Formalizing User/Environment: Markov Decision Processes (MDPs)

A Markov decision process is a tuple $\langle S, A, P, R \rangle$ where

- $S$ is a set of states,
- $A$ is a finite set of actions,
- $P$ is a state transition probability matrix s.t. $P_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$,
- $R$ is a reward function s.t. $R_s = E[R_{t+1} | S_t = s, A_t = a]$. 

Markov Decision Processes
Dynamics of MDPs

One-step dynamics of the environment under the Markov property is completely specified by probability distribution over pairs of next state and reward $s', r$, given state and action $s, a$:

$$p(s', r|s, a) = P[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a].$$

Exercise: Specify $P^a_{ss'}$ and $R^a_s$ in terms of $p(s', r|s, a)$. 
Formalizing Agent/System: Policies

A **stochastic policy** is a distribution over actions given states s.t.

- $\pi(a|s) = P[A_t = a|S_t = s]$.
- A policy completely specifies the behavior of an agent/system.
- Policies are parameterized $\pi_\theta$, e.g. by a linear model or a neural network - we use $\pi$ to denote $\pi_\theta$ if unambiguous.
- Deterministic policies $a = \pi(s)$ also possible.
Policy Evaluation and Policy Optimization

Two central tasks in RL:

- **Policy evaluation (a.k.a. prediction):** Evaluate the expected reward for a given policy.

- **Policy optimization (a.k.a. learning/control):** Find the optimal policy / optimize a parametric policy under the expected reward criterion.
Return and Value Functions

- The **total discounted return** from time-step $t$ for discount $\gamma \in [0, 1]$ is
  \[ G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \]

- The **action-value function** $q_\pi(s, a)$ on an MDP is the expected return starting from state $s$, taking action $a$, and following policy $\pi$ s.t.
  \[ q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \]

- The **state-value function** $v_\pi(s)$ on an MDP is the expected return starting from state $s$ and following policy $\pi$ s.t.
  \[ v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_{a \sim \pi}[q_\pi(s, a)]. \]
Bellman Expectation Equation

The state-value function can be decomposed into immediate reward plus discounted value of successor state s.t.

\[ v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \]

\[ = \sum_{a \in A} \pi(a|s) \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right). \]

In matrix notation:

\[ v_\pi = R^\pi + \gamma P^\pi v_\pi. \]
Policy Evaluation by Linear Programming

The value of $v_\pi$ can be found directly by solving the linear equations of the Bellman Expectation Equation:

- **Solving linear equations:**

  $$v_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

- Only applicable to small MDPs.
Policy Evaluation by Dynamic Programming (DP)

Value of $v_\pi$ can also be found by iterative application of Bellman Expectation Equation:

- **Iterative policy evaluation:**
  
  $$v_{k+1} = R^\pi + \gamma P^\pi v_k.$$

- Performs **dynamic programming** by recursive decomposition of Bellman equation.
- Can be parallelized (or backed up asynchronously), thus applicable to large MDPs.
- Converges to $v_\pi$. 
Dynamic Programming

Policy Optimization using Bellman Optimality Equation

An optimal policy $\pi^*$ can be found by maximizing over the optimal action-value function $q^*(s, a) = \max_\pi q_\pi(s, a)$ s.t.

$$
\pi^*(s) = \arg\max_a q^*(s, a).
$$

The optimal value functions are recursively related by the Bellman Optimality Equation:

$$
q^*(s, a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a')|S_t = s, A_t = a] \\
= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^{a'} \max_{a'} q^*(s', a').
$$
Dynamic Programming

Policy Optimization by Value Iteration

The Bellman Optimality Equation is non-linear and requires iterative solutions such as value iteration by dynamic programming:

- **Value iteration for \( q \)-function:**

\[
q_{k+1}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^{a'} \max_{a'} q_k(s', a').
\]

- Converges to \( q^*(s, a) \).

**Exercise:** Write the \( q \)-value iterations in terms of matrix operations.
Summary: Dynamic Programming

- Earliest RL algorithms with well-defined convergence properties.
- Bellman equation gives recursive decomposition for iterative solution to various problems in policy evaluation and policy optimization.
- Can be trivially parallelized or even run asynchronously.
- We need to know a full MDP model with all transitions and rewards, and touch all of them in learning!