AMRITA
VISHWA VIDYAPEETHAM

21AIE311 - REINFORCEMENT LEARNING

MARKOV DECISION PROCESS

DR. AMUDHA J. AND MR. A. A. NIPPUN KUMAAR
DEPARTMENT OF CSE
AMRITA SCHOOL OF COMPUTING, BANGALORE

PREVIOUSLY

- Recap and Terminologies
- Markov Decision Process
- The Agent-Environment Interface
- Goal and Rewards
- Returns and Episodes
- Unified Notion

LECTURE OVERVIEW

- Recap - Markov Decision Process
- MDP - Examples
- Policies and Value Functions
- Optimal Policies and Optimal Value Functions

RECAP

- Markov Property: The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- MDP: If a reinforcement learning task has the Markov Property, it is basically a Markov Decision Process (MDP).
- MDP framework abstraction - goal-directed learning
- Three signals pass back and forth between the agent and the environment
  - The choice made by the agent - Actions
  - The basis on which the choice is made - States
  - Agent's goal - Reward

RECAP…

- Goals and Rewards: Rewards should be designed to achieve the goal.
- The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.
- Returns: Expected reward over time steps.
- Episodic Tasks: Interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.
- Continual Tasks: Interaction does not have natural episodes. Here terminal state time T is infinite.
- Discounter Return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

AN EXAMPLE FINITE MDP – RECYCLING ROBOT

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Reward = number of cans collected

$$S = \{\text{high}, \text{low}\}$$
$$A(\text{high}) = \{\text{search}, \text{wait}\}$$
$$A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$$

$$R^{\text{search}} = \textit{expected no. of cans while searching}$$
$$R^{\text{wait}} = \textit{expected no. of cans while waiting}$$
$$R^{\text{search}} > R^{\text{wait}}$$

