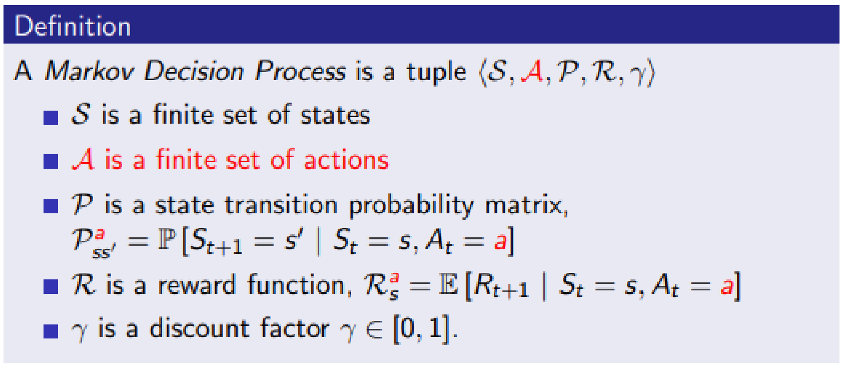
Week 0 Problem Statement

SoC: Street Fighter II – Reinforcement Learning

Since this is the first week, this problem will be optional. In the subsequent weeks, the Problem Statement will be mandatory (i.e., required for certification).

In this problem we would like to simulate a Markov Decision Process. MDPs are often visualised as state transition diagrams(recall the student MDP). Markov Decision Processes are defined by a tuple, as you know:



Definition

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

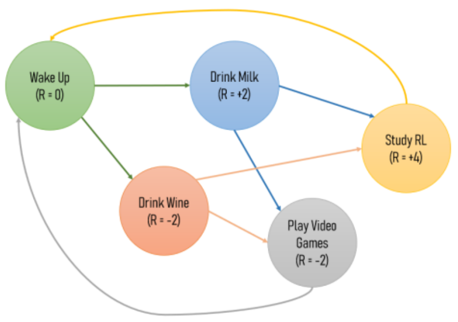
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

In MDPs, the reward function can be defined in two ways:

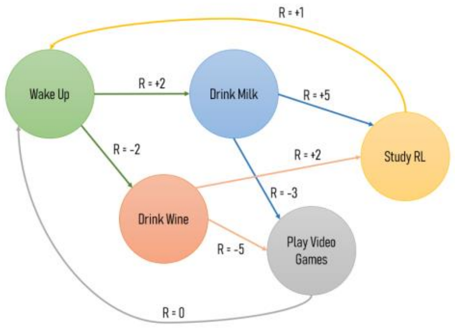
1. You can have the rewards associated with states. That means, on reaching a state X by any action, you will always receive a reward R_x.
2. You can have rewards associated with a particular transition in the state transition diagram. That means that if you reach X via an action A, you receive a reward $R_{a,x}$ but if you reach it via an action B, you may receive some other reward, say, $R_{b,x}$.

Look at the following diagrams:

Type(1) MDP:



Type(2) MDP:

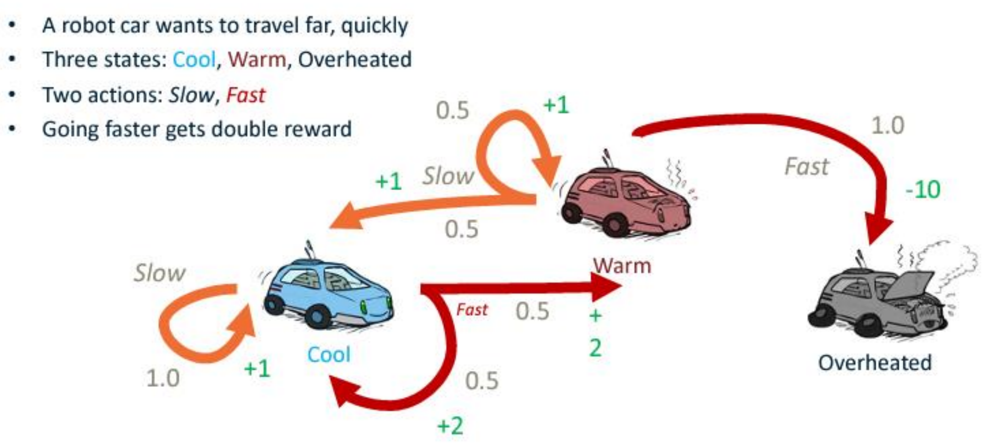


This is a subtlety that is easy to ignore, but important to appreciate. If you think about it, MDPs of type(1) are essentially MDPs of type(2) because you can reduce type(1) MDPs to type(2) by simply moving the rewards written on the state onto just every transition that moves you into the state.

