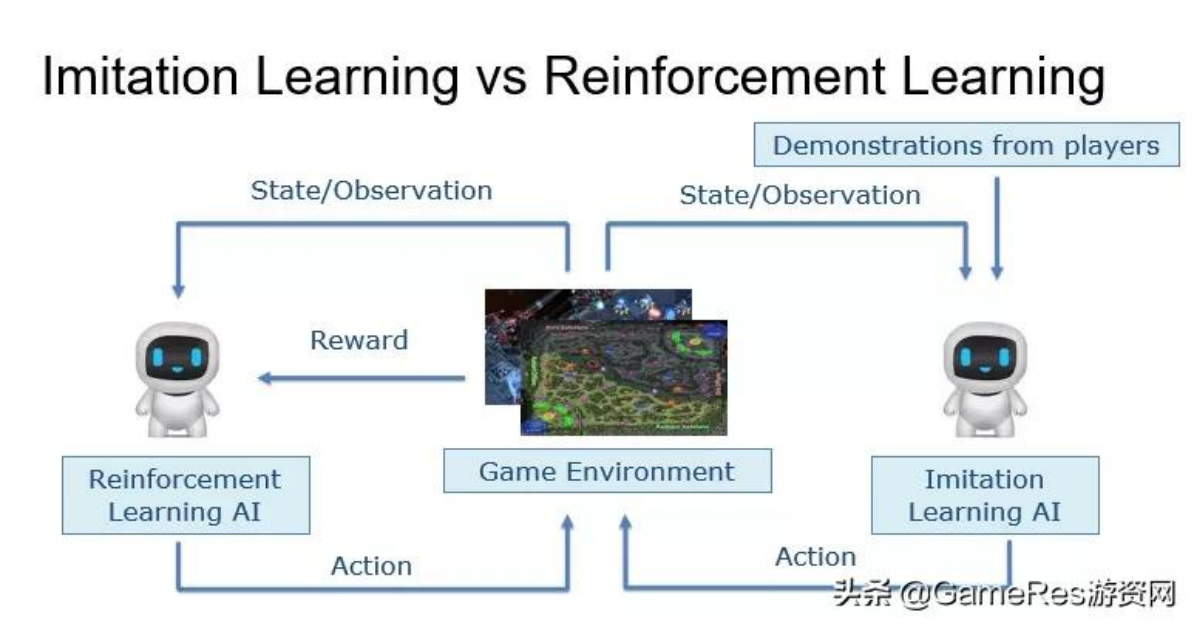
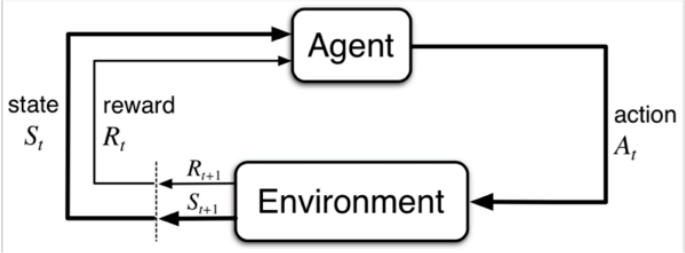


Image from *https://inf.news/en/game/3e21da98acd998d2774deac09944e5b0.html*

Summary of AI approaches

- Different approaches for decision-making: from data-driven to model-based approaches.
- Different settings require different solutions.



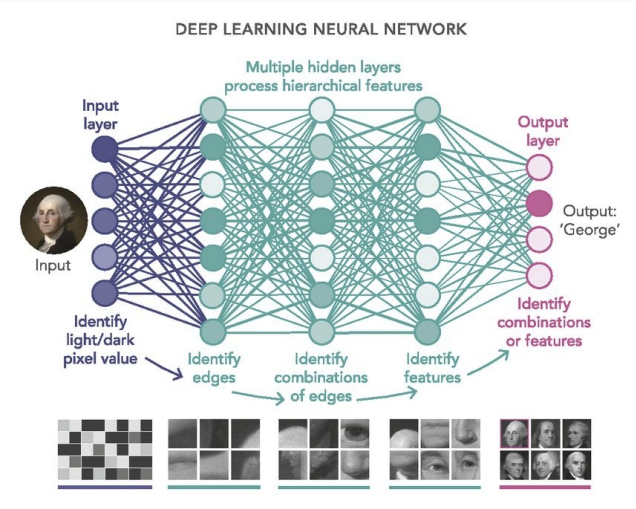
Open Questions:

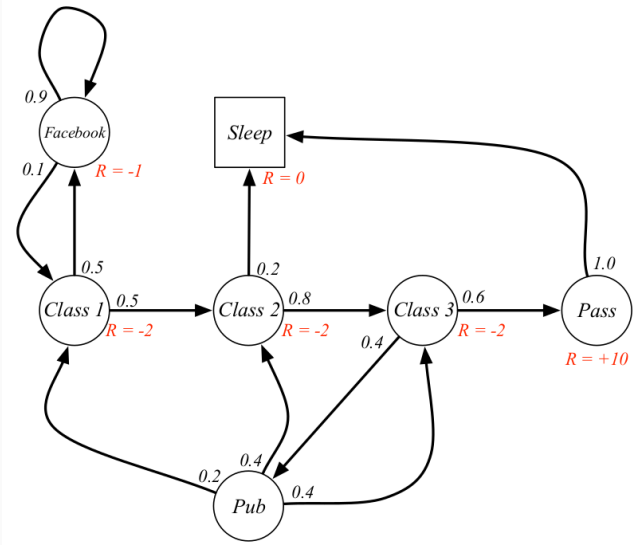
- How to choose the right paradigm for your problem?
- What is a good model for the decision making process?
- How does the presence of **other agents** affect these decision?

Characteristics of AI Settings

Models as World Descriptions

- In ML:
  - **Model** is used to represent the structure of the network used for classification / regression.
  - Part of the output of the algorithm
- Here:
  - **Model** denotes the representation / abstraction of the environment.
  - Part of the input



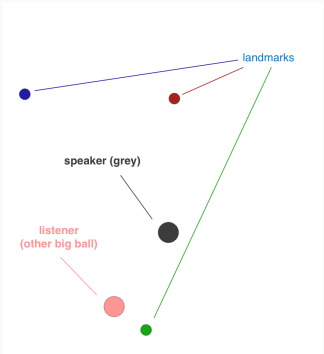


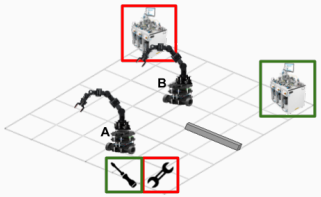
What do all these have in common?

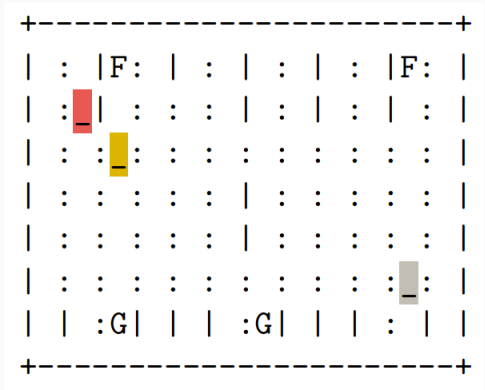
- Route selection (from Arad to Bucharest)
- Solving 15-puzzle (or Rubik's cube, or ...)
- Selecting and ordering movements of an elevator or a crane
- Production lines control
- Autonomous robots
- Crisis management
- ...

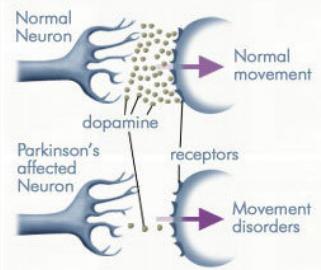
What do all these have in common?

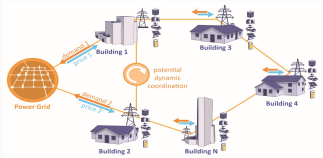
What do all these have in common?





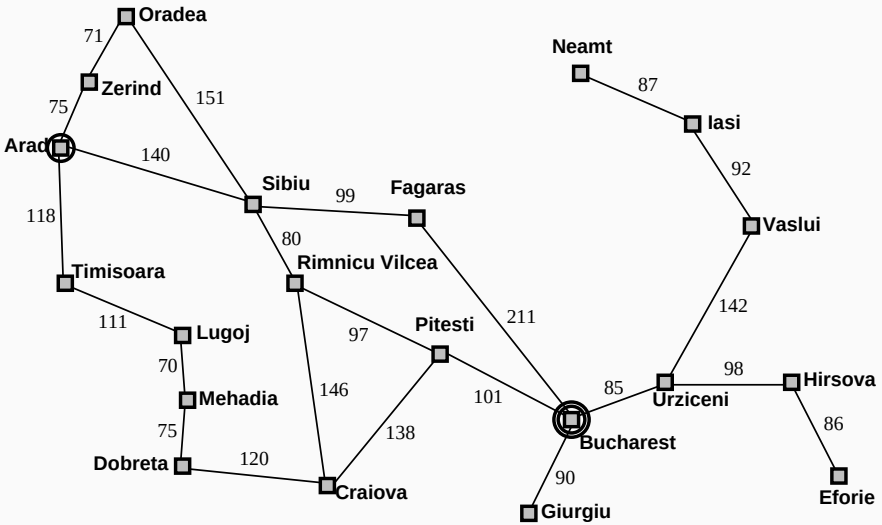


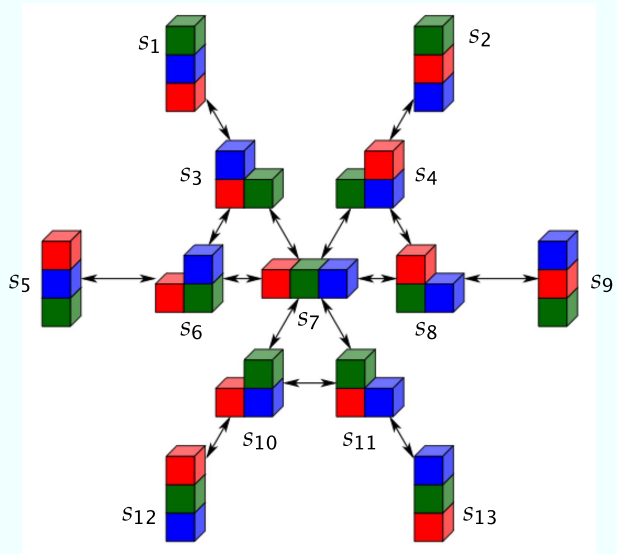




Planning and sequential decision making

- All these problems deal with action selection or control
- Some notion of problem state
- (Often) specification of initial state and/or goal state
- Legal moves or actions that allow transition from states to other states





Modeling the Environment

What do we need to model ?

Modeling the Environment

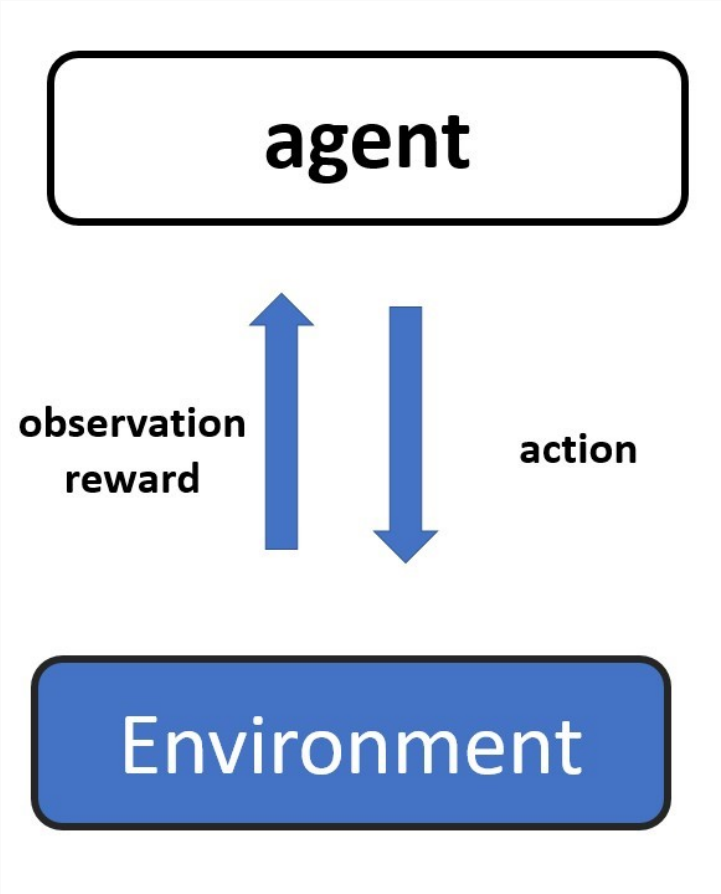
What do we need to model ?