RECYCLING ROBOT MDP

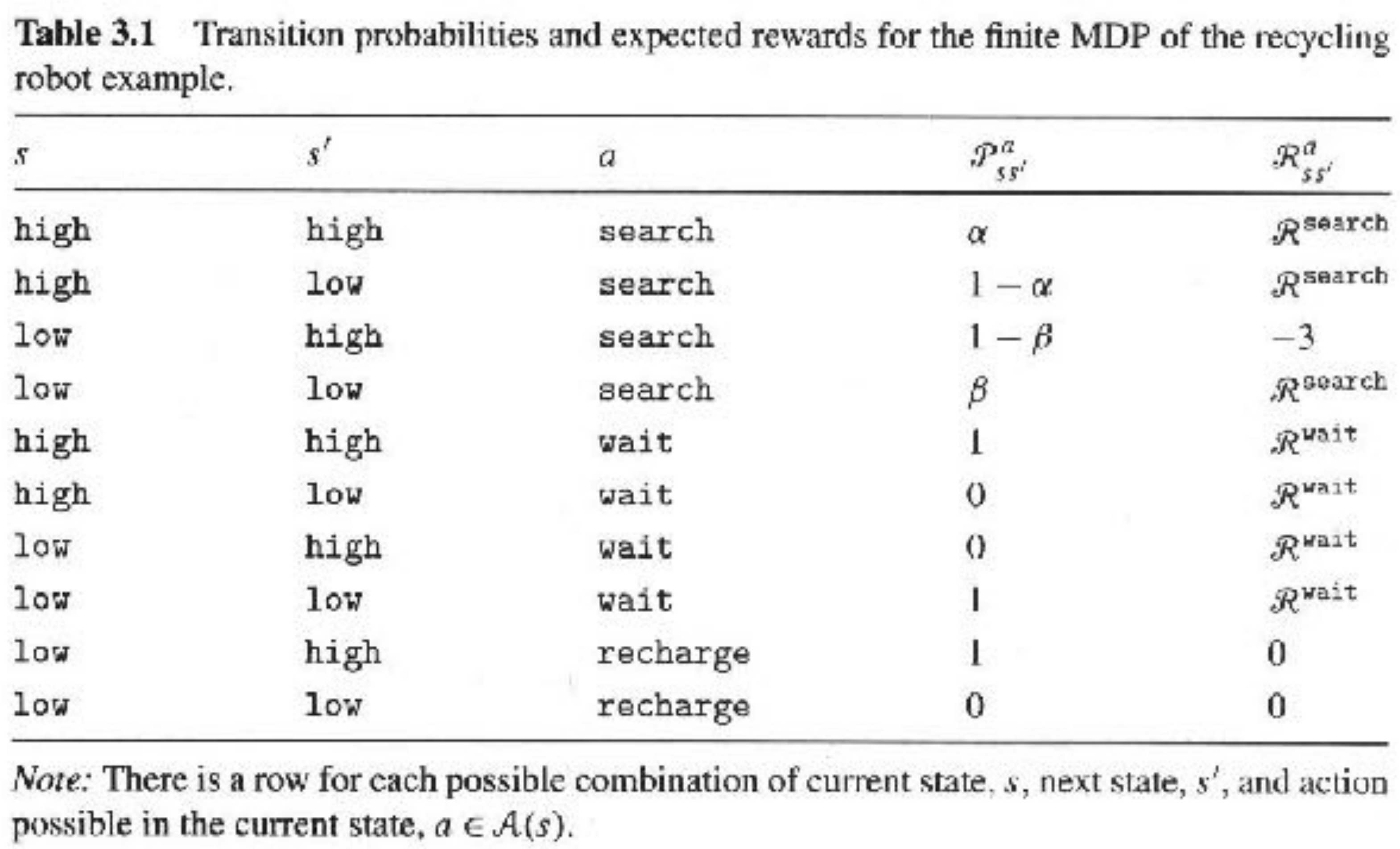
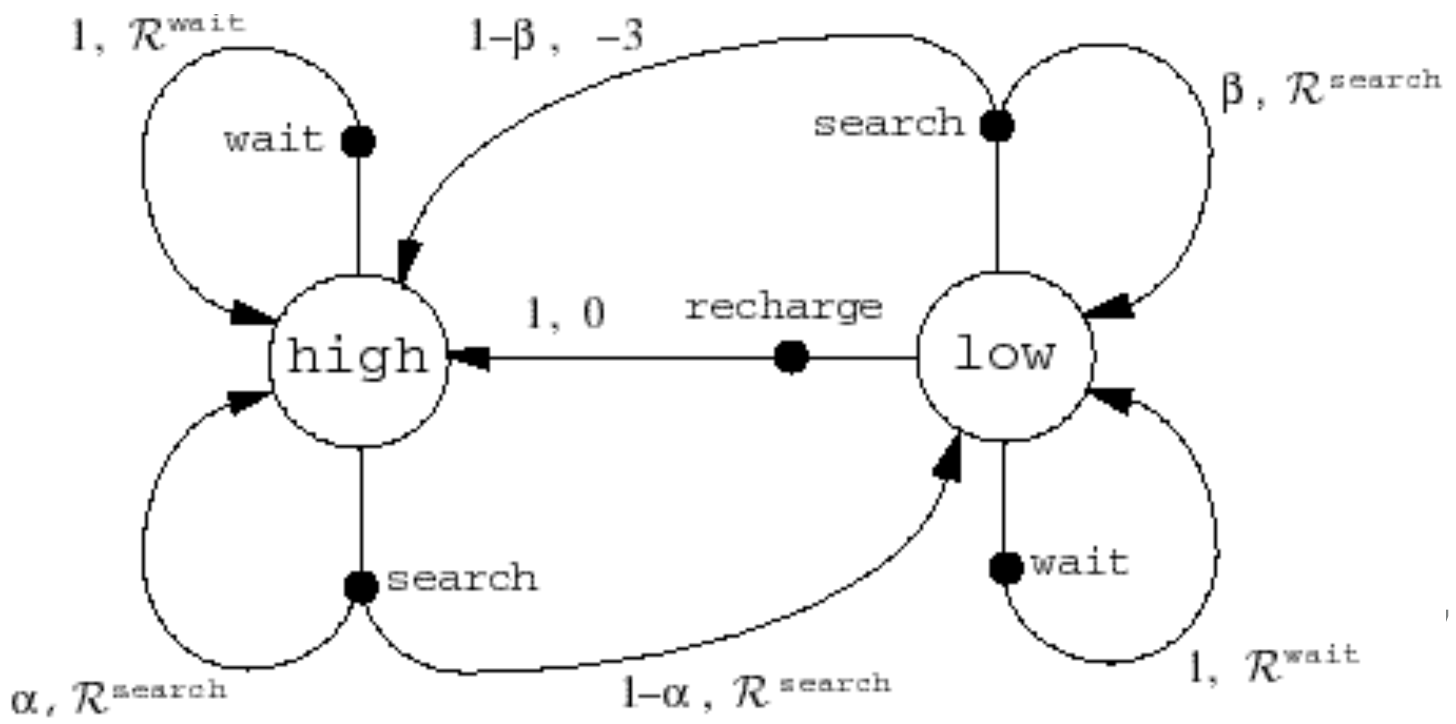


