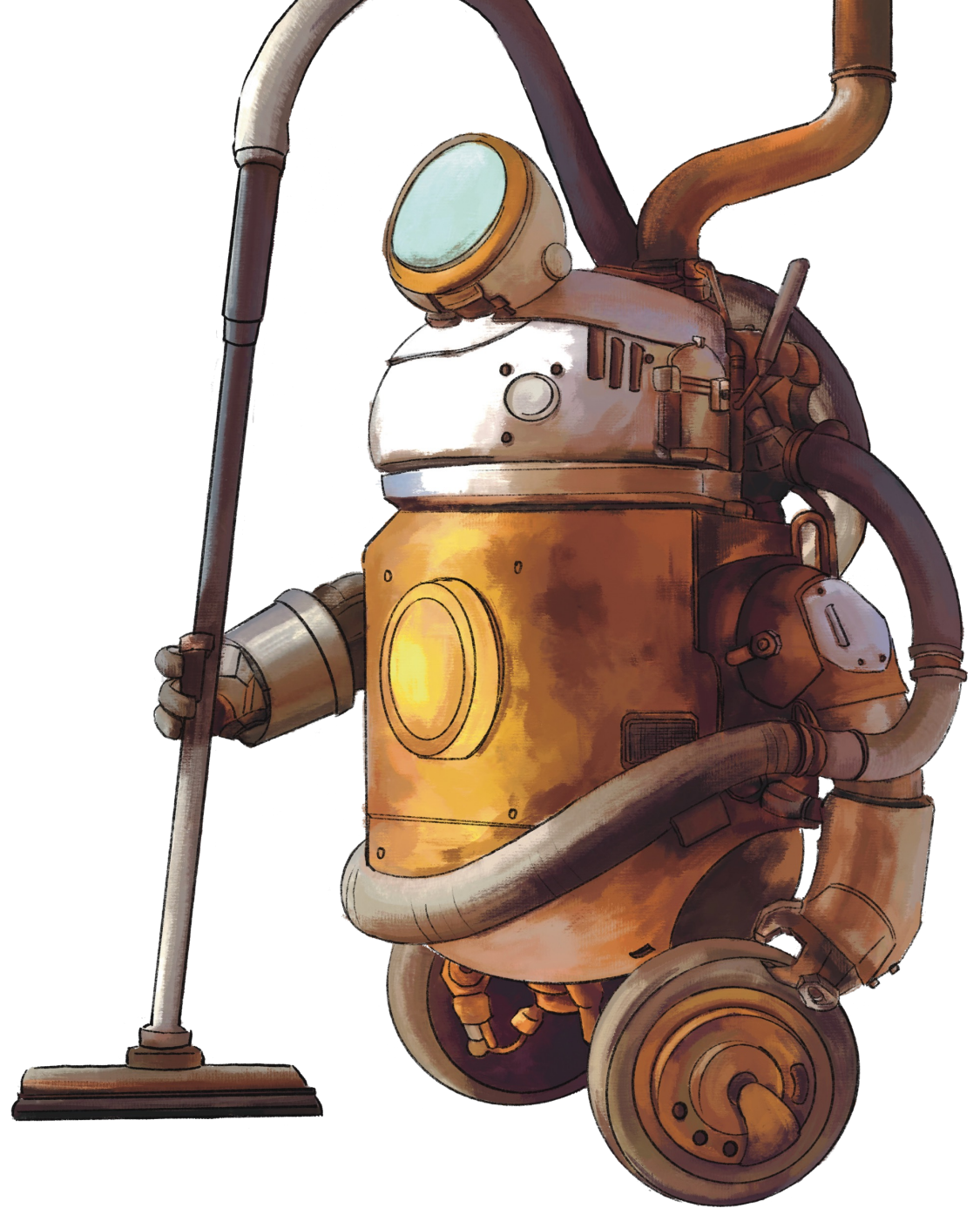


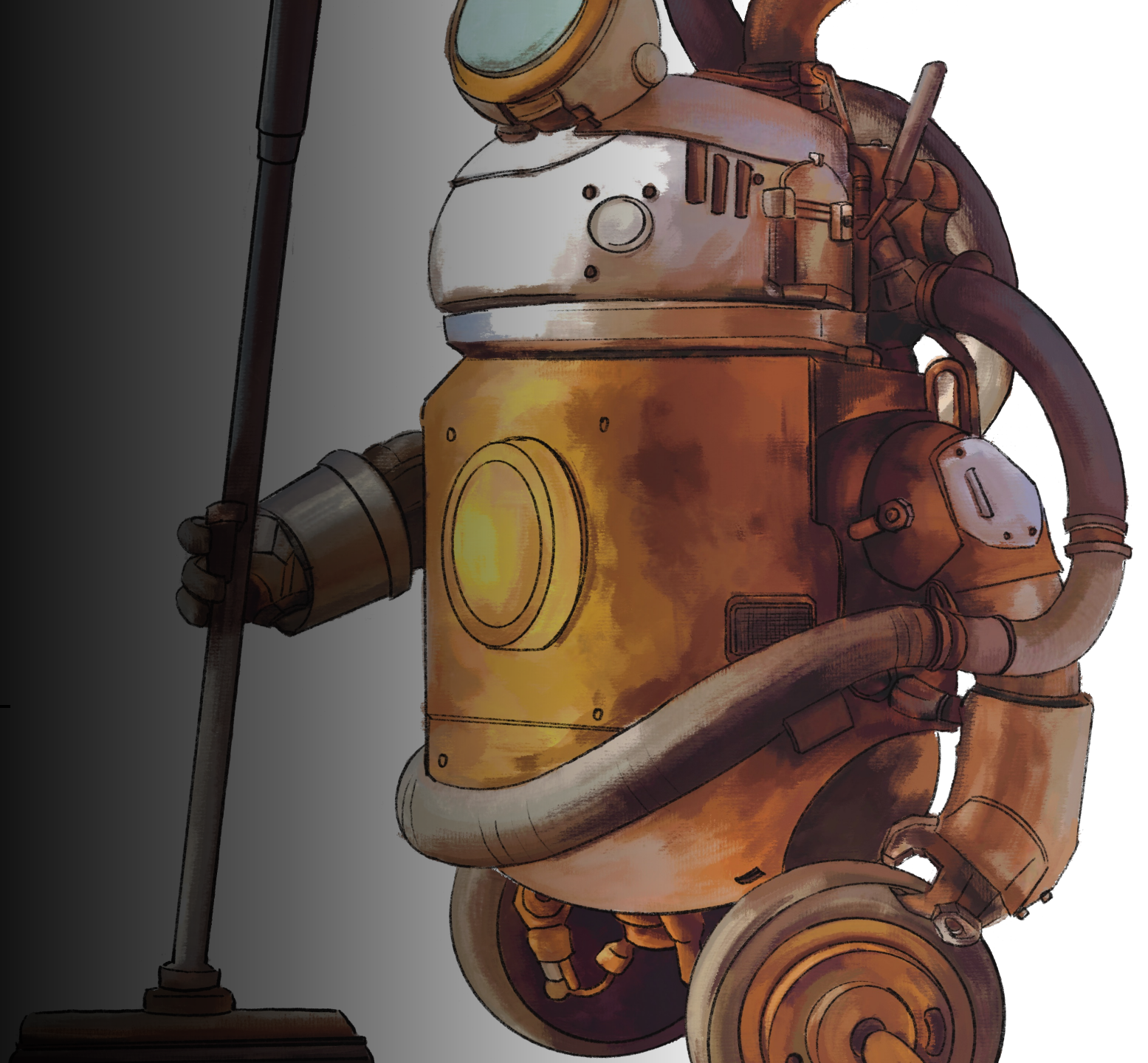
CS 3630, Fall 2025

Lecture 10: *Markov Decision Processes*

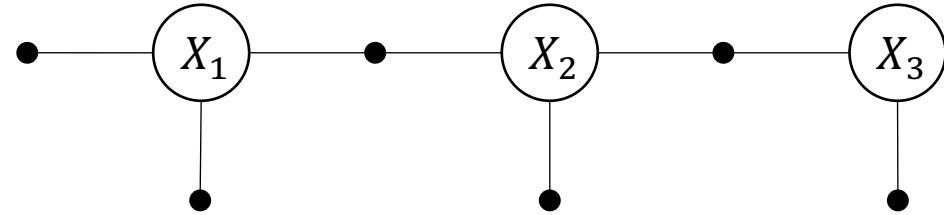