- Sequential decision making
- Uncertainty in action outcomes
- Partial observability

Additional considerations:

The Model needs to be compact enough to be solvable but informative enough to be useful.

Modeling the Environment



Is this always a reasonable assumption?

Different classes of problems

Properties

- **Dynamics:** deterministic, nondeterministic or probabilistic
- **Observability:** full, partial, or none
- **Horizon:** finite or infinite
- ...

Side comment ...

- It is not that deterministic problems are easy
- It is not even clear that they are too far from modeling real-world problems well!

Properties of the setting: dynamics

Deterministic dynamics

Action + current state uniquely determine successor state.

Nondeterministic dynamics

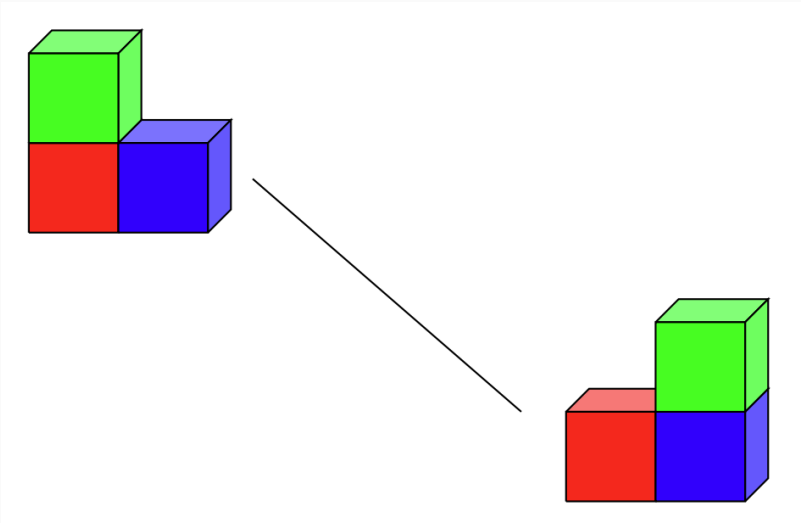
For each action and current state there may be several possible successor states.

Probabilistic dynamics

For each action and current state there is a probability distribution over possible successor states.

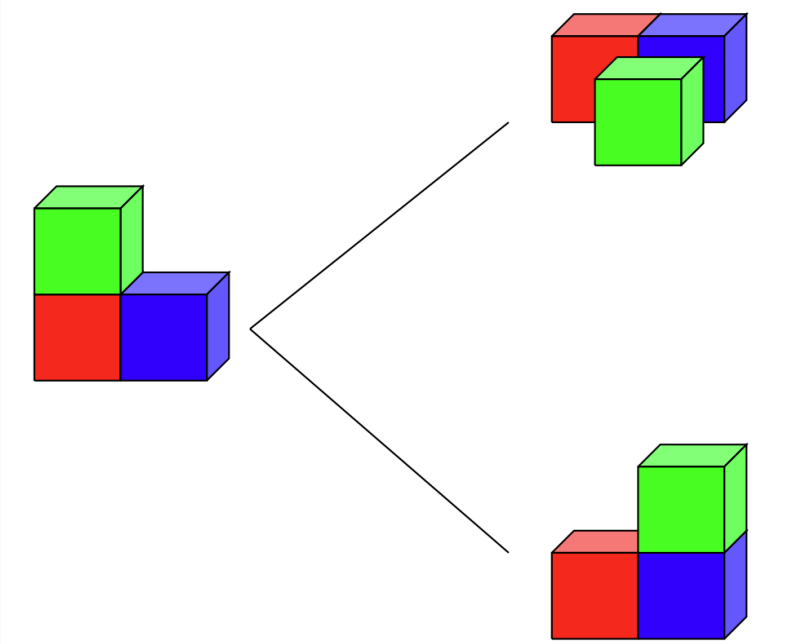
Deterministic dynamics example

Moving objects with a robotic hand:
move the green block onto the blue block.



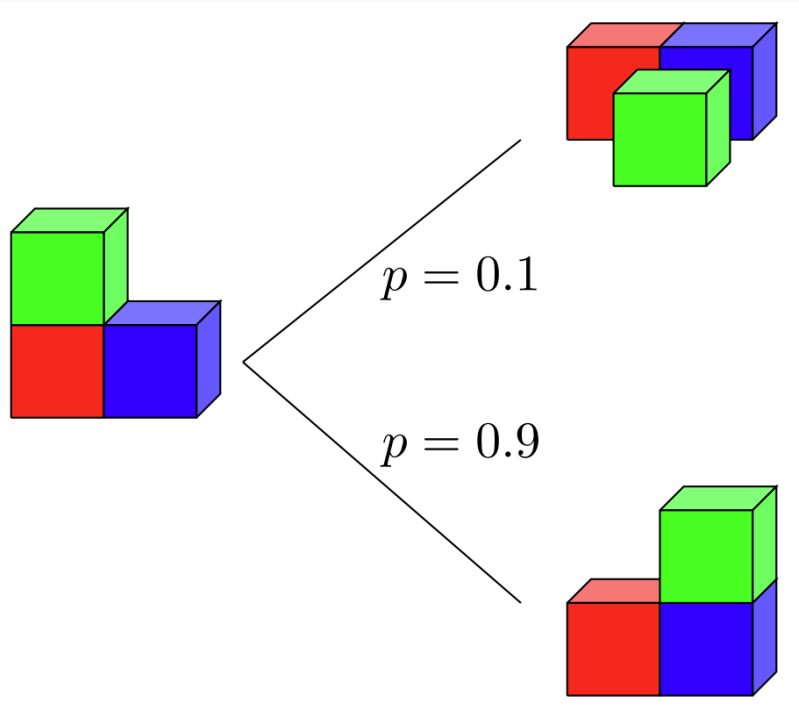
Nondetermistic dynamics example

Moving objects with an unreliable robotic hand: move the green block onto the blue block.



Probabilistic dynamics example

Moving objects with an unreliable robotic hand: move the green block onto the blue block.



Properties of the setting: observability

Full observability

Observations/sensing determine current world state uniquely.

Partial observability

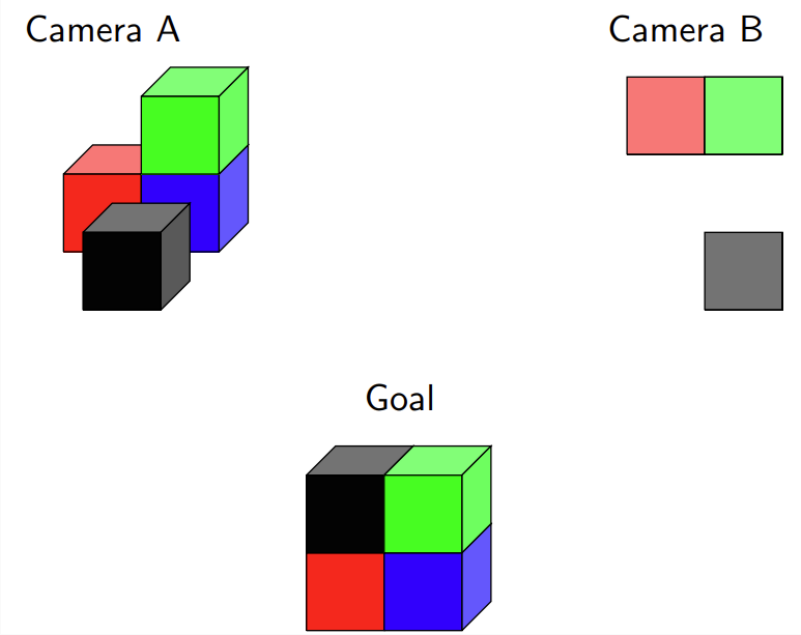
Observations determine current world state only partially: we only know that current state is one of several possible ones.

No observability

There are no observations to narrow down possible current states. However, can use knowledge of action dynamics to deduce which states we might be in (a.k.a funneling).

Consequence: If observability is not full, must represent the knowledge an agent has.

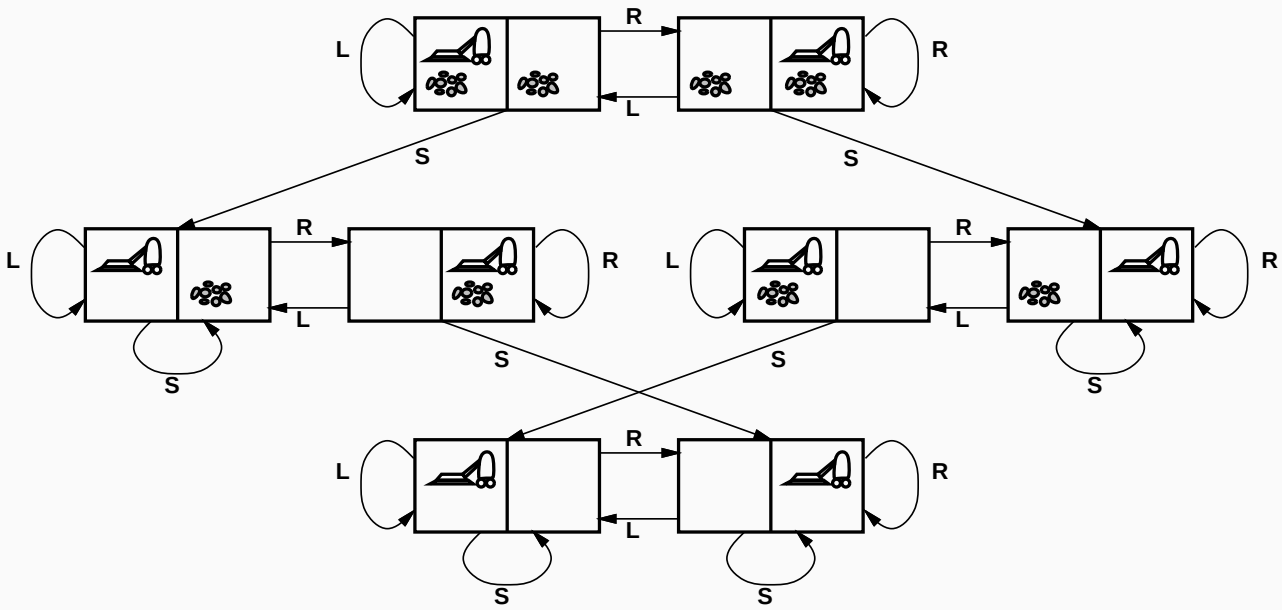
What difference does observability make?



Properties of the setting: different objectives

1. Reach a goal state.
   - **Example**: Earn 500 NIS.
2. Stay in goal states indefinitely (infinite horizon).
   - **Example:** Never allow the bank account balance to be negative.
3. Maximize the probability of reaching a goal state.
   - **Example:** To be able to finance buying a house by 2027 study hard and save money.
4. Collect the maximal *expected* rewards/minimal expected costs (infinite horizon).
   - **Example:** Maximize your future income.
5. ...

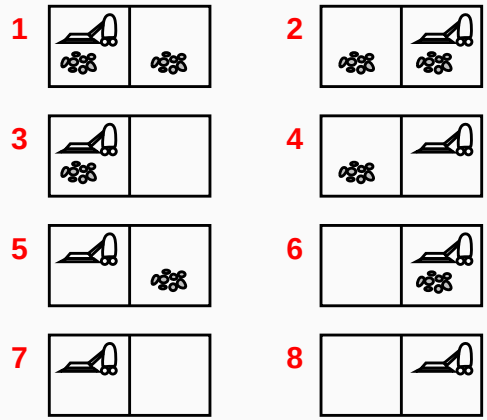
Example: vacuum world state space graph



states: 0/1 dirt and robot locations (ignore dirt amounts etc.)
actions: $Left$, $Right$, $Suck$, $NoOp$
goal test: no dirt
path cost: 1 per action (0 for $NoOp$)

Vacuum world: different classes of problems

Single-state, start in #5. Solution?