Table 3.1 Transition probabilities and expected rewards for the finite MDP of the recycling robot example.

s	s'	a	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\texttt{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\texttt{search}}$
high	high	wait	1	$\mathcal{R}^{\texttt{wait}}$
high	low	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	high	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	low	wait	1	$\mathcal{R}^{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0

Note: There is a row for each possible combination of current state, s, next state, s', and action possible in the current state, $a \in \mathcal{A}(s)$.


1, $\mathcal{R}^{\texttt{wait}}$
wait
$1-\beta$, -3
search
β, $\mathcal{R}^{\texttt{search}}$
high
1, 0
recharge
low
search
wait
α, $\mathcal{R}^{\texttt{search}}$
$1-\alpha$, $\mathcal{R}^{\texttt{search}}$
1, $\mathcal{R}^{\texttt{wait}}$

GRID WORLD ENVIRONMENT

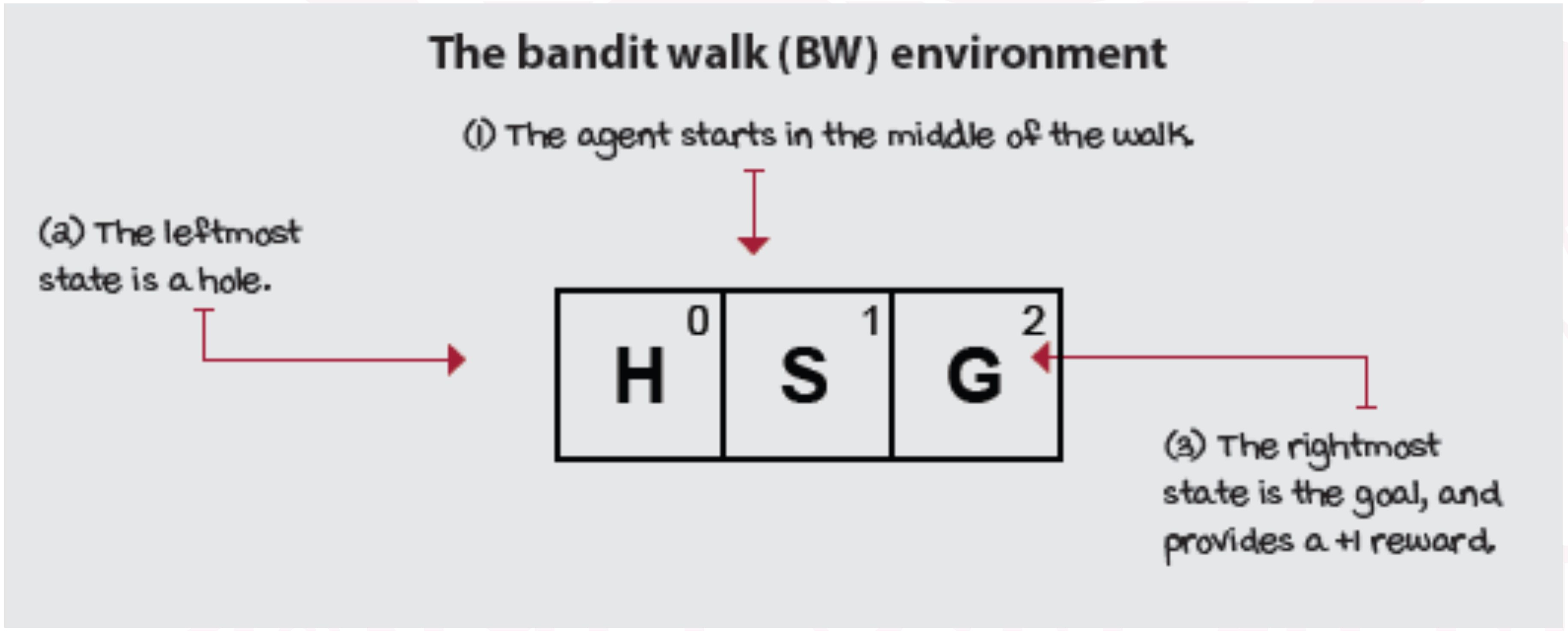
- It is a simple Grid World (GW) Environment.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