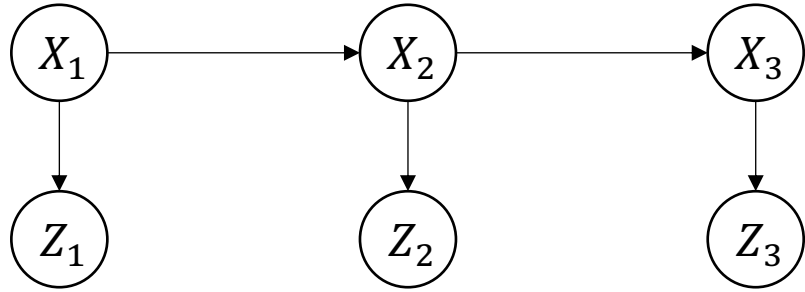


Lecture 9

Recap



Factor Graphs



- Measurements are given – get rid of them!

$$P(\mathcal{X}|\mathcal{Z}) \propto P(X_1)L(X_1; z_1)P(X_2|X_1)L(X_2; z_2)P(X_3|X_2)L(X_3; z_3)$$

- This becomes:

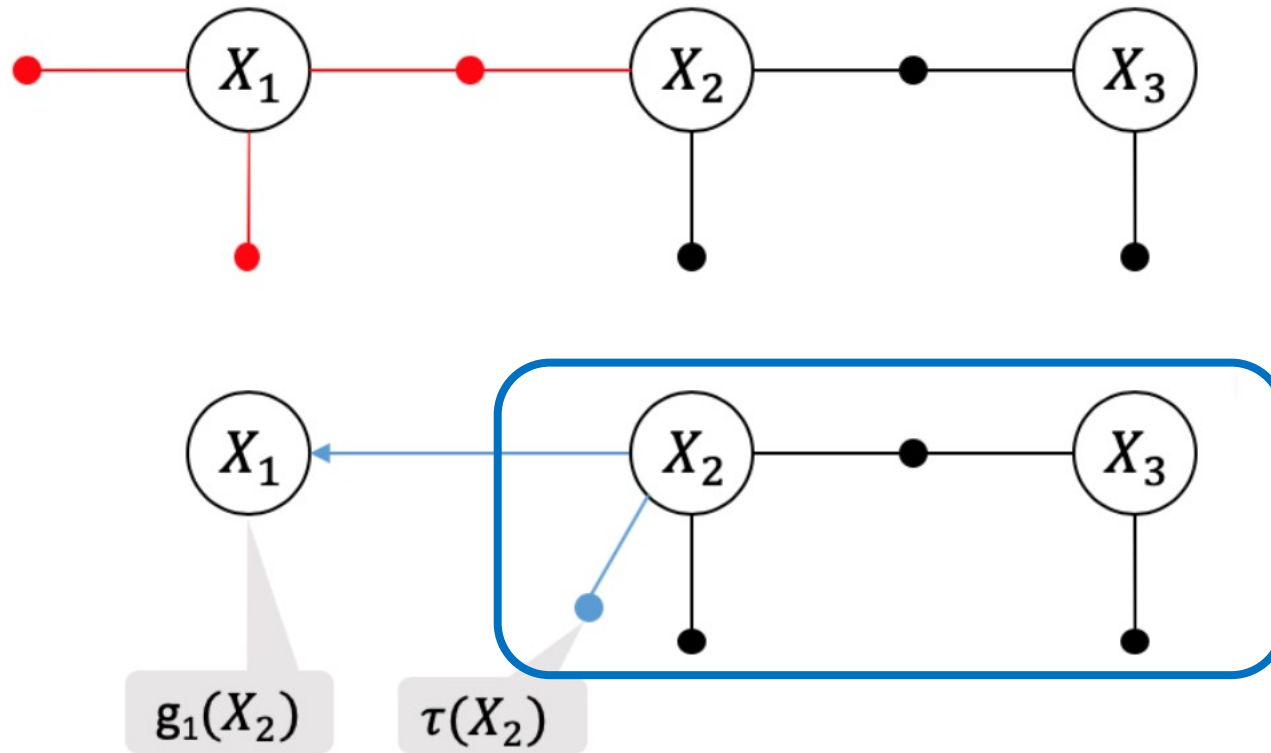
$$\phi(\mathcal{X}) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)\phi_4(X_2)\phi_5(X_2, X_3)\phi_6(X_3)$$