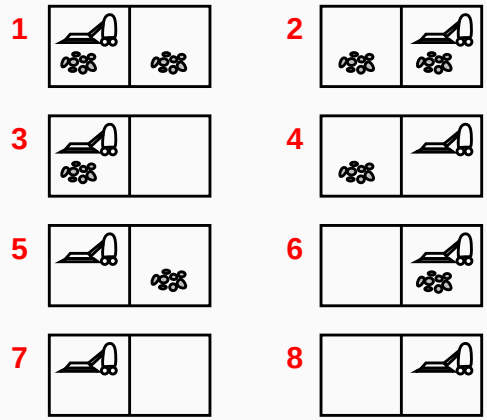
Vacuum world: different classes of problems

Single-state, start in #5. Solution?

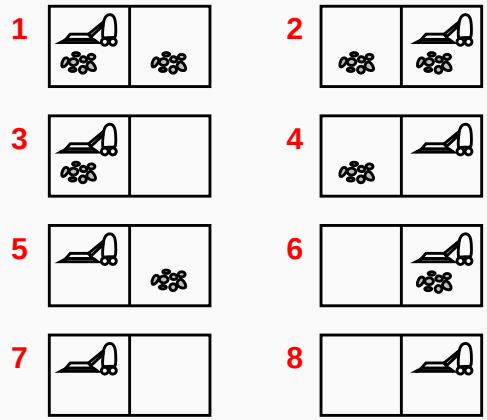
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$

e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution?



Vacuum world: different classes of problems



Single-state, start in #5. Solution?

$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$

e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution?

$[Right, Suck, Left, Suck]$

Conditional, start in #5

Murphy's Law: $Suck$ can dirty a clean carpet

Local sensing: dirt, location only.

Solution?

$[Right, if\ dirt\ then\ Suck]$

Formulation

Formulation

Typically, a model for AI decision-making comprises the following components:

- state space
- sensor function
- action space
- reward function
- horizon
- objective

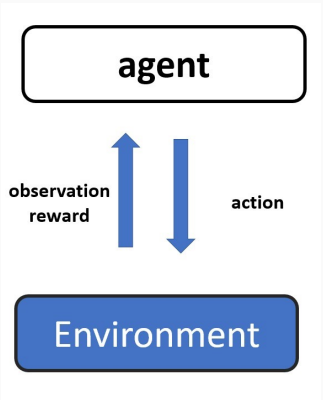
A model is an **abstraction** of the underlying environment.

Each model can be used by different decision-making procedures.

Environment: State and action space

- **State Space:** $s \in \mathcal{S}$ is a world state (more accurately, a representation of a world state)
  - Typically, it is impossible / impractical to explicitly the full state space.
  - We therefore use a **factored representation** in which the state space is described via a set of variables $\mathcal{X} = X_1, \ldots, X_n$, and a state is an assignment of a value $X_i \in Dom(X_i)$ for each variable X_i.
- **Action space:** $a \in \mathcal{A}$ is an action agents can perform
  - can have deterministic / non-deterministic / stochastic effects
  - may be associated with **preconditions** and **effects**.
    - For example: a gripper can pick up an object only if its free (precondition) and will not be free after the action is successfully executed (effect).

Reward



- **Reward:** a signal passed from the environment to the agent (typically referred to as **cost** when reward is negative).
- A reward r_t is a scalar feedback signal which indicates how well agent is doing at step t.
- The reward signal is your way of communicating to the agent *what* you want achieved, not *how* you want it achieved.

https://deepmind.com/research/publications/2021/Reward-is-Enough

Reward (cont.)

- Often described by a reward function:
  - Depending on the setting, defined as $\mathcal{R}(s, a, s')$, $\mathcal{R}(s, a)$ or $\mathcal{R}(s)$
  - The reward can be stationary $\mathcal{R}(s, a, s') \in \mathbb{R}$ or non-stationary, in which case we consider the expectation $\mathcal{R}(s, a, s') = \mathbb{E}[r_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$
- An agent's objectives is to maximize it's **utility** $\mathcal{U}$ which can be defined in various ways. Examples:
  - the expected total reward (*return*)
  - the worst-case (min) reward
- Utility is sometimes known as *goal* - but we will refer to a goal as a state (or state set) an agent aims to reach.

https://deepmind.com/research/publications/2021/Reward-is-Enough

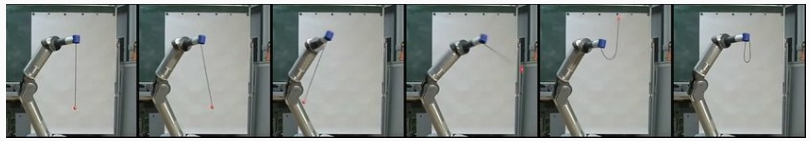
Reward (by David Sliver)

- Fly stunt manoeuvres in a helicopter
  - + for following desired trajectory
  - – for crashing
- Defeat the world champion at Backgammon
  - +/– reward for winning/losing a game
- Manage an investment portfolio
  - + reward for each $ in bank
- Control a power station
  - + reward for producing power
  - - reward for exceeding safety thresholds
- Make a humanoid robot walk
  - + reward for forward motion
  - - reward for falling over
- Play many different Atari games better than humans
  - +/– reward for increasing/decreasing score

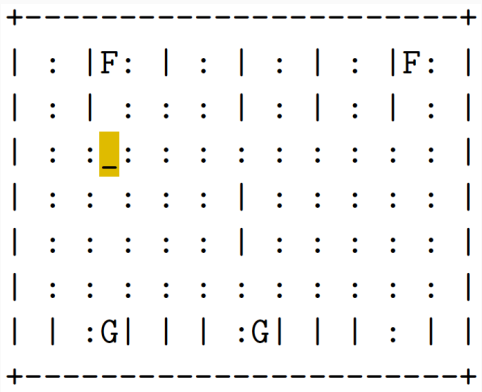
Problems with this approach ?

Reward

Designing a good reward function is an art.

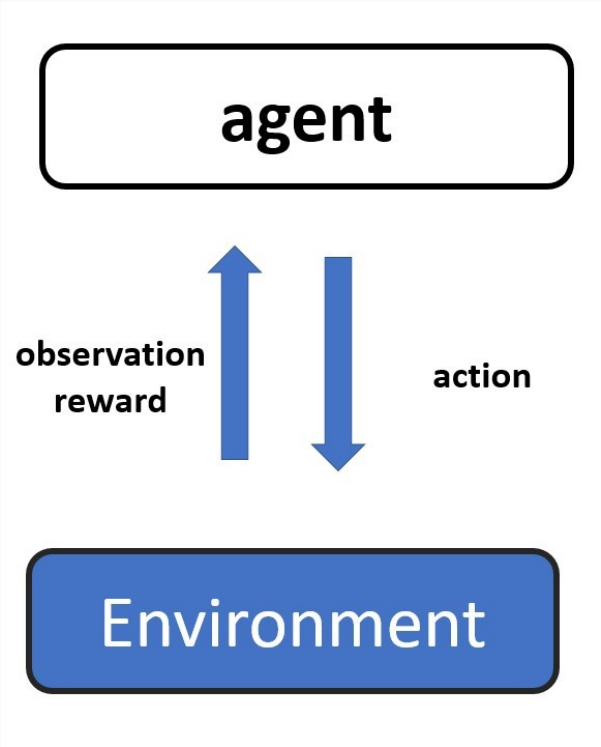


Fantasia 1940 The Sorcerer's Apprentice Walt Disney Cartoon Movie:
https://youtu.be/Rrm8usaH0sM
Ball in cup image from Kober et al (2009)



The Agent-Environment Interface

- An agent operates in the environment by taking actions.
- An agent (and its behavior) is characterized by its:
  - **Objectives (reward and utility)**
  - Ability to **observe** and **sense** the environment (observability and beliefs)



The Agent-Environment Interface

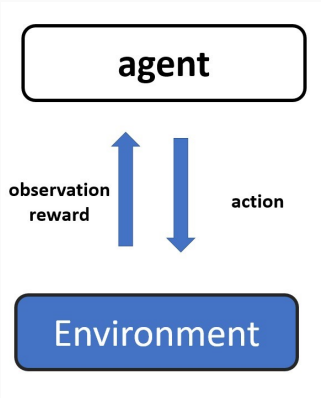
The agent and environment interact at each of a sequence of discrete time steps $t = 1, 2, 3, ...$
At each time step t:

- The agent receives some representation of the environment's state $s_t \in \mathcal{S}$
- Selects an action $a_t \in \mathcal{A}(s)$

One time step later $t + 1$:

- As a consequence of its action, the agent receives a numerical reward $r_{t+1} \in R$
- Finds itself in a new state s_{t+1}

The environment and agent together thereby give rise to a sequence or **trajectory**: $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...$

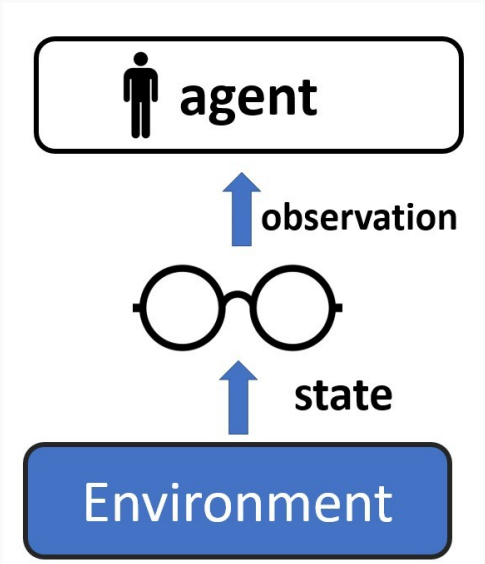


The Agent-Environment Interface

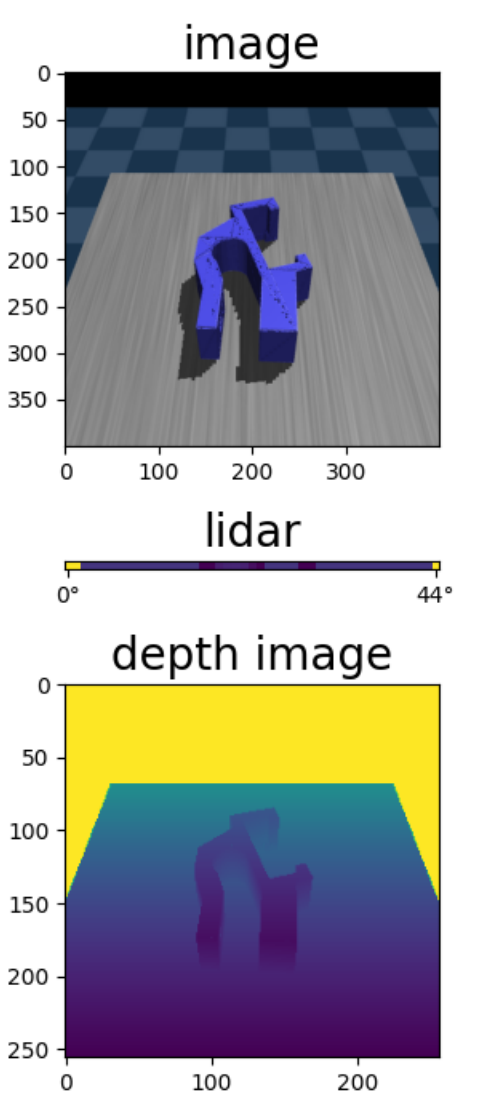
- **Episodes:** when the agent–environment interaction breaks naturally into sub-sequences. Each episode ends in a special state called the *terminal state*.
- **Episodic tasks:** Tasks with episodes. The time of termination T is a random variable that normally varies from episode to episode.
- **Continuing tasks:** When the agent–environment interaction does not break naturally into identifiable episodes, but goes on continually without limit.

Real-world examples of episodic vs. continuing tasks ?