- A finite set of States (Each cell in the GW) is represented by the cell numbers.
- All states tend to have a fully observable discrete state and observation spaces (that is, state equals observation).
- A finite set of Actions (Left, Down, Right, Up) that directly leads to a state change.

BANDIT WALK (BW)

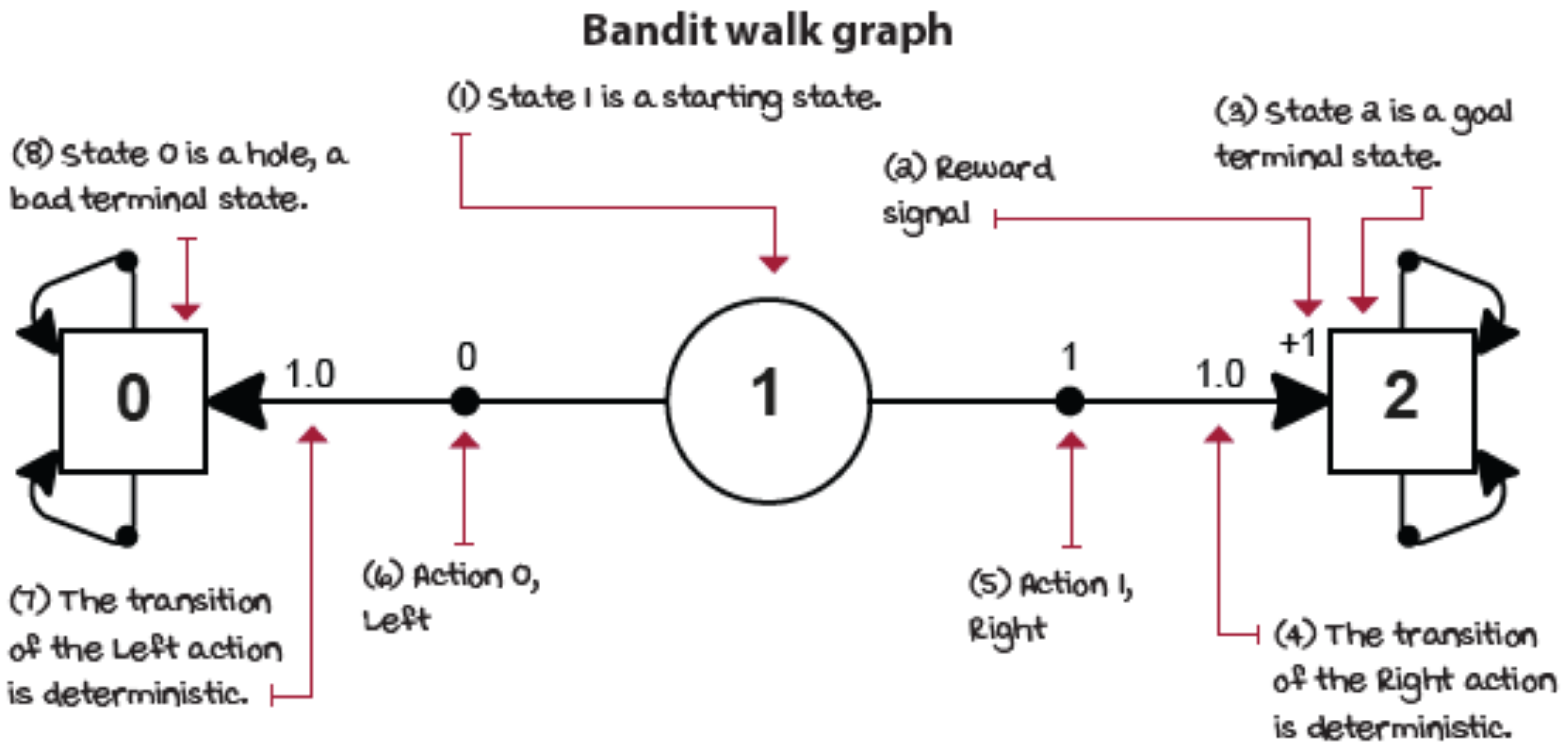
- It uses a GW environment.
- A "walk" is a special case of grid-world environments with a single row.
- The bandit walk (BW) has three states, but only one non-terminal state.