Each factor defines a function ϕ which is a function only of its (non-factor node) neighbors.

MPE via max-product

- Eliminate one variable at a time by forming product, then max:

$$\phi(X_1, X_2) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2).$$

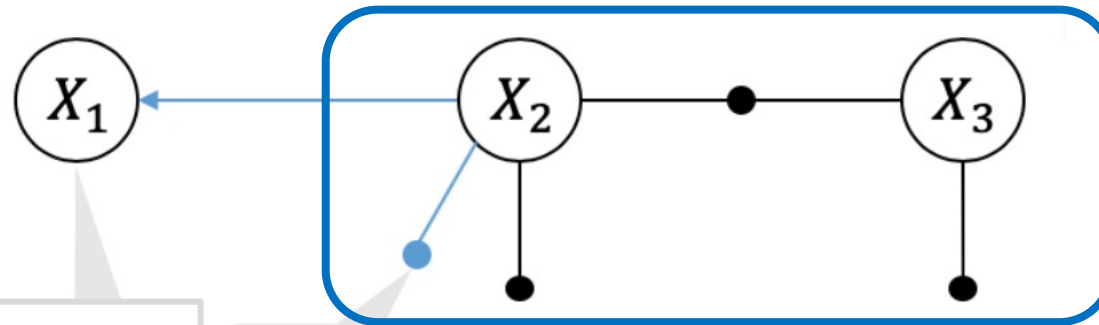
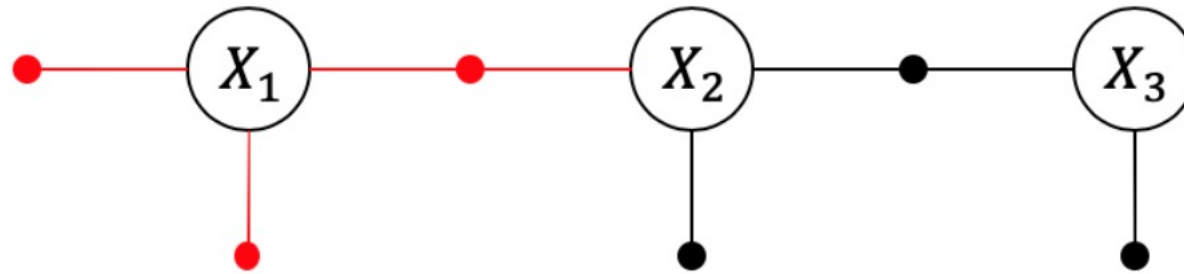


$$g_1(X_2) = \arg \max_{x_1} \phi(x_1, X_2) \quad \tau(X_2) = \max_{x_1} \phi(x_1, X_2)$$

Posterior via sum-product:

- Eliminate one variable at a time by forming product, then sum:

$$\phi(X_1, X_2) = \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2).$$



$$P(X_1|X_2) = \frac{\phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)}{\tau(X_2)}.$$

$$\tau(X_2)$$

$$\tau(X_2) \doteq \sum_{X_1} \phi_1(X_1)\phi_2(X_1)\phi_3(X_1, X_2)$$

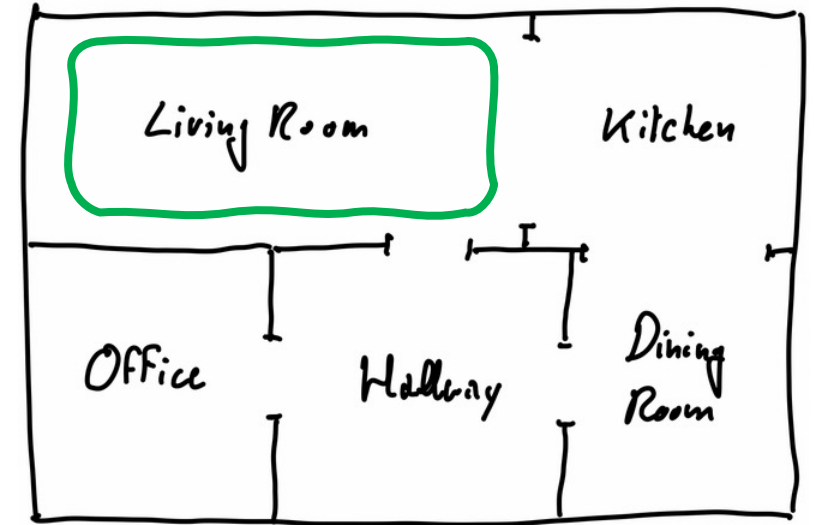
Markov Decision Processes

- Planning is the process of choosing which actions to perform.
- In order to plan effectively, we need quantitative criteria to evaluate actions and their effects.
- MDPs include a reward function that characterizes the immediate benefit of applying an action.
- Policies describe how to act in a given state.
- The value function characterizes the long-term benefits of a policy.
- We assume that the robot is able to **know** its current state with certainty.

➤ *We will see how to define reward functions and use these to compute optimal policies for MDPs.*

Reward Functions

- Most general form depends on current state, action, and next state:
$$R: \mathcal{X} \times \mathcal{A} \times \mathcal{X} \rightarrow \mathbb{R}$$
- In our example, we just care about where we end up after taking an action:



```
def reward_function(state:str, action:str, next_state:str):  
    """Reward that returns 10 upon entering the living room."""  
    return 10.0 if next_state == "Living Room" else 0.0  
  
print(reward_function("Kitchen", "L", "Living Room"))  
print(reward_function("Kitchen", "L", "Kitchen"))
```

```
10.0  
0.0
```

Expected Reward

- A greedy way to act would be to calculate the immediate expected reward for every possible action:

$$\bar{R}(x, a) = E[R(x, a, X')]$$

- Since we know the transition probabilities, we can easily compute this:

$$\bar{R}(x, a) \doteq E[R(x, a, X')] = \sum_{x'} P(x'|x, a) R(x, a, x')$$

- We then have a simple **greedy planning** algorithm:

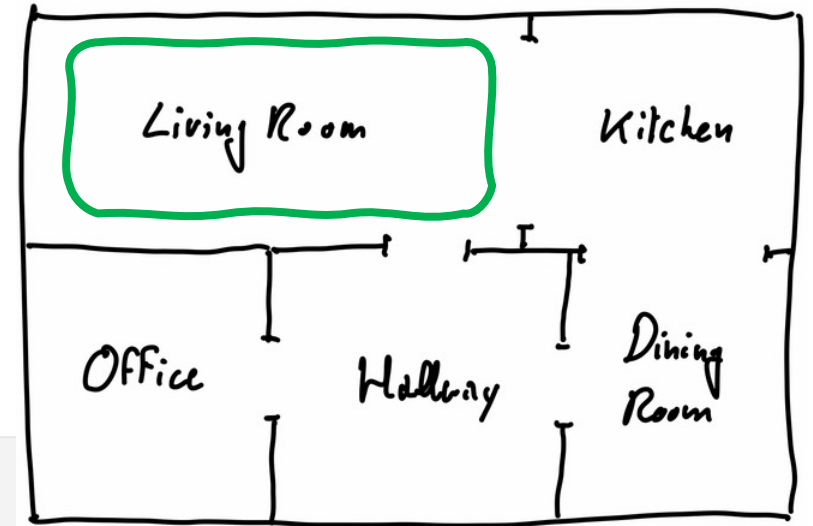
$$a^* = \arg \max_{a \in \mathcal{A}} E[R(X_t, a, X_{t+t})]$$