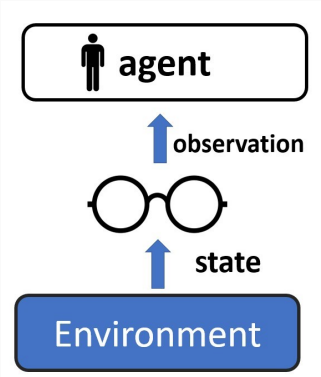
Accounting for Partial Observability: Observation and Belief



Accounting for Partial Observability: Observation and Belief



Observation and Belief



- At each time step the agent receives an observation $o_t \in \Omega$ that reflects the current state of the world.
- A **belief** $\beta \in \mathcal{B}$ represents the possible world states (i.e., the states that are deemed possible by the agent).
- Induced by an **observability/ sensor function** $\mathcal{O}^o_{s,a}$ that maps states to observations:
  - non-deterministic $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}[\Omega]$
  - probabilistic: $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$

Belief

A belief $\beta \in \mathcal{B}$ can have different representations:

- **Deterministic:** a direct mapping from states to beliefs $\beta = s$ (in which case we simply talk about states)
- **Non-deterministic:** $\beta \subseteq \mathcal{S}$ represents the set of possible world states
- **Stochastic:** the belief is a probability distribution over possible underlying world states. $\beta : \mathcal{S} \rightarrow \mathcal{P}[\mathcal{S}]$ such that $\beta(s)$ represents the probability that s is the actual world state.

Recall that the environment-agent interactions give rise to a sequence or trajectory: $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...$

From the point of view of the partially informed agent, we have a **history**: $o_0, a_0, r_1, o_1, a_1, r_2, o_2, a_2, r_3, ...$

The agent needs to maintain its belief based on the current observation.

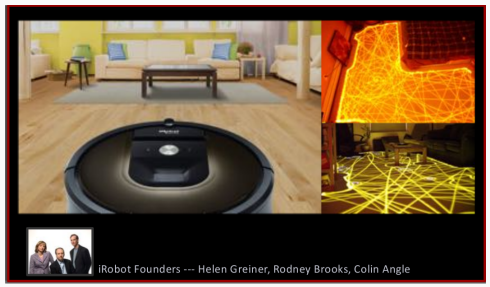
Policy (for a single agent)

- A policy is a mapping from states to actions.
- More accurately: we seek a mapping from the agent's understanding of the state, i.e., its **belief**, to actions.
- Our objective is to find a **(sub)-optimal policy** for an agent to follow in order to achieve it's objective / maximize its **utility**.

Formal definitions: Policy

- A **deterministic** policy is a mapping:
  - $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions
  - $\pi : \mathcal{B} \rightarrow \mathcal{A}$ from beliefs to action
  - $\pi(s)/\ \pi(\beta)$ is the (single) action the agent will perform at state s or belief β.
- A **non-deterministic** policy is a mapping:
  - $\pi : \mathcal{S} \rightarrow \mathcal{A}^n$ from states to actions,
  - $\pi : \mathcal{B} \rightarrow \mathcal{A}^n$ from beliefs to actions
  - $\pi(s)/\ \pi(\beta)$ is the set of actions one of which the agent will perform at state s or belief β.
- A **stochastic** policy is a mapping:
  - $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ from states to actions
  - $\pi : \mathcal{B} \times \mathcal{A} \rightarrow [0, 1]$ from beliefs to action
  - $\pi(s, a)/\ \pi(\beta, a)$ is the probability the agent will perform action a at state $s/$ belief β.

Policies for Reactive Agents



SIMPLE ROOMBA

If BUMP = TRUE
THEN Turn (random direction/amt)

ELSE MOVE-STRAIGHT

https://www.youtube.com/watch?v=7FSUtSurqA4

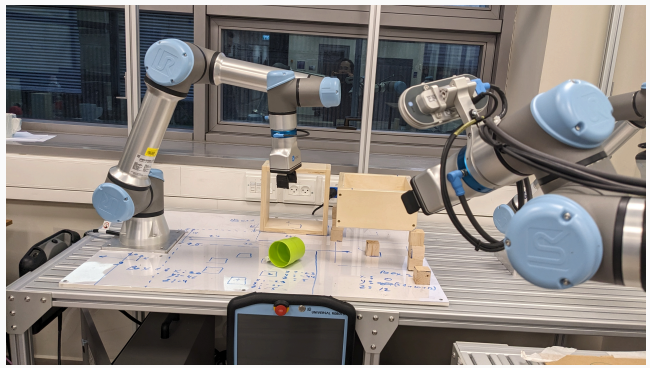
We will focus on rational agents.

Policies for Rational Agents

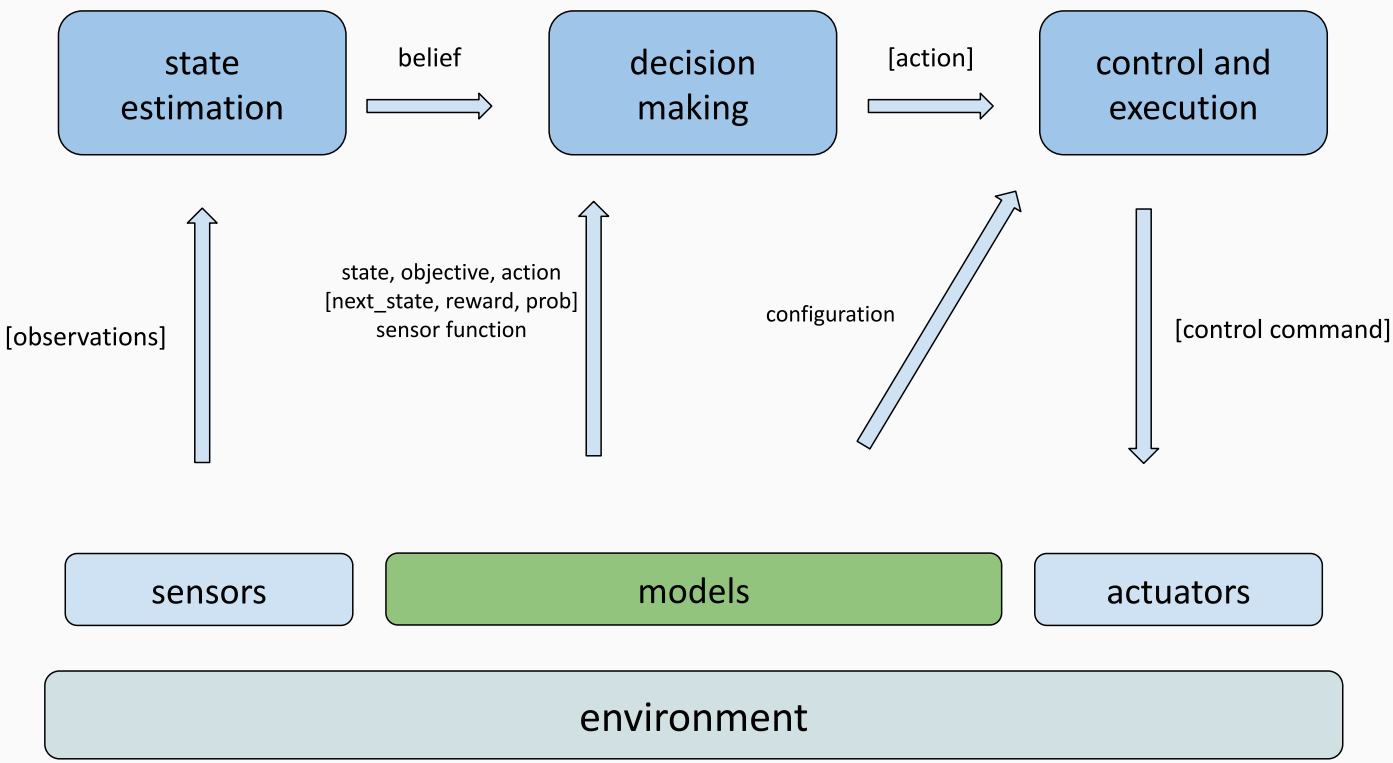
- Typically cannot be described explicitly for each state and require a compact representation.
- An **optimal policy**, denoted π^* is *better* than or equal to all other policies.
- Since our focus is on rational agents with a clearly defined utility measure, an optimal policy maximizes the agent's utility function $\mathcal{U}$.

Discussion

How to model these settings? What are the states, actions, rewards, and observations? What is the objective?



The Big Picture

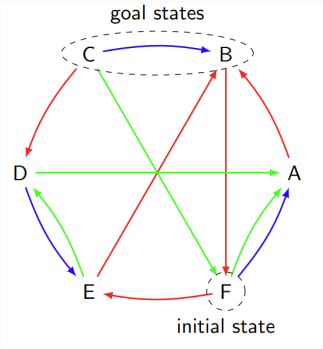


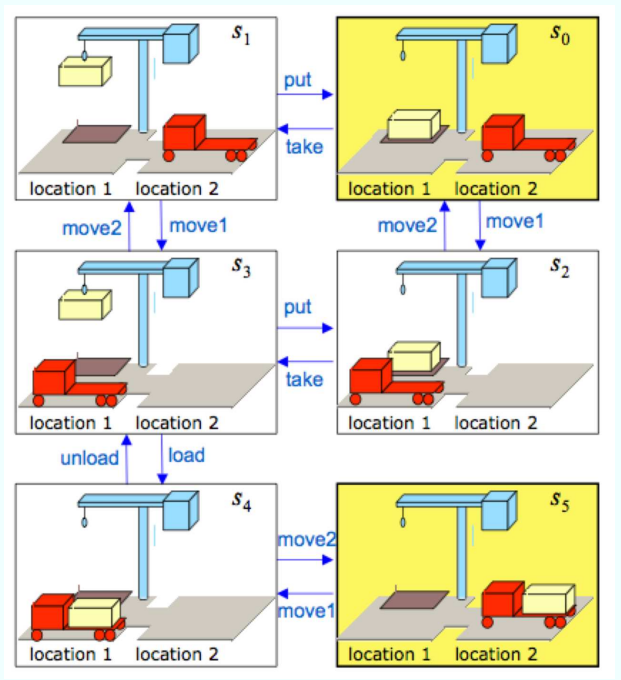
Deterministic Fully Observable Domains

State Model for Deterministic planning

- finite state space $\mathcal{S}$ (typically defined via a feature space $\mathcal{X}$).
- an initial state $s_0 \in \mathcal{S}$
- a set $\mathcal{S}_G \subseteq \mathcal{S}$ of goal states
- applicable actions $\mathcal{A}(s) \subseteq \mathcal{A}$ for $s \in \mathcal{S}$
- a transition function $s' = f(a, s)$ for $a \in \mathcal{A}(s)$
- a cost function $\mathcal{C} : \mathcal{A} \to [0, \infty)$

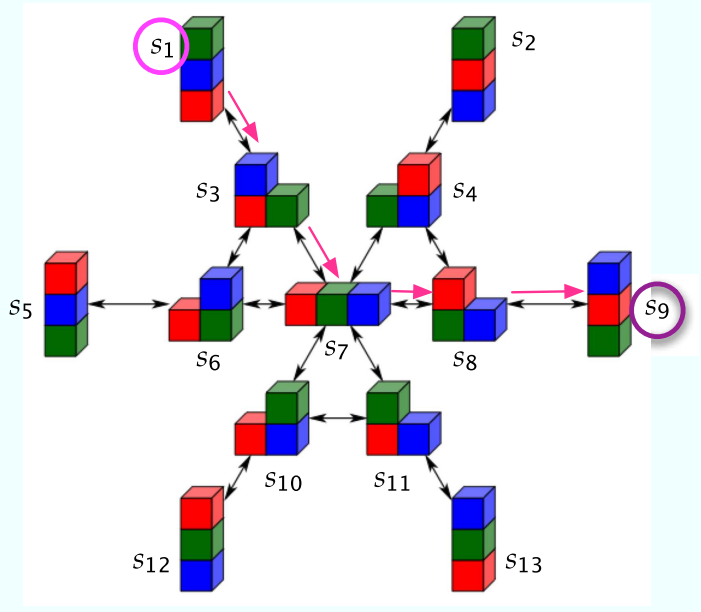
A **solution** is a sequence of applicable actions that maps s_0 into $\mathcal{S}_G$
An **optimal solution** minimizes the accumulated cost $\mathcal{C}$





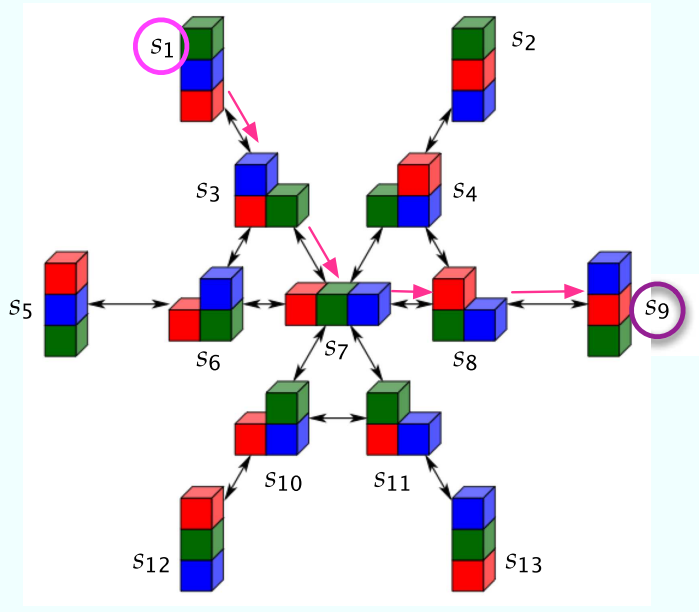
Why planning is difficult?