The bandit walk (BW) environment
(1) The agent starts in the middle of the walk.
(a) The leftmost state is a hole.
H 0
S 1
G 2
(3) The rightmost state is the goal, and provides a +1 reward.

BANDIT WALK (BW)

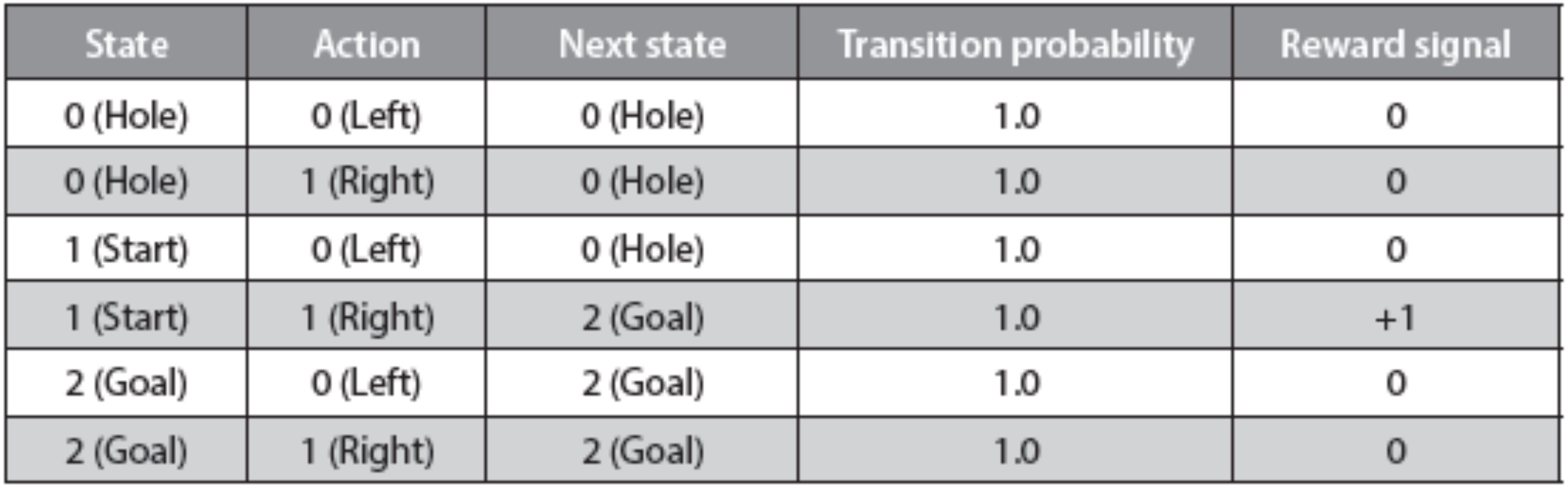
- States: Hole(0), Start(1), and Goal(2)
- Action: Left (0) and Right (1)
- Reward: +1 in State-2(Goal)
- A deterministic transition function

Transition table ??



Bandit walk graph
(1) State 1 is a starting state.
(8) State 0 is a hole, a bad terminal state.
(2) Reward signal
(3) State 2 is a goal terminal state.
0
1.0
0
1
1
1.0
+1
2
(7) The transition of the Left action is deterministic.
(6) Action 0, Left
(5) Action 1, Right
(4) The transition of the Right action is deterministic.