Example

- The expected immediate reward for all four actions in the Kitchen:

```
x = vacuum.rooms.index("Kitchen")
for a in range(4):
    print(f"Expected reward ({vacuum.rooms[x]}, {vacuum.action_space[a]}) = {T[x,a] @ R[x,a]}")
✓ 0.9s
```

Expected reward (Kitchen, L) = 8.0
Expected reward (Kitchen, R) = 0.0
Expected reward (Kitchen, U) = 0.0
Expected reward (Kitchen, D) = 0.0



- Hence, when in the kitchen, always do L !
- This is a **greedy planning algorithm**

Utility

$$U: \mathcal{A}^n \times \mathcal{X}^{n+1} \rightarrow \mathbb{R}$$

$$U(a_1, \dots, a_n, x_1, \dots, x_{n+1}) = R(x_1, a_1, x_2) + \gamma R(x_2, a_2, x_3) + \dots + \gamma^{n-1} R(x_n, a_n, x_{n+1})$$

- Because actions are uncertain, let's look further into the future!
- Introduce a **discount factor γ** to
 - still bias towards more immediate payoff;
 - allow infinite time horizons:

$$U(a_1, \dots, x_1, \dots) = \sum_{i=1}^{\infty} \gamma^{i-1} R(x_i, a_i, x_{i+1})$$



Expected Utility

$$\mathbb{E}[U(a_1, \dots, a_n, x_1, X_2, \dots, X_{n+1})] = \mathbb{E}[R(x_1, a_1, X_2) + \gamma R(X_2, a_2, X_3) + \dots + \gamma^{n-1} R(X_n, a_n, X_{n+1})]$$

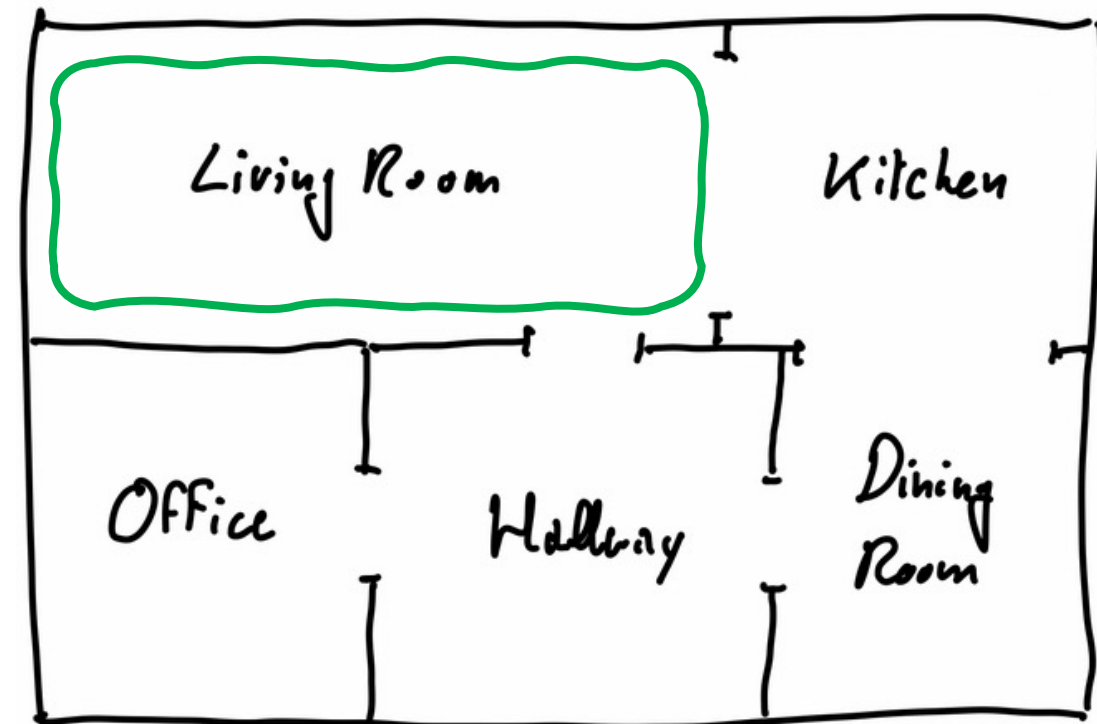
- Again, we can compute the expectation to choose between finite horizon plans
- For $n=3$, we have $4^3 = 64$ possible plans, and for each plan we must evaluate $5^4 = 625$ possible state sequences
- An approximate algorithm to evaluate a given plan:
 - Simulate multiple rollouts
 - Average the result
- Still expensive, only practical for short horizon plans...