- Solutions to planning problems are paths from an initial state to a goal state in the transition graph
- Dijkstra's algorithm solves this problem in $O(|V| \log(|V|) + |E|)$
- Can we go home??



Why planning is difficult?

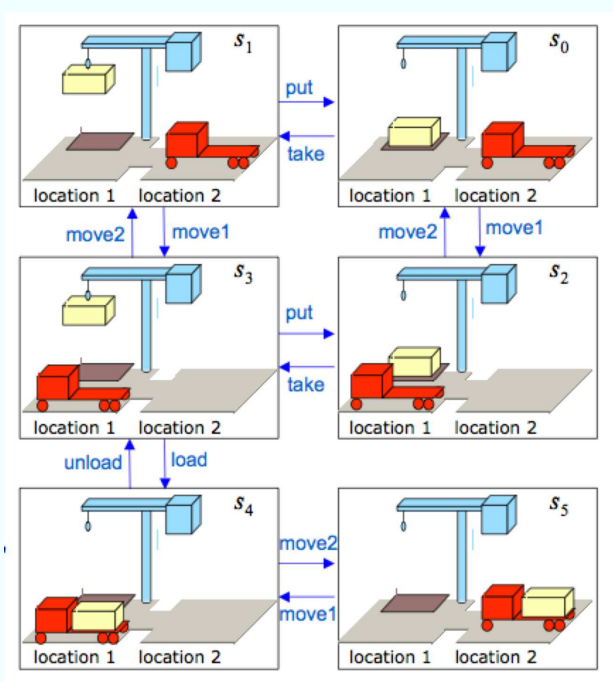
- Solutions to planning problems are paths from an initial state to a goal state in the transition graph
- Dijkstra's algorithm solves this problem in $O(|V| \log(|V|) + |E|)$
- Can we go home??
- ♠ Not exactly $\Rightarrow |V|$ of our interest is $10^{10}, 10^{20}, 10^{100}, \ldots$



- *But do we need such values of $|V|$?!*

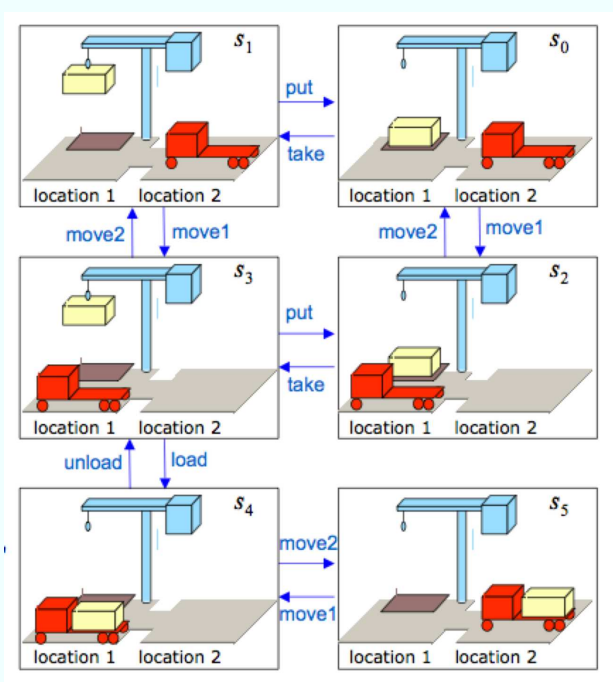
Why planning is difficult?

- Generalize the earlier example:
  - Five locations, three robot carts, 100 containers, three piles
  - $|V| \approx 10^{277}$
- The number of atoms in the universe is only about 10^{87}
  - The state space in our example is more than 10^{109} times as large (uppss ...)



Why planning is difficult?

- Generalize the earlier example:
  - Five locations, three robot carts, 100 containers, three piles
  - $|V| \approx 10^{277}$
- The number of atoms in the universe is only about 10^{87}
  - The state space in our example is more than 10^{109} times as large (uppss ...)



Good news:

Very much not hopeless!

Transition System and Classical Planning

- A transition system (or state space) is used to represent fully observable domains with deterministic actions.
- Used in classical automated planning
- A common way to represent classical planning problems is by using the STRIPS formalism [Fikes 1972]

From *https://ai.dmi.unibas.ch/misc/tutorial_aaai2015/planning02.pdf*

Propositional STRIPS

Propositional STRIPS planning task defined by a tuple $P = \langle \mathcal{F}, I, \mathcal{A}, G, \mathcal{C} \rangle$, where

- $\mathcal{F}$ is a set of fluents and a state s is represented by the fluents that are true in s
- $I \subseteq F$ is the initial state - the set of fluents that are true at the initial state, while the others are set of false.
- $G \subseteq F$ represents the set of goal states by specifying the set of fluents that need to be true (while the others can be either true or false).
- $\mathcal{A}$ is a set of actions.
  - Each action is a triple $a = \langle pre(a), add(a), del(a) \rangle$, which represents the precondition, add, and delete lists respectively, all subsets of $\mathcal{F}$.
  - An action a is applicable in state s if $pre(a) \subseteq s$.
  - If action a is applied in state s, it results in a new state $s' = (s \setminus del(a)) \cup add(a)$.
- $\mathcal{C} : \mathcal{A} \to \mathbb{R}_0^+$ is a cost function that assigns each action a non-negative cost.

Propositional STRIPS

- The objective is to find a plan $\pi = \langle a_1, \ldots, a_n \rangle$: a sequence of actions that brings an agent from the initial state to a goal state.
- The cost $c(\pi)$ of a plan π is $\Sigma_{i=1}^{n}(C(a_i))$.
- Often, the objective is to find an optimal solution for P: a plan π^* that minimizes the associated cost.
- The literature is rich with different approaches developed to solve the planning problem (Bonet and Geffner 2013): more on this later on in the course.

From *https://ai.dmi.unibas.ch/misc/tutorial_aaai2015/planning02.pdf*

PDDL-STRIPS example

Domain:

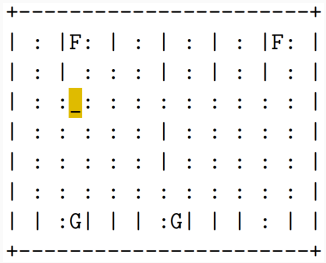
```
(define (domain gripper-strips)
  (:predicates (room ?r)  (ball ?b) (gripper ?g) (at-robby ?r)
               (at ?b ?r) (free ?g) (carry ?o ?g))
  (:action move
   :parameters (?from ?to)
   :precondition (and  (room ?from) (room ?to) (at-robby ?from))
   :effect (and (at-robby ?to) (not (at-robby ?from))))
  (:action pick
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                      (at ?obj ?room) (at-robby ?room) (free ?gripper))
   :effect (and (carry ?obj ?gripper) (not (at ?obj ?room))
                (not (free ?gripper))))
  (:action drop
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                      (carry ?obj ?gripper) (at-robby ?room))
   :effect (and (at ?obj ?room) (free ?gripper)
                (not (carry ?obj ?gripper)))))
```

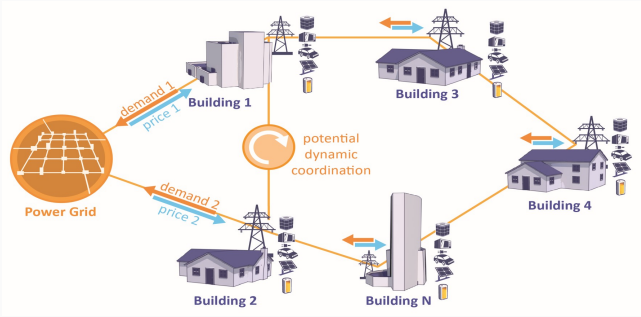
Problem:

```
(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma)
         (room roomb)
         (ball ball1)
         (ball ball2)
         (gripper left)
         (gripper right)
         (at-robby rooma)
         (free left)
         (free right)
         (at ball1 rooma)
         (at ball2 rooma))
  (:goal (at ball1 roomb)))
```

A collection of tools for working with planning domains can be found at:
http://planning.domains/

How to model our domains as STRIPS ?



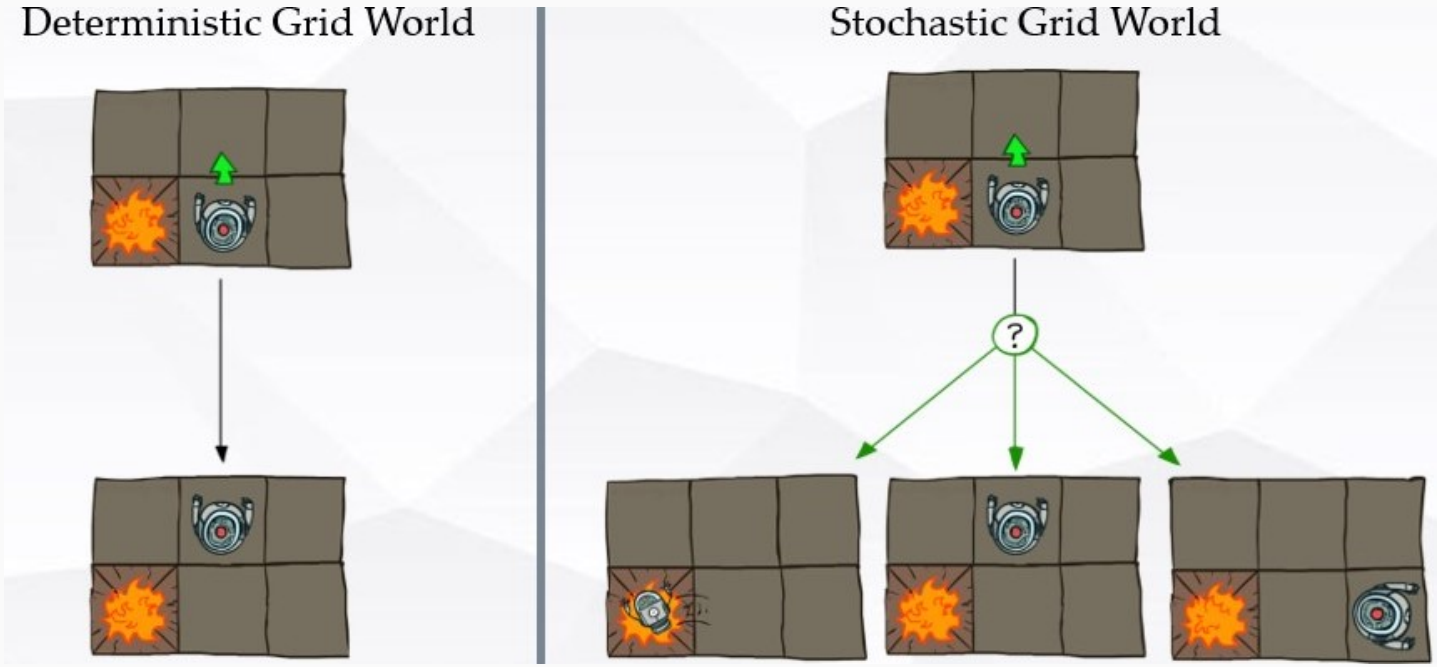


$$P = \langle \mathcal{F}, I, \mathcal{A}, G, \mathcal{C} \rangle$$

Is this model appropriate? Limitations? Strengths?