However, from a coding point of view, type(1) can have a potentially different implementation in which each of the rewards is associated with the state class, rather than with the action class.

I have attached python code that simulates the first MDP diagram on the left, on the previous page. Go through that code. Notice that defining such classes helps us write readable code. If you are unfamiliar with Python Classes, go through the additional resources provided on the Classroom page.

Your job here is to simulate the following MDP:



- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

At each time-step, the car as the agent gets to decide whether it must go “Slow” or go “Fast”. Notice that when the car is in the Cool state, the agent’s decision to go Fast may result in the car finding itself in either Cool state or the Warm state with 0.5 probability. This means that the agent’s action still leaves scope for the environment to dispense a different next-state, based on chance.

On the other hand, a decision to go Fast at the Warm state results in the Overheated state with 100% probability, where the agent terminates its execution.

When the robot takes the Fast decision, it covers 2km of distance (see the +2 Reward? Here, let us model the distance covered as the reward, because that is what we want to maximise) whereas, the Slow decision makes it cover only 0.5km if the car is hot, but allows 1km if the car is cool. This is clearly a type(2) MDP.

The -10 reward however, does not signify any distance in kilometres. It just returns a large negative reward. The car just stalls once it is overheated and the program ends. This tells you that rewards are not necessarily a reflection of some aspect of the problem; they roughly just model the notions of *good* and *bad*.

I shouldn't be calling these as type(1) and type(2) MDPs. On the previous page, the diagram on the left represents a state machine which is called a Moore machine. And the second diagram on the right is an example of a Mealy machine.

Moore machines can be trivially converted into Mealy machines(How?*). The same cannot be said in the reverse direction. However, Mealy machines can indeed be converted into Moore Machines with the addition of a few states and then associating the rewards with these states. This conversion is non-trivial and not in the scope of RL, so I will omit this discussion. Feel free to read on the internet about this.

However, I will emphasize that due to their interconvertibility, the implementation of a Mealy-like MDP can be converted to a Moore-like MDP and vice-versa. In this problem, however, we target a Mealy MDP.

Pro Tips: Object Oriented Programming will come to your rescue. If you are unfamiliar with OOP in Python, check out the "Additional Resources" tab uploaded in Week 0 on Google Classroom. Ask doubts. Take my assistance in coding if needed.

Specification:

Python, C++ or Java code is allowed. Your program must take input from stdin (i.e., your program takes input from the command-line/console and not from, say, a text file, for example).

The inputs are a sequence of "Slow" and "Fast" strings, each on a new line. A bad input should terminate the program. The first line of the input gives the number of actions that will be taken.

On moving fast, the car can always cover 2km. If the car begins to move slowly from a Cool state, it moves for 1km, and if it does so from a Warm state it can only move 0.5km.

The task is for the program to determine if the input sequence can cross 20km without overheating. The last line of the output should only be one of the three strings "Overheated" or "Reached" or "Halted Midway". Once you have given one of these outputs, no need to process the rest of the input.

The car is initially in the Cool state. In the output, after each action that is read, output a string which says what state the car went into as "Warm" or "Cold".

Since you will be using Random number generators to handle the probability related issues, the same input might lead to different outputs unless you seed your generators. So, seed your generators. (Not sure what seeding is? Search the internet or ask me!)

Input example 1:

```
5
Slow
Fast
Slow
Slow
Fast
```

Possible Output for 1:

```
Cool
Warm
Cool
Cool
Warm
Halted Midway
```

Input example 2:

```
7
Slow
Fast
Slow
Fast
Slow
Fast
Slow
```

Possible Output for 2:

```
Cool
Warm
Warm
Overheated
```

Possible Output for 2:

```
Cool
Warm
Cool
Warm
Cool
Warm
Cool
Halted Midway
```

Submission Details:

Submit only two files `carMDP.py` or `carMDP.cpp` or `carMDP.java` on Google Classroom.

Submit desc.txt describing your code implementation and any sources you referred to that were not explicitly given by me. This includes GPT and stackoverflow sources.

Feel free to use the internet or talk to me; this is not a semester course; it's a learning project. Although this has no deadline; I will use this same Car MDP next week for a theory problem and also for a coding problem, so if you implement this before the end of Week 1, you will be good to go.