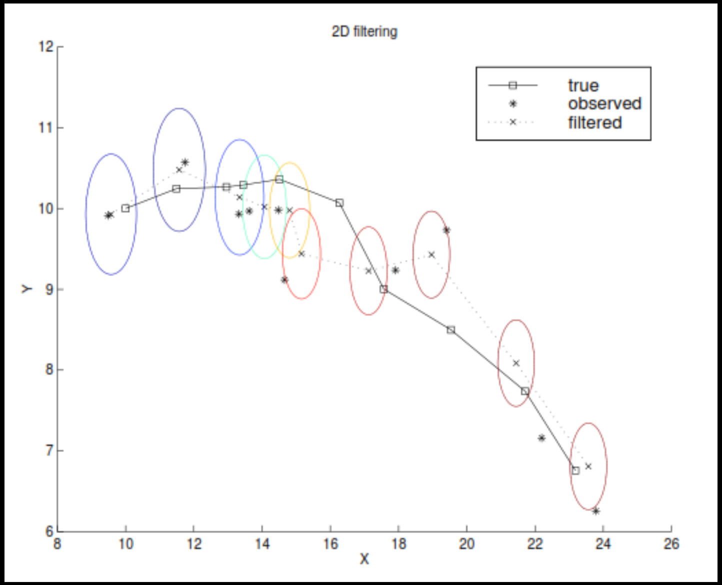
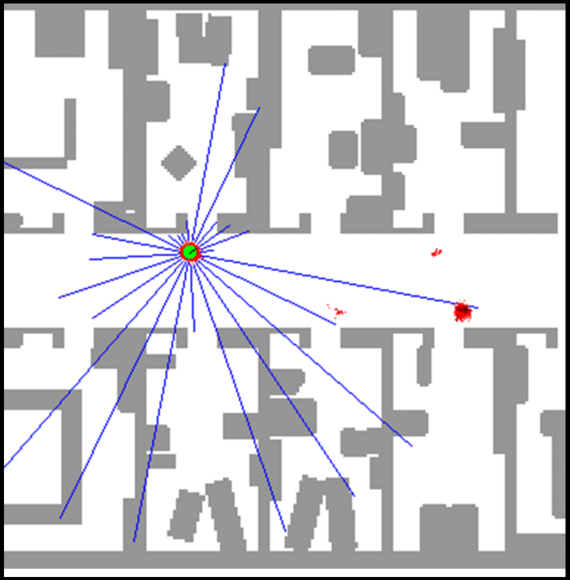
MDPs and POMDPs

Last time…

- We saw two types of filters for estimating continuous random variables





- But so far, we haven't used uncertain information to make decisions

Outline

- Markov Decision Process (MDPs)
  - Definition
  - Value-iteration algorithm
  - Policy-iteration algorithm
- Partial-observable Markov Decision Processes (POMDPs)
  - Definition
  - Overview of algorithms