Accounting for Stochastic Actions

From Deterministic to Stochastic Domains



Markov Decision Process (MDP)

MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards.

Markov Decision Process

A **Markov Decision Process**(MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

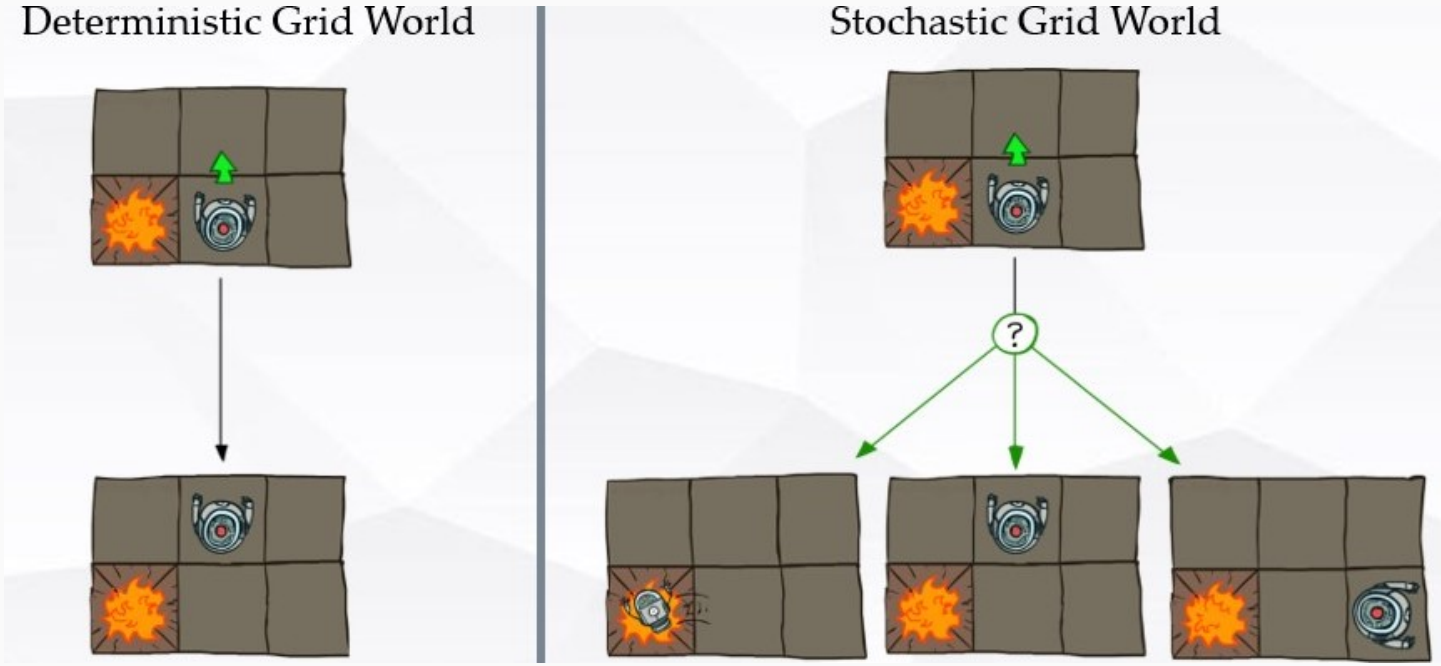
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\mathcal{P}^a_{s,s'} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$, and
- **optional:** γ is a discount factor $\gamma \in [0, 1]$ that is used to favor immediate rewards over future rewards.

The Markov property: "The future is independent of the past given the present".

Extensions: Infinite and continuous MDPs, partially observable MDPs, undiscounted, average reward MDPs. etc.

State Space

- The action space is typically (!) limited, but the state space $\mathcal{S}$ can be very large.
- There is a lot of information that is shared between states.
- We need a compact way to represent the dynamics of the system.



Factored Representation of Markov Decision Process

- Instead of an explicit representation of the state space $\mathcal{S}$, it is common to use **factored state representations***, where the set of states is described via a set of random variables $\mathcal{X} = X_1, \ldots, X_n$, and where each variable X_i takes on values in some finite domain $Dom(X_i)$.
- A state is an assignment of a value $X_i \in Dom(X_i)$ for each variable X_i.
- A **Markov Decision Process**(MDP) with a factored representation is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ in which the state space $\mathcal{S}$ is induced by $\mathcal{X}$.

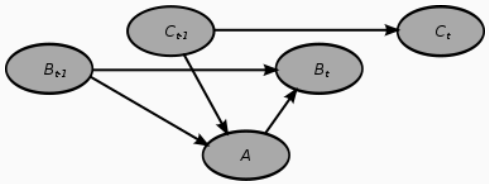
What's the benefit of using a factored representation ?

*Boutilier, Craig, Richard Dearden, and Moisés Goldszmidt. "Stochastic dynamic programming with factored representations." Artificial intelligence 121.1-2 (2000): 49-107 and Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. Journal of Artificial Intelligence Research, 19, 399-468.

Factored Representation of Markov Decision Process

Benefits:

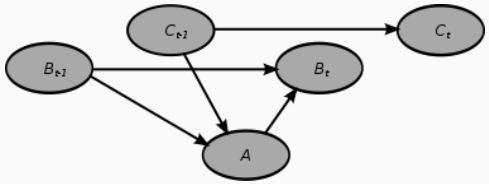
- Factored MDPs allow representing large, structured MDPs compactly.
- Transition dynamics can then be described compactly, e.g. using a Dynamic Bayesian network (DBN) (Dean and Kanazawa, 1989). This exploits the fact that the transition of a variable often depends only on a small number.
- Momentary rewards can often also be decomposed as a sum of rewards related to individual variables or small clusters of variables
- Allow using classifiers.



Factored Representation of Markov Decision Process

Two main types of structures that can (simultaneously) be exploited in factored MDPs:

- **Additive structure**: captures the fact that typical large-scale systems can often be decomposed into a combination of locally interacting components.
- **Context-specific structure** encodes a different type of locality of influence: Although a part of a large system may be influenced by all other parts, at any given point in time only a small number of parts may influence it directly.



Example

- **Additive structure**:
  - A large factory with many production cells. Faulty production affects whole factory, but the quality of the parts a cell generates depends directly only on the state of this cell and the quality of the parts it receives from neighboring cells.
  - Additive reward: factory operation cost depends on the sum of the costs of maintaining each local cell.
- **Context-specific structure**
  - In the factory example, a cell responsible for anodization may receive parts directly from any other cell in the factory. However, a work order for a cylindrical part may restrict this dependency only to cells that have a lathe. Thus, in the context of producing cylindrical parts, the quality of the anodized parts depends directly only on the state of cells with a lathe.

Terminology (from Sutton and Barto 2018)

- Expected return for episodic tasks:

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T = \sum_{k=0}^{T} r_{t+k+1}$$

- Expected return for continuing tasks:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

  when γ is the discount factor.

- Returns at successive time steps are related to each other:

$$G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \ldots) = r_{t+1} + \gamma G_{t+1}$$

Policy

A solution to an MDP is a **policy** that prescribes the action to be taken at every state.

- **Deterministic Policy**: a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions.
- **Stochastic Policy**: a mapping $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ from state and action pairs to to the probability $\pi(a|s)$ of taking action a when in state s.

Value Functions

Value functions estimate how 'good' it is for the agent to be in a given state or how good it is to perform a given action in a given state.

"How good" is defined in terms of expected return.

Since the rewards the agent can expect to receive in the future depend on what actions it will take, value functions are defined with respect to particular policies.

State-Value Function for Policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Action-Value Function for Policy π:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

What is G_t ?

Optimal Value Functions

Optimal State-Value Function:

$$v^*(s) \doteq \max_\pi v_\pi(s)$$

Optimal Action-Value Function:

$$q^*(s,a) \doteq \max_\pi q_\pi(s,a)$$

The relationship between the two :

$$q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma v^*(S_{t+1}) | S = s, A_t = a\right]$$

Discussion

- When are MDPs relevant ?
- Where do probabilities come from ?
- Isn't the Markovian assumption too restrictive ?

Accounting for Partial Information

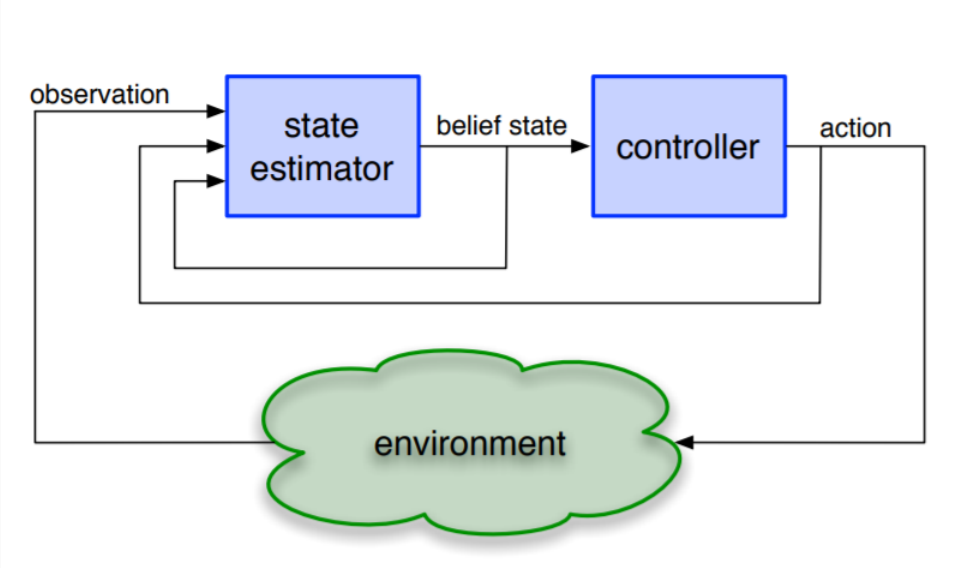
Partial Observability

Beliefs and Belief Tracking

- A **belief** represents the possible world states. $\beta : \mathcal{S} \rightarrow \mathcal{P}[\mathcal{S}]$ such that $\beta(s)$ represents the probability that s is the actual world state.
- In partially observable domains, we may have a **sensor model** represented as a mapping function from what is observed to the actual world state.
- A probabilistic sensor model is defined by the probability $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of observing o when action a is performed at state s.

Beliefs and Belief Tracking

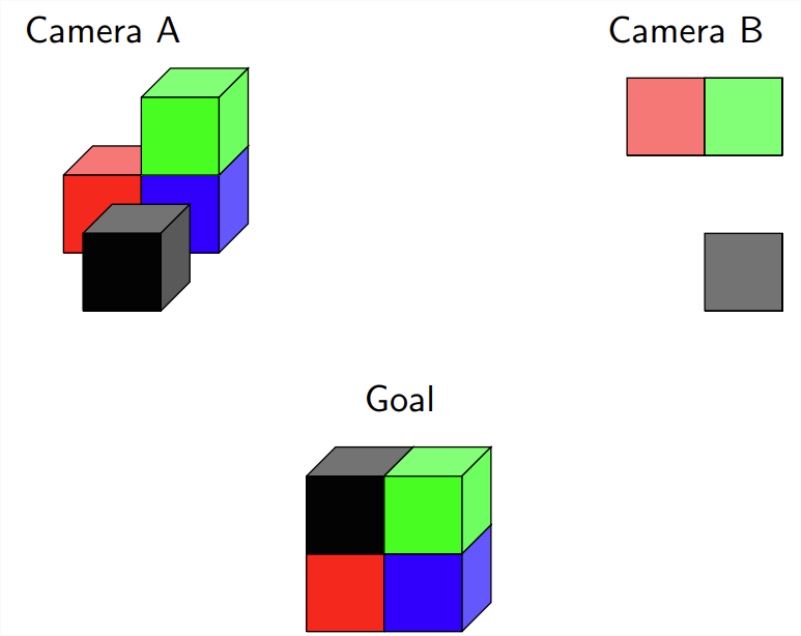
- The agent maintains its belief via a **state estimator** - which we will refer to as the process of **Belief Tracking**.