Policies $\pi: \mathcal{X} \rightarrow \mathcal{A}$

```
def reward_function(state:str, action:str, next_state:str):  
    """Reward that returns 10 upon entering the living room."""  
    return 10.0 if next_state == "Living Room" else 0.0
```

- Because actions are non-deterministic, fixed plans are brittle and prone to failure.
- Better to have a *state-dependent* plan
- A **policy** $\pi(X)$ is a function that specifies which action to take in each state.
- Let us come up with a policy together:

- $\pi(L) =$
- $\pi(K) =$
- $\pi(O) =$
- $\pi(H) =$
- $\pi(D) =$



The Value Function for a Policy

- Recall the Expected Utility

$$\bar{U}(a_1 \dots a_n, x_1) = E \left[\sum_{i=1}^n \gamma^{i-1} R(X_i, a_i, X_{i+1}) \right]$$

- For a policy, we can define this similarly:

$$\bar{U}(\pi, n, x_1) \doteq E [R(x_1, \pi(x_1), X_2) + \gamma R(X_2, \pi(X_2), X_3) + \dots + \gamma^n R(X_n, \pi(X_n), X_{n+1})]$$

- Can be extended to infinite horizon policy, defining the **value function**:

$$V^\pi(x_1) \doteq E [R(x_1, \pi(x_1), X_2) + \gamma R(X_2, \pi(X_2), X_3) + \gamma^2 R(X_3, \pi(X_3), X_4) + \dots]$$

- Of course, above holds for arbitrary x_t , not just x_1 .

Recursive Definition of V^π

$$V^\pi(x_1) = E[R(x_1, \pi(x_1), X_2) + \gamma R(X_2, \pi(X_2), X_3) + \gamma^2 R(X_3, \pi(X_3), X_4) + \dots]$$

$$V^\pi(x_1) = \sum_{x_2} P(x_2|x_1, \pi(x_1)) \{R(x_1, \pi(x_1), x_2) + \gamma E[R(x_2, \pi(x_2), X_3) + \gamma R(X_3, \pi(X_3), X_4) + \dots]\}$$

$$V^\pi(x_1) = \sum_{x_2} P(x_2|x_1, \pi(x_1)) \{R(x_1, \pi(x_1), x_2) + \gamma V^\pi(x_2)\}$$

$$V^\pi(x_1) = \sum_{x_2} P(x_2|x_1, \pi(x_1)) R(x_1, \pi(x_1), x_2) + \gamma \sum_{x_2} P(x_2|x_1, \pi(x_1)) V^\pi(x_2)$$