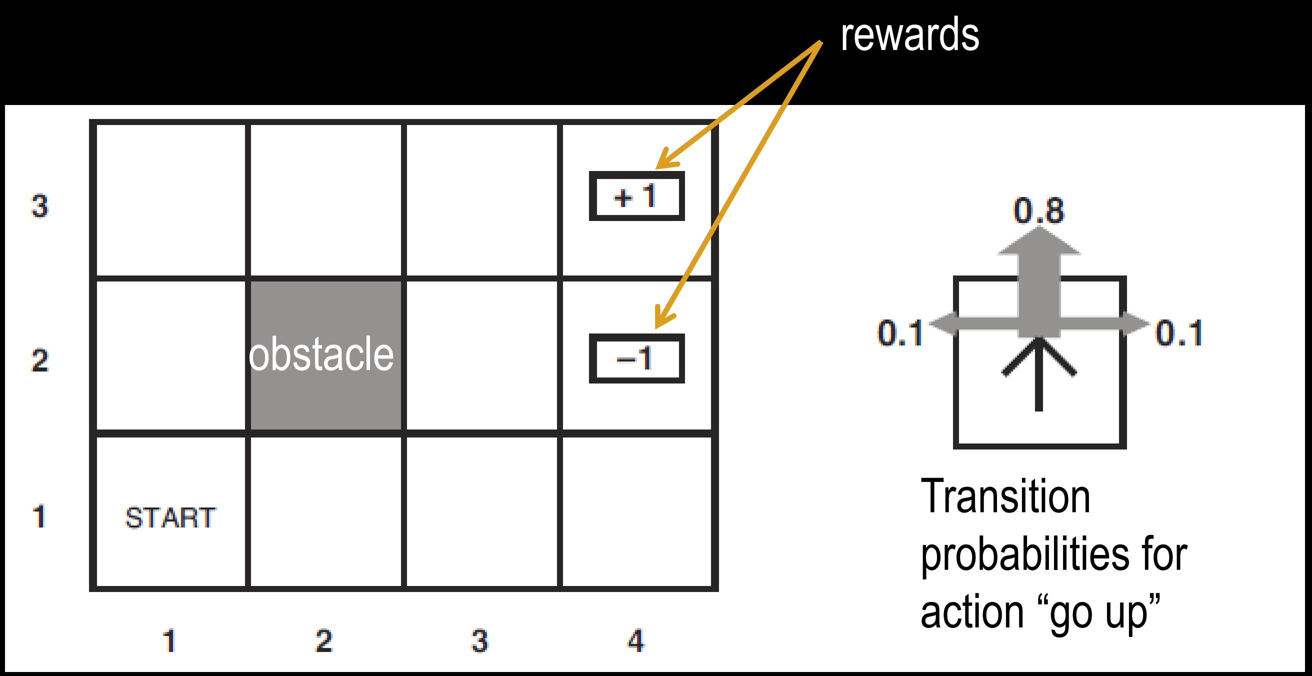
The Markov Zoo

- Markov process + partial observability = HMM
- Markov process + actions = MDP
- Markov process + partial observability + actions = HMM + actions = MDP + partial observability = **POMDP**

	full observability	*partial observability*
no actions	**Markov process** ✓	**HMM** ✓
actions	**MDP**	**POMDP**

Markov Decision Processes (MDPs)

- Used to represent a series of decisions that need to be made
- State **is known** at each time step
- State transitions can be uncertain
- Example: grid world



Change of notation

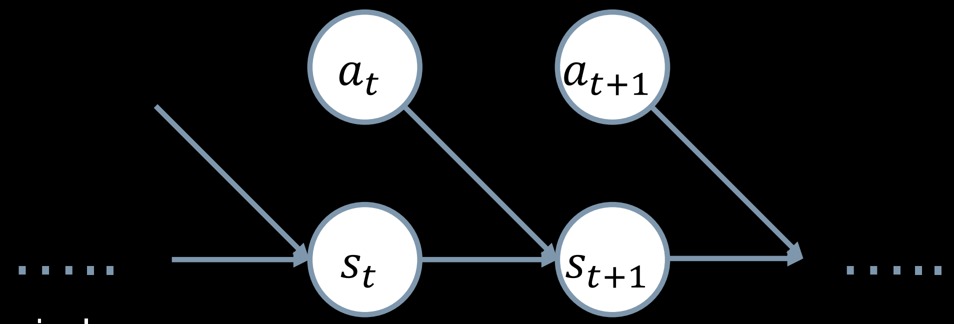
- So far we've been using
  - x is the state
  - u is the action
  - z is the sensor data
- To match the book, we now switch to:
  - s is the state
  - a is the action
  - e is the sensor data

- Why doesn't everyone use the same notation?
  - A long time ago, there was a schism between the AI and Control communities in the 1960s

MDPs

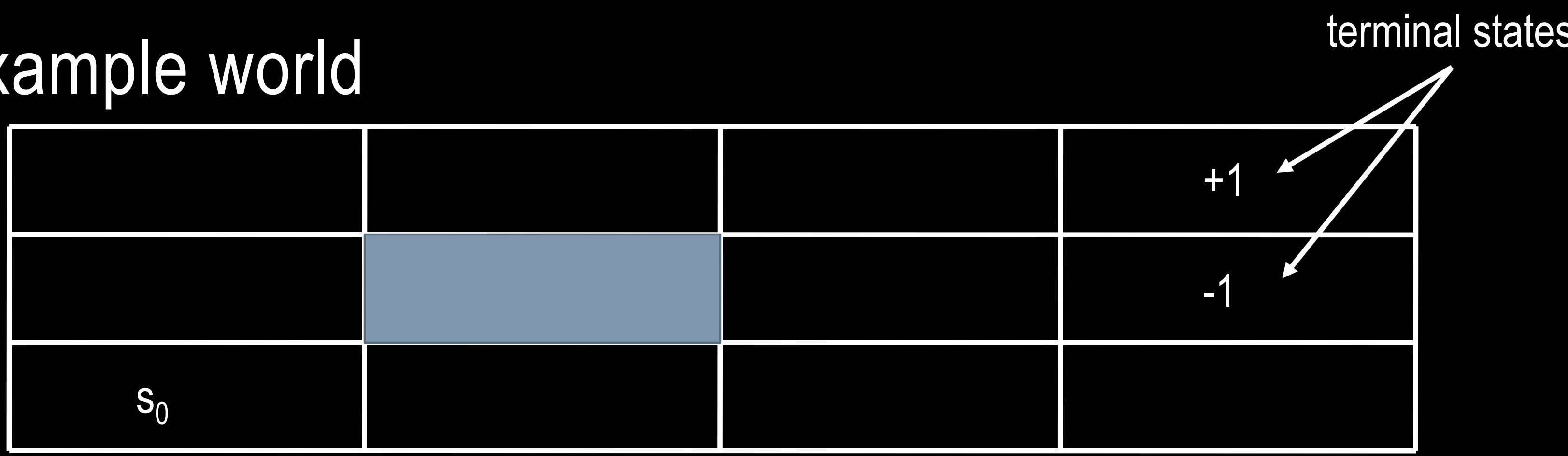
- Inputs
  - Initial State: s_0
  - Transition Model: P(s' | s, a)
  - Reward function: R(s)

Outputs a single real number



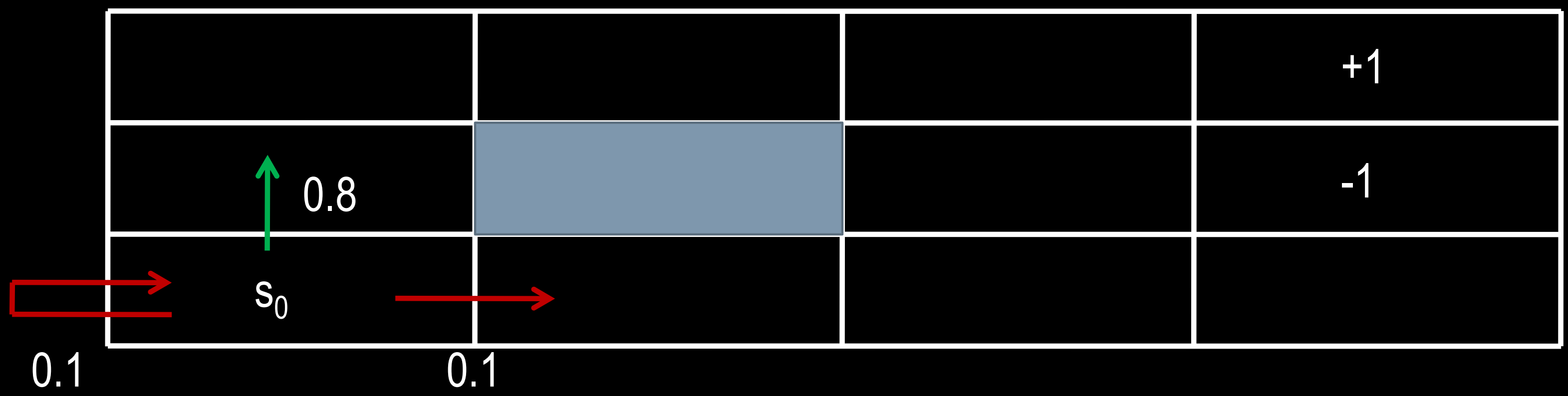
- Outputs
  - Policy, π(s) = a
  - The policy outputs an action to take for ANY state

Example world



- Transition model P(s' | s, a):
  - 80% of desired direction
  - 10% 90 degrees to the left
  - 10% 90 degrees to the right
  - Hitting a wall keeps you in place
- Reward R(s):
  - -0.04 per time step, +1 or -1 for terminal states

For example



- In s_0, try to go up
  - Will probably arrive one cell up
  - Might arrive one cell to the right
  - Reward of -0.04 no matter where you end up

- Assumes **you know where you are** (unlike POMDPs, discussed later)

Handling rewards

- Can just sum them up along the path
  - Sequence of rewards: 1, 4, -3
  - Total reward: 1+4 + (-3) = 2

- Problems
  - Infinite length trials
  - The future is uncertain (and changing)

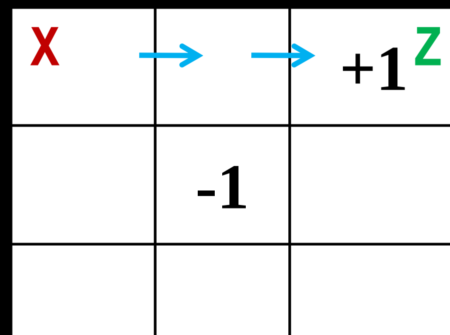
Discounting future rewards

- Graceful solution to both infinite-length and uncertain future:

$$\text{Total reward} = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

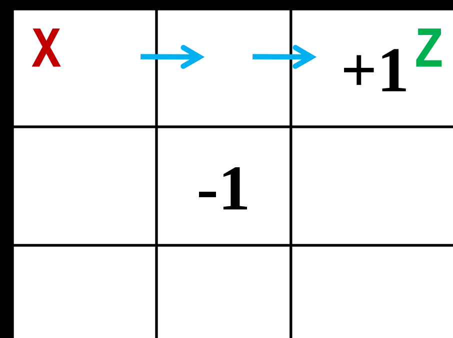
- $0<\gamma<=1$ is a parameter you specify
  - $\gamma = 1$ means just sum the rewards
  - $\gamma \approx 0$ is a very myopic agent
- $\gamma \approx 0.9$ is often used

Example



- γ= 0.9, R(s) = -0.1
- States in **bold** are terminal states (with rewards)
- What is expected future reward for starting at X and ending at Z using movements in blue?

Example



- γ= 0.9, R(s) = -0.1
- Call total reward the *utility* U

- $U(s_0, s_1, s_2) = -0.1 + \gamma*-0.1 + \gamma^2*1$
- $U(s_0, s_1, s_2) = -0.1 + -0.09 + 0.81$
- $U(s_0, s_1, s_2) = 0.62$

The value function

.812	.868	.918	+1
.762		.660	-1
s_0 .705	.655	.611	.388

The Value function is the expected future reward when starting from a state (value is a synonym for utility here)

- Transition model P(s' | s, a):
  - 80% of desired direction
  - 10% 90 degrees to the left
  - 10% 90 degrees to the right
  - Hitting a wall keeps you in place
- Given the state where we currently are, in which direction should we move?

What action should we take from **this state**?

.812	.868	.918	+1
.762		.660	-1
s_0 .705	.655	**.611**	.388

Computing optimal policy

.812	.868	.918	+1
.762		.660	-1
s_0 .705	.655	**.611**	.388

- Expected utility of going **up** = 0.8 * .66 + 0.1 * .655 + 0.1 * .388
  = 0.6323
- Expected utility of going **left** = 0.8 * .655 + 0.1 * .66 + 0.1 * .611
  = 0.6511
- (ignoring constant -0.04 per time step)

Optimal policy

- A policy $\pi(s) = a$ says what action to take at every state
- Optimal policy

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s' \mid s,a)\, U(s')$$

- Utility values are

$$U(s,a) = \sum P(s' \mid s,a) U(s')$$

Do the utility values depend on the policy the agent follows?

.812	.868	.918	+1
.762		.660	-1
s_0 .705	.655	.611	.388

- Yes

Intuition: if policy from **this state** is to always move right, is utility 0.66?

.812	.868	.918	+1
.762		**.660**	-1
S_0 .705	.655	.611	.388

Utility of a state given a policy

- $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0=s]$
- Note: s_t depends on π (policy being followed)
    - Is not necessarily the best possible outcome

Computing the optimal utilities

- Approximate optimal utilities using Value-Iteration algorithm
  - Initialize all state utilities to 0 (except terminal rewards)
  - Iteratively update all states' utilities using the Bellman update:

$$U_{i+1}(s) = R(s) + \gamma * \max_{a} \sum_{s'} P(s' \mid s,a)\, U_i(s')$$

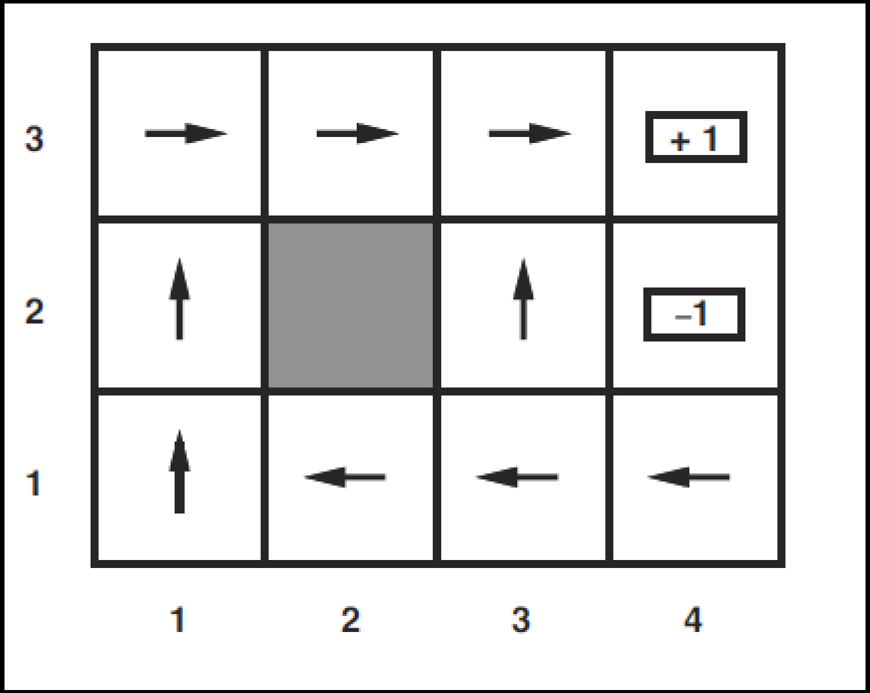
  - Apply update to all states simultaneously at each iteration
  - Check for convergence (in the book)

- After value iteration, optimal policy is:

$$\pi^*(s) = \operatorname{argmax}_{a} \sum_{s'} P(s' \mid s,a)\, U(s')$$

Example optimal policy

- For R(s) = -0.04 in non-terminal states:



Issues with value iteration

- Why do we need to know the precise utilities of each state?
  - If one action is clearly better than others, then exact magnitude of utilities is irrelevant
- So, computing optimal policy is easier than computing precise utilities
- If optimal policy isn't changing during computation, we should stop
- Leads to new algorithm: Policy-Iteration

Policy iteration

- Very similar to value iteration

- Key difference:
  - Value iteration computes the *max* of all possible moves from a state
  - Policy iteration only computes result of following *current* optimal policy

Value Iteration vs. Policy Iteration

- Value iteration update:

$$U_{i+1}(s) = R(s) + \gamma * \max_a \sum_{s'} P(s' \mid s,a)\, U_i(s')$$

- Policy iteration update:

$$U_{i+1}(s) = R(s) + \gamma * \sum_{s'} P(s' \mid s, \pi_i(s))\, U_i(s')$$

Policy iteration algorithm

1. Policy evaluation
   - Compute utility of each state using current policy π:

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0=s]$$

   - Not just one step look-ahead, true utility of entire sequence
2. Policy improvement
   - Revise policy by looking at new utilities of actions for each state and picking best action
3. Repeat until policy stops changing

Value/Policy Iteration Issues

- Value/Policy iteration does not scale well with number of states
  - Need to compute value for **every state**
- But the goal is **not** to estimate value of states
  - Goal is to find a *policy*
- Can we better allocate computation?
  - Should we spend time where it is likely to influence policy?
  - Should we spend more time on (and near) well-trod paths?
- Reinforcement learning is a better way to do this in many cases
  - We won't cover this in our class

Break

POMDPs

The Markov Zoo

- Markov process + partial observability = HMM
- Markov process + actions = MDP
- Markov process + partial observability + actions = HMM + actions = MDP + partial observability = **POMDP**

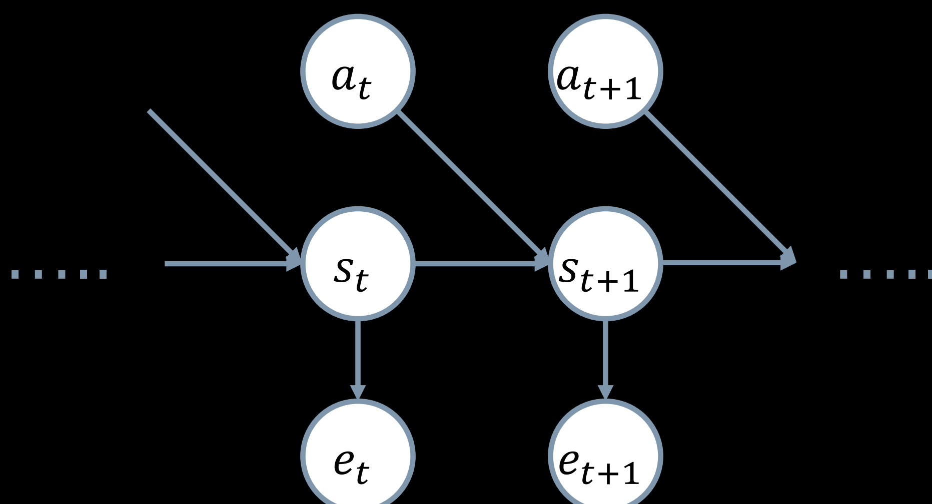
	full observability	*partial observability*
no actions	**Markov process** ✓	**HMM** ✓
actions	**MDP** ✓	**POMDP**

POMDP model

- Finite set of states: $s_1,\dots,s_n = S$
- Finite set of actions: $a_1,\dots, a_m = A$
- Probabilistic state/action transitions: $P(s' \mid a,s)$
- Immediate reward (cost) for each state: $R(S)$

MDP

- Conditional observation probabilities: $P(e \mid s)$
  - (more generally, $P(e \mid s, a, s')$, but we'll ignore that)



Belief state

- Belief b(s) is a probability distribution over all states
  - Vector length |S| of values [0, 1]
  - b(s) = probability the system is at state s
  - Belief is *continuous* (can range from 0 to 1 for each element of the vector)

- If b(s) was previous belief state, then agent does action a and perceives evidence e, new belief state is b'(s):

$$b'(s) = \alpha\, P(e \mid s') \sum_{s} P(s' \mid a, s)\, b(s)$$

Normalizing constant (makes belief state sum to 1)

Choosing the optimal action

$$b'(s) = \alpha\, P(e \mid s') \sum_{s} P(s' \mid a, s)\, b(s)$$

- This should look familiar: same as filtering for Bayes nets with time!
  - b’ = FORWARD(b, a, e)

- Key to POMDPs: The optimal action depends only on the agent’s current *belief state* (true state is unknown)
  - Policy, $\pi(b) = a$

Decision Cycle of a POMDP

1. Given current belief state b, execute action a = $\pi^*(b)$
2. Receive perception e
3. Set new belief state b' = FORWARD(b,a,e)

Computing Optimal Policies for POMDPs

- The trick: convert the POMDP to an MDP whose state is the belief

- Each MDP state is a ***probability distribution*** (continuous belief state *b*) over the states of the original POMDP

The MDP over belief state of the POMDP

- State transitions are the products of actions and evidence:

$$P(b' \mid a, b) = \sum_{e} P(b' \mid e, a, b)\, P(e \mid a, b)$$

$$P(b' \mid a, b) = \sum_{e} P(b' \mid e, a, b) \sum_{s'} P(e \mid a, s', b) P(s' \mid a, b)$$

$$P(b' \mid a, b) = \sum_{e} P(b' \mid e, a, b) \sum_{s'} P(e \mid s') P(s' \mid a, b)$$

$$P(b' \mid a, b) = \sum_{e} P(b' \mid e, a, b) \sum_{s'} P(e \mid s') \sum_{s} P(s' \mid s, a) b(s)$$

$P(b' \mid e, a, b)$ is 1 if b’ = FORWARD(b,a,e) and 0 otherwise

The MDP over belief state of the POMDP

- Reward function for POMDP is:

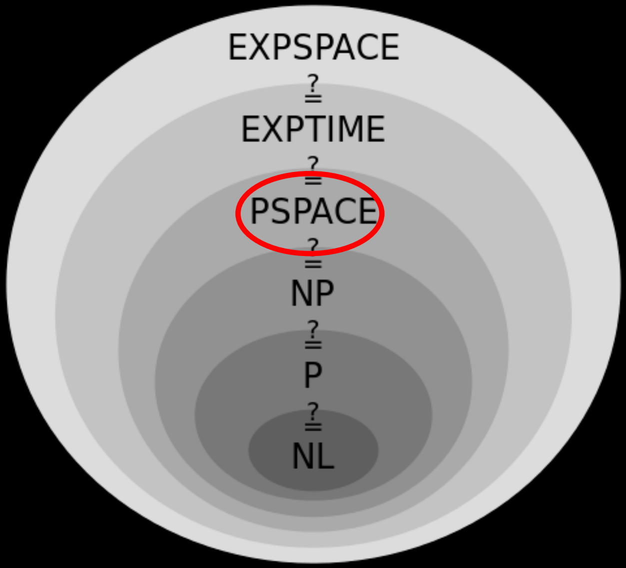
$$\rho(b) = \sum_{s} b(s) R(s)$$

- Why the trick works: belief state is always observable by the agent, so you can make it the state of an MDP

- Can we apply the Value/Policy iteration we saw to this MDP to get the optimal policy?
    - No. Those dealt with discrete MDPs, this one is continuous.

Optimal Policies for POMDPs

- Possible to adapt value iteration to POMDPs (in book)
  - But algorithm is hopelessly inefficient
- For general POMDPs, finding optimal policies is PSPACE-hard!
- Must rely on approximation methods



Common POMDP methods

- Value Iteration (too inefficient)
  - Dynamic programming approach using Bellman equation
- Policy Iteration (too inefficient)
  - Represent policy as a state machine and incrementally modify
- Point-Based Approaches
  - Solve for a **single** initial belief state, rather than all states
  - Iterate until expected value of the given initial belief state converges to within some threshold
  - Amenable to heuristic search
- Greedy Approaches
  - Use solution to underlying MDP
  - Basically assumes that world becomes observable after 1 step

POMDP Software

- Approximate POMDP Planning (APPL) Toolkit (David Hsu et al.)
  - Implementation of the SARSOP algorithm for solving POMDPs
    - SARSOP is a point-based POMDP solver
  - Implementation of Monte Carlo Value Iteration
  - http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

- Applied to real problems!
  - POMDP for aircraft collision avoidance

Drawbacks of POMDPs/MDPs

- Scale poorly with number of states (discrete) or dimension (continuous)
  - Hsu et al. made some progress on this issue
- How do you get all the transition/observation probabilities?
  - Major learning/modeling problem
- Where does the reward come from?
  - Can use Inverse Reinforcement Learning from demonstrations of optimal behavior, but this is often ill-posed

Summary

- MDPs represent sequences of decisions with uncertain outcomes
  - Assume the state is fully observable
- Value iteration for MDPs produces the Utility function (AKA the Value function)
  - Optimal policy is extracted from Utility function
- Policy iteration for MDPs does same thing as Value iteration but is more efficient
  - Optimal policy is output directly
  - Doesn't need to compute precise utility of every state
- POMDPs use evidence to estimate the belief b(s) of being in state s
  - Can convert a POMDP into an MDP by making b the state of the MDP
  - Value/Policy iteration are hopeless for POMDPs
  - Active research on how to efficiently approximate optimal policy in POMDPs

Homework

- Read AI book Ch. 18.1-18.3
- Homework 5 due next week!
- No class Wednesday