From Kaelbling, L. P., and T. Lozano-Perez. "Integrated Task and Motion Planning in Belief Space" 2013 *https://dspace.mit.edu/bitstream/handle/1721.1/87038/Kaelbling_Integrated%20task.pdf?sequence=1&isAllowed=y*

Planning Under Partial Observability with Logical Models

- The true state of the environment is not known, yet partial information about the state is assumed to be available from sensors.
- Uncertainty is represented by sets of possible world states, referred to as **beliefs**.



Planning Under Partial Observability with Logical Models

A *planning under partial observability problem* is a tuple $P = \langle \mathcal{F}, \mathcal{A}, G, \mathcal{O}, s_o \rangle$ where

- $\mathcal{F}$ is a set of fluent symbols,
- $\mathcal{A}$ is a set of deterministic actions,
- G is a set of $\mathcal{F}$-literals defining the goal condition, and
- $\mathcal{O}$ represents the agent sensor model,
- s_o is a set of *clauses* over $\mathcal{F}$-literals (referred to as *facts*) defining the initial situation

Planning Under Partial Observability with Logical Representations

- An action $a \in \mathcal{A}$ has a set $pre(a)$ of $\mathcal{F}$-literals as preconditions, and a set $eff(a)$ of conditional effects $C \rightarrow L$, where C is a set of $\mathcal{F}$-literals and L is an $\mathcal{F}$-literal.
- The sensor model $\mathcal{O}$ is a set of observations $o \in \mathcal{O}$ represented as pairs (C, L), where
  - C is a set of $\mathcal{F}$-literals, and
  - L is a positive fluent indicating that the value of L is observable when C is true.
- Each observation $o = (C, L)$ can be conceived as a sensor on the value of L that is activated when C is true.
  - e.g., I can see the color of a block when I am within 1 meter from it.

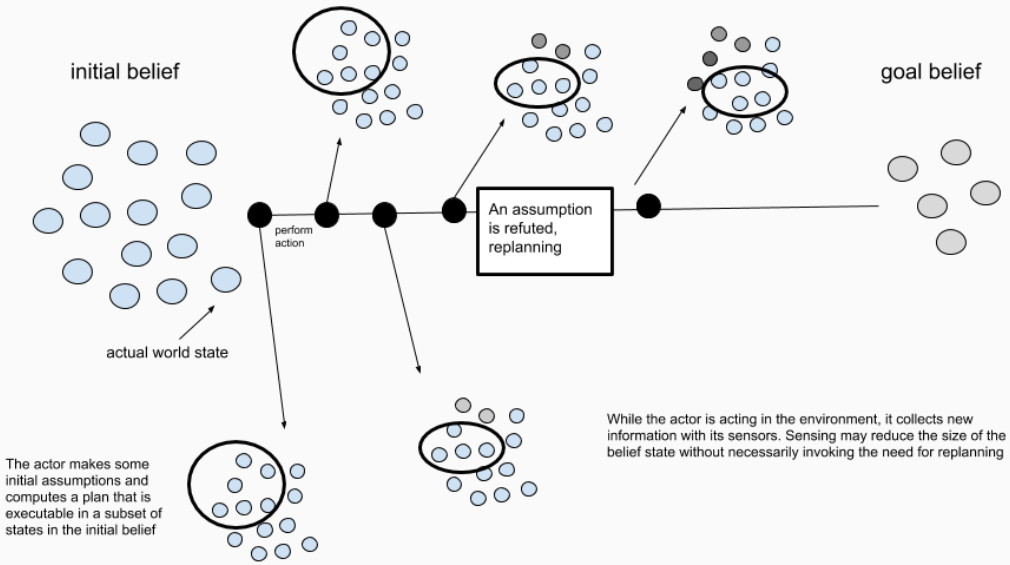
Planning Under Partial Observability with Logical Representations

- A state s is a truth valuation over the fluents $\mathcal{F}$ ('true' or 'false').
- For an agent, the value of a fluent may be known or unknown. A fluent is *hidden* if its true value is unknown.
- **A *belief* β is a non-empty collection of states that the agent deems as possible.**
- The initial belief is the set of states that satisfy s_o, and the goal belief is the set of those that satisfy g.

Planning Under Partial Observability with Logical Representations

- A formula $\mathbb{F}$ *holds* in b if it holds for every state $s \in b$.
- An action a is *applicable* in b if the preconditions of a hold in b, and the *successor* belief b' is the set of states that result from applying the action a to each state s in b.
- When an observation $o = (C, L)$ is activated, the successor belief is the set of states in b that agree on L (i.e., the set of states where fluent L has the sensed value).

Planning Under Partial Observability with Logical Representations



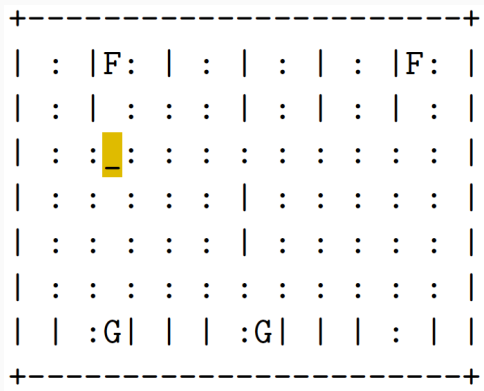
Partially Observable Markov Decision Process (POMDP)

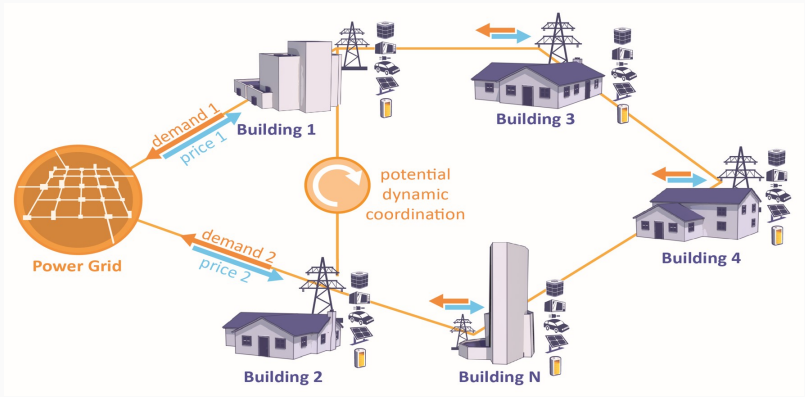
A **Partially Observable Markov Decision Process**(POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ where

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as for an MDP.
- Ω is a set of observations (observation tokens),
- $\mathcal{O}$ is a sensor function specifying the conditional observation probabilities $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of receiving observation token $o \in \Omega$ in state s after applying action a [1].
- β_0 the initial belief: a probability distribution over the states such that $\beta_0(s)$ stands for the probability of s being the true initial state.

[1] alternatively: $\mathcal{O}^o_s = \mathcal{P}[o_t = o | S_t = s]$

How to model various domains as POMDPs ?





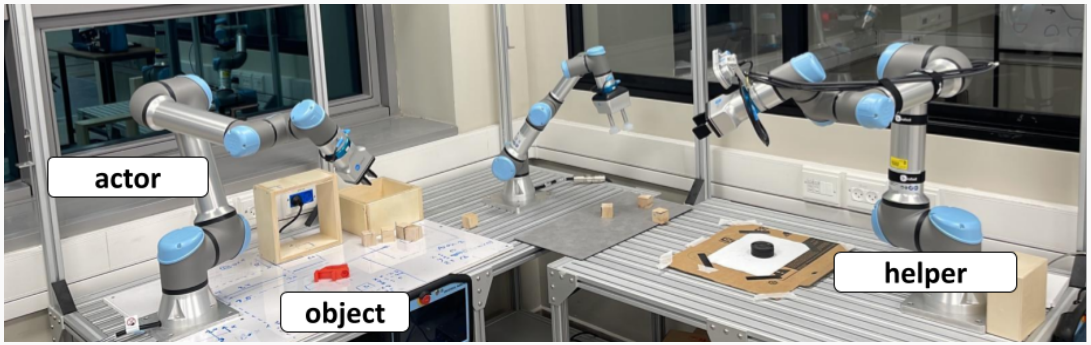
Belief Update

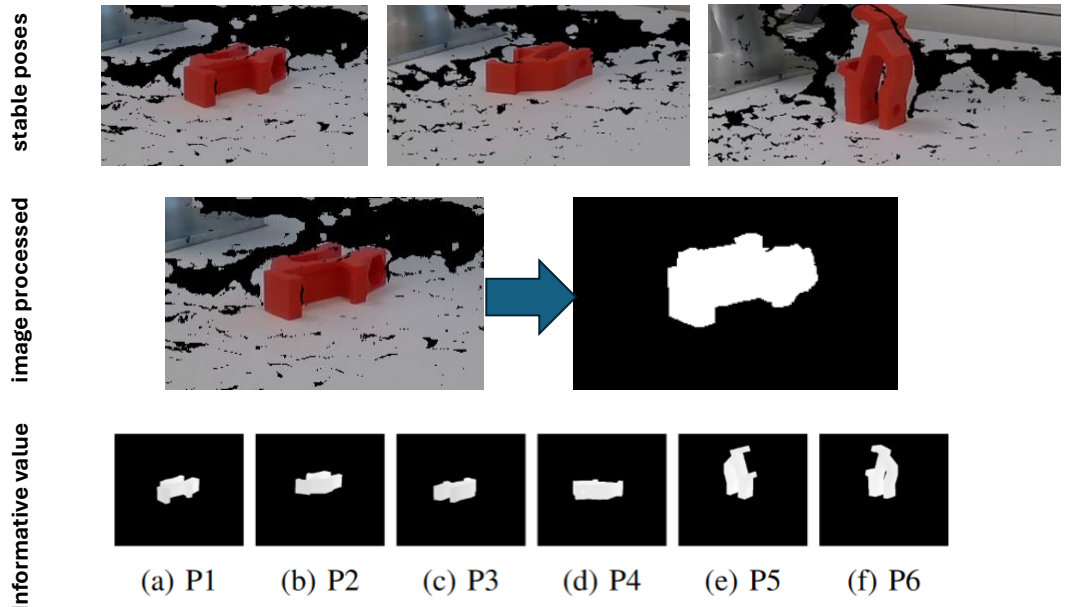
- In the logical model we keep track of the possible world states.

$$\beta_a^o = \{s | s \in \beta_a \text{ and } o \in \mathcal{O}(s, a)\}$$

- In the probabilistic model we keep track of the probability of all states.

$$\beta'(s') = \mathcal{P}(s' | o, a, \beta)$$





$$\beta^{o,m}(p) = \frac{\hat{P}(o|p,m)\,\beta(p)}{\int_{p' \in \mathcal{P}} \hat{P}(o|p',m)\,\beta(p')\,dp'} \tag{1}$$

where $\beta(p)$ is the estimated probability that p is the pose prior to considering observation o.

Supervised Learning

Sample Set

In supervised learning:

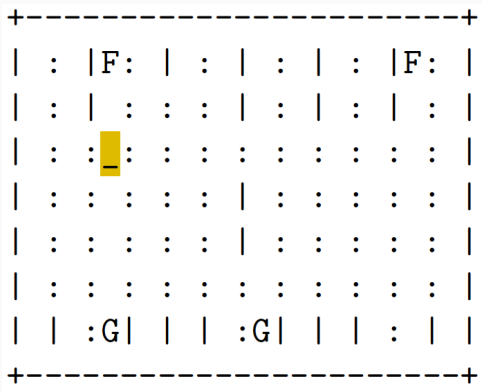
- A sample set: $S = \{(x_i, y_i)_{i=1}^m\}$
- Typically used to learn for an appropriate classifier (e.g., with lowest expected error or loss) among a hypothesis class H.

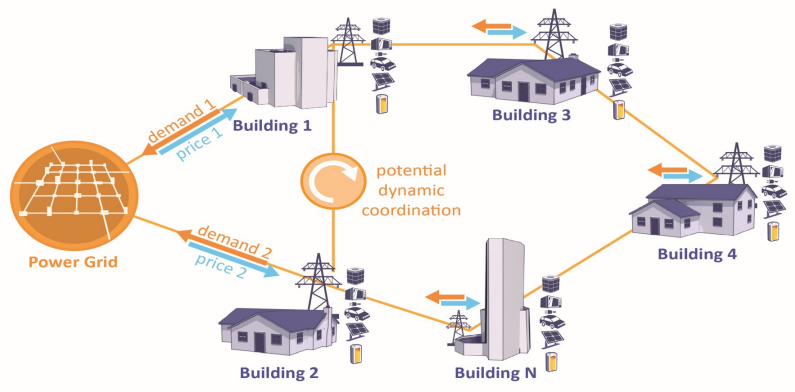
Learning a policy π:

- A sample set: $S = \{(s_i, a_i)_{i=1}^m\}$ or $S = \{(\beta_i, a_i)_{i=1}^m\}$
- Typically, a parameterized policy over a factored state-space representation is used.
- Parameterized policy as a conditional probability $\pi_\theta(a_t|\beta_t)$ or $\pi_\theta(a_t|s_t)$ (for fully observed)

Sample set representation of our domains ?