BANDIT WALK (BW)

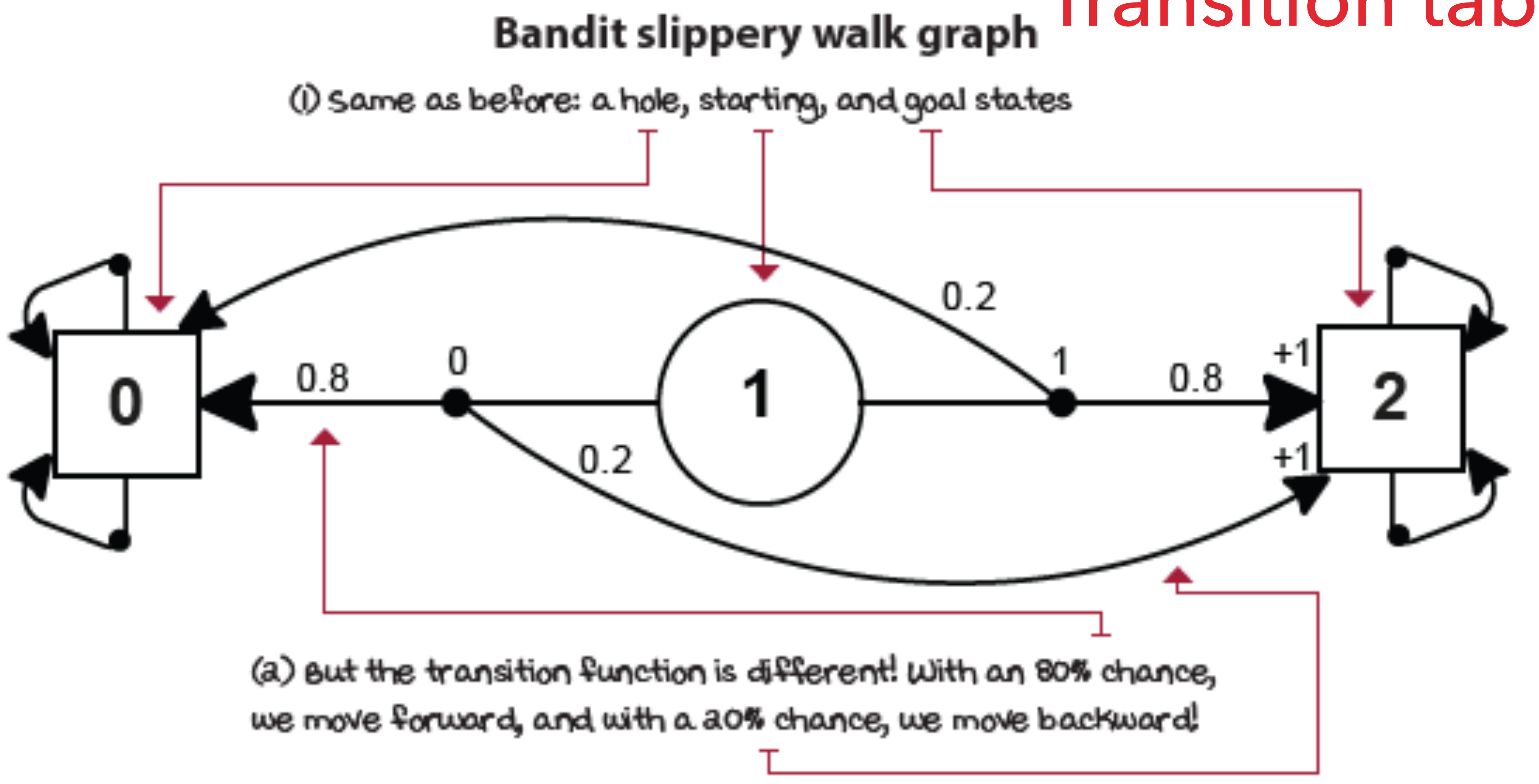


State	Action	Next state	Transition probability	Reward signal
0 (Hole)	0 (Left)	0 (Hole)	1.0	0
0 (Hole)	1 (Right)	0 (Hole)	1.0	0
1 (Start)	0 (Left)	0 (Hole)	1.0	0
1 (Start)	1 (Right)	2 (Goal)	1.0	+1
2 (Goal)	0 (Left)	2 (Goal)	1.0	0
2 (Goal)	1 (Right)	2 (Goal)	1.0	0

BANDIT SLIPPERY WALK (BSW)

- This a stochastic version of BW
- Surface of the walk is slippery : Each action there a 20% chance that the agent can go backward.

Transition table ??