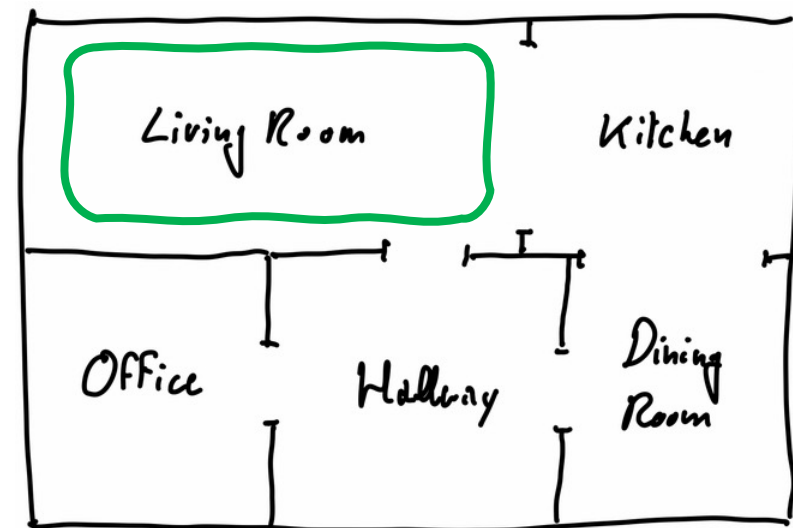
$$V^\pi(x) = \bar{R}(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')$$

Exact Computation for V^π

- Because we have a finite set of states, we get 5 linear equations in 5 unknowns $V^\pi(x)$:

$$V^\pi(x) = \bar{R}(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')$$

- Can be solved efficiently with `np.linalg.solve`
- Example in book:



```
reasonable_policy = [UP, LEFT, RIGHT, UP, LEFT]
```

[[0.1 -0. -0. -0. -0.]	[[10.]
[-0.72 0.82 -0. -0. -0.]	[8.]
[-0. -0. 0.82 -0.72 -0.]	[0.]
[-0.72 -0. -0. 0.82 -0.]	[8.]
[-0. -0. -0. -0.72 0.82]	[0.]



```
V(reasonable_policy):  
Living Room : 100.00  
Kitchen      : 97.56  
Office       : 85.66  
Hallway      : 97.56  
Dining Room  : 85.66
```

Policy Iteration

Start with a random policy π^0 , and repeat until convergence:

1. Compute the value function V^{π^k}
2. Improve the policy for each state x using the update rule:

$$\pi^{k+1}(x) \leftarrow \arg \max_a \left\{ \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^{\pi^k}(x') \right\}$$



```
always_right = [RIGHT, RIGHT, RIGHT, RIGHT, RIGHT]
```

✓ 0.7s

```
optimal_policy, optimal_value_function = policy_iteration(always_right)  
print([vacuum.action_space[a] for a in optimal_policy])
```

✓ 0.7s

```
['L', 'L', 'R', 'U', 'U']
```

Optimal Value Function

The optimal value function is the one corresponding to the optimal policy:

$$\begin{aligned} V^*(x) &= \max_{\pi} V^{\pi}(x) \\ &= \max_{\pi} \left\{ \bar{R}(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^{\pi}(x') \right\} \\ &= \max_a \left\{ \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x') \right\} \end{aligned}$$

The Bellman equation:

$$V^*(x) = \max_a \left\{ \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x') \right\}$$

Value Iteration

Start with a random value function V^0 , and repeat until convergence:

- Improve the value function V^k using the update rule:

$$V^{k+1}(x) \leftarrow \max_a \left\{ \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^k(x') \right\}$$

```
V_k = np.full((5,), 100)
for k in range(10):
    Q_k = np.sum(T * (R + 0.9 * V_k), axis=2) # 5 x 4
    V_k = np.max(Q_k, axis=1) # max over actions
    print(np.round(V_k, 2))
```

✓ 0.4s

```
[100.  98.  90.  98.  90.]
[100.   97.64  86.76  97.64  86.76]
[100.   97.58  85.92  97.58  85.92]
[100.   97.56  85.72  97.56  85.72]
```

Optimal Policy

Given the $V^*(x)$, computing the optimal policy is a straightforward optimization:

$$\pi^*(x) = \underset{a}{\operatorname{arg\,max}} \left\{ \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x') \right\}$$

For convenience, we define the Q^* function as

$$Q^*(x, a) = \bar{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V^{\pi}(x')$$

and we can write the optimal policy as:

$$\pi^*(x) = \underset{a}{\operatorname{arg\,max}} Q^*(x, a)$$

The Q function plays a role in reinforcement learning, to be continued...