A sample set: $S = \{(s_i, a_i)_{i=1}^m\}$ or $S = \{(\beta_i, a_i)_{i=1}^m\}$





Effective for sequential decision making ?

Do we need the MDP ?

Discussion

- Logical vs. probabilistic models for partial observability
- Do we need to model the sensor function (and use a POMDP) or can we use an MDP ?

Fig. 1: Our method learns visuomotor policies that directly

End-to-End Training of Deep Visuomotor Policies Levine et al. 2016

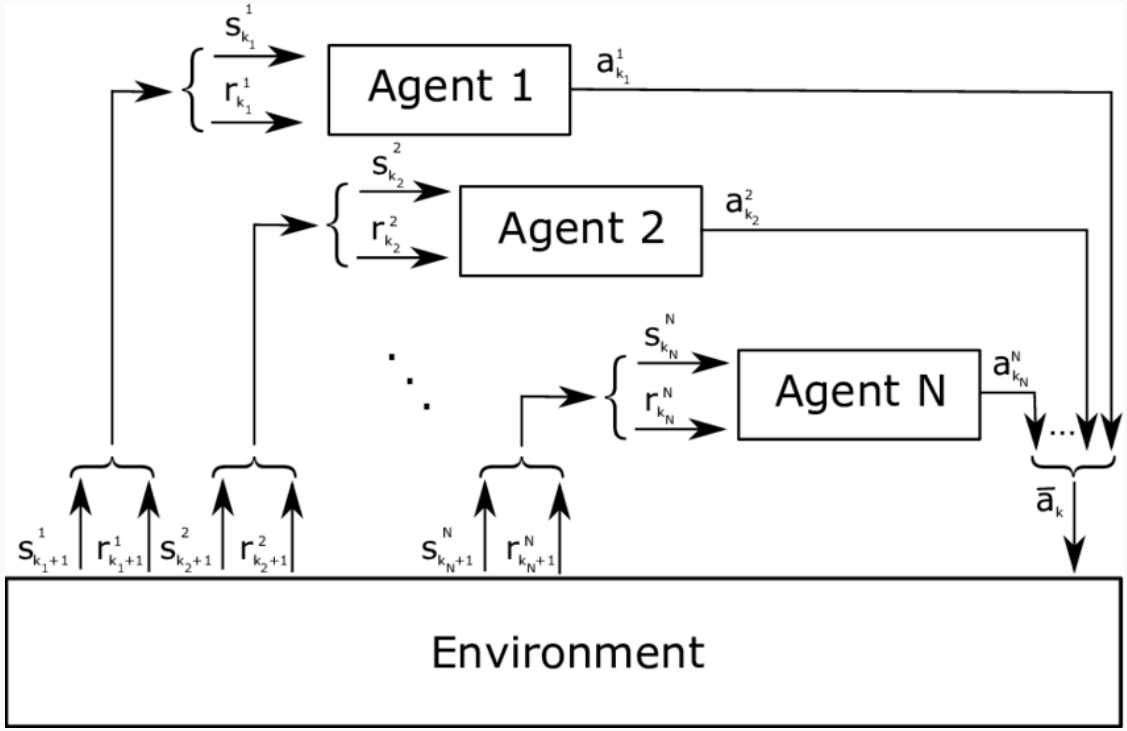
https://www.jmlr.org/papers/volume17/15-522/15-522.pdf

Multi-agent Settings

From Single to Multi-Agent Settings

What changes in the **model** when we consider the other agents in the environment?

The Multi-agent-Environment Interface



To model a multi-agent setting, we will use a **Multi-Agent MDP**, or *Markov game*, or *stochastic game* which is a generalization of the MDP to multi-agent settings (Littman'94).

Image by Michele Chincoli and Antonio Liotta

Multi-Agent Markov Decision Process - Markov Games

A **Multi-Agent MDP** is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states

Multi-Agent Markov Decision Process - Markov Games

A **Multi-Agent MDP** is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A} = \{\mathcal{A}^i\}_{i=1}^n$ are the **joint actions**: a collection of action sets $\mathcal{A}^i$, one for each agent in the environment. At each timestep t, each agent i chooses an action $a_t^i \in \mathcal{A}^i$. The actions of all N agents are combined to form a **joint action** $\mathbf{a}_t = [a_t^0, \ldots, a_t^N]$.

Multi-Agent Markov Decision Process - Markov Games

A **Multi-Agent MDP** is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A} = \{\mathcal{A}^i\}_{i=1}^n$ are the **joint actions**: a collection of action sets $\mathcal{A}^i$, one for each agent in the environment. At each timestep t, each agent i chooses an action $a_t^i \in \mathcal{A}^i$. The actions of all N agents are combined to form a **joint action** $\mathbf{a}_t = [a_t^0, \ldots, a_t^N]$.
- $\mathcal{P} = \{\mathcal{P}^i\}_{i=1}^n$ describes the **joint probability** distribution over next states when a joint action is performed and produces a transition in the environment, where $\mathcal{P}_{s,s'}^{\mathbf{a}} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = \mathbf{a}_t]$ describes the probability of ending at state s' when the joint action $\mathbf{a}_t \in \mathcal{A}$ is performed at state s.

Multi-Agent Markov Decision Process - Markov Games

A **Multi-Agent MDP** is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A} = \{\mathcal{A}^i\}_{i=1}^n$ are the **joint actions**: a collection of action sets $\mathcal{A}^i$, one for each agent in the environment. At each timestep t, each agent i chooses an action $a_t^i \in \mathcal{A}^i$. The actions of all N agents are combined to form a **joint action** $\mathbf{a}_t = [a_t^0, \ldots, a_t^N]$.
- $\mathcal{P} = \{\mathcal{P}^i\}_{i=1}^n$ describes the **joint probability** distribution over next states when a joint action is performed and produces a transition in the environment, where $\mathcal{P}_{s,s'}^{\mathbf{a}} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = \mathbf{a}_t]$ describes the probability of ending at state s' when the joint action $\mathbf{a}_t \in \mathcal{A}$ is performed at state s.
- $\mathcal{R} = \{\mathcal{R}^i\}_{i=1}^n$ is a collection of rewards functions $\mathcal{R}^i$ defining the reward $r^i(a_t, s_t)$ each agent receives when the joint action a_t is performed at state s_t,

Multi-Agent Markov Decision Process - Markov Games

A **Multi-Agent MDP** is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A} = \{\mathcal{A}^i\}_{i=1}^n$ are the **joint actions**: a collection of action sets $\mathcal{A}^i$, one for each agent in the environment. At each timestep t, each agent i chooses an action $a_t^i \in \mathcal{A}^i$. The actions of all N agents are combined to form a **joint action** $\mathbf{a}_t = [a_t^0, \ldots, a_t^N]$.
- $\mathcal{P} = \{\mathcal{P}^i\}_{i=1}^n$ describes the **joint probability** distribution over next states when a joint action is performed and produces a transition in the environment, where $\mathcal{P}_{s,s'}^{\mathbf{a}} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = \mathbf{a}_t]$ describes the probability of ending at state s' when the joint action $\mathbf{a}_t \in \mathcal{A}$ is performed at state s.
- $\mathcal{R} = \{\mathcal{R}^i\}_{i=1}^n$ is a collection of rewards functions $\mathcal{R}^i$ defining the reward $r^i(a_t, s_t)$ each agent receives when the joint action a_t is performed at state s_t,

Multi-Agent Markov Decision Process

What about partially observability?

Multi-Agent Markov Decision Process - Markov Games

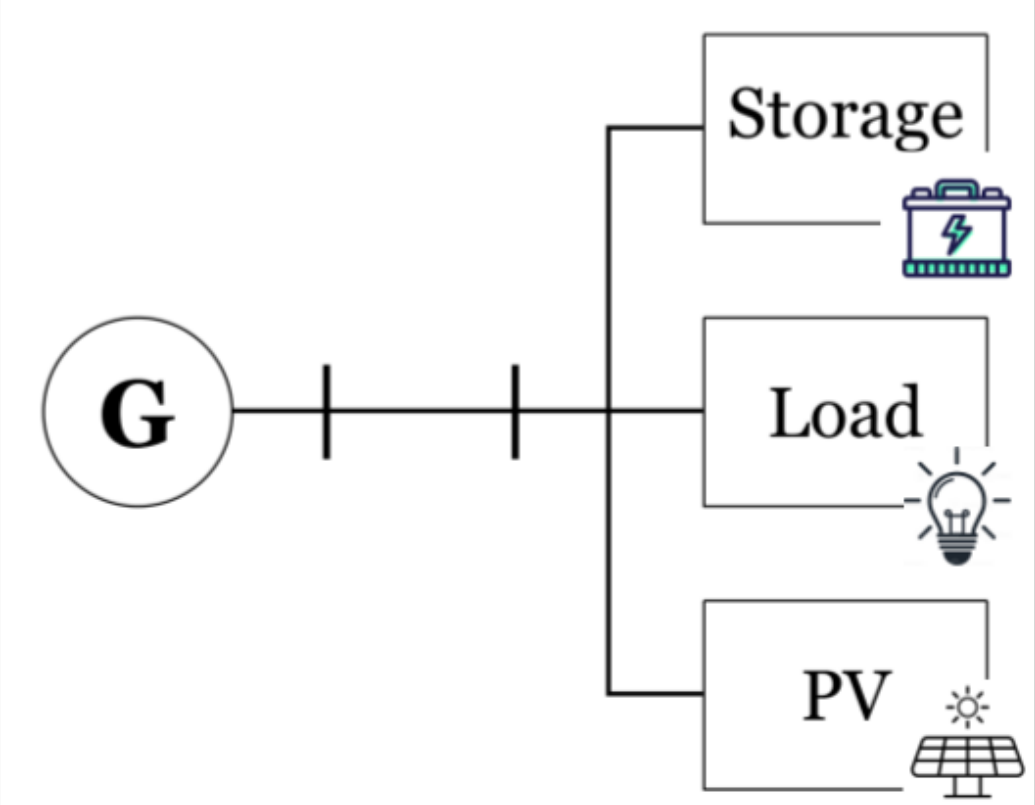
In partially observable environments the ith agent can only view a portion of the true state, s_t^i.



Multi-Agent POMDP

- A **Multi-Agent POMDP** can be defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathfrak{O}, \mathfrak{S}, \mathfrak{B}_0 \rangle$ where
  - $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as in the definition of the Markov Game,
  - $\mathfrak{O} = \{O^i\}_{i=1}^n$ are the **joint observations**: a collection of observation token sets O^i, one for each agent in the environment. At each timestep t, each agent i observes an observation $o_t^i \in O^i$. The observations of all N agents are combined to form a **joint observation** $\mathbf{o}_t = [o_t^0, \ldots, o_t^N]$.
  - $\mathfrak{S}$ is the **sensor function** describing the probability distribution over next joint observations when a joint action is performed in some state, where $\mathfrak{S}^{\mathbf{o}}_{s,\mathbf{a}} = \mathcal{P}[\mathbf{O}_{t+1} = \mathbf{o} | S_t = s, A_t = \mathbf{a}_t]$ describes the probability of receiving joint observation $\mathbf{o}$ when the joint action $\mathbf{a}_t$ is performed at state s (alternatively: $\mathfrak{S}^{\mathbf{o}}_{s} = \mathcal{P}[\mathbf{O}_{t+1} = \mathbf{o} | S = s]$.)
  - $\mathfrak{B}_0 = \{b_0^i\}_{i=1}^n$ is the initial joint belief, specifying for each agent i the probability distribution over states such that $b_0^i(s)$ stands for the probability agent i associates with s being the true initial state.

Discussion



What is the best model?

Summary

- Models for decision making in AI for single and multi-agent settings
- Limitations and benefits of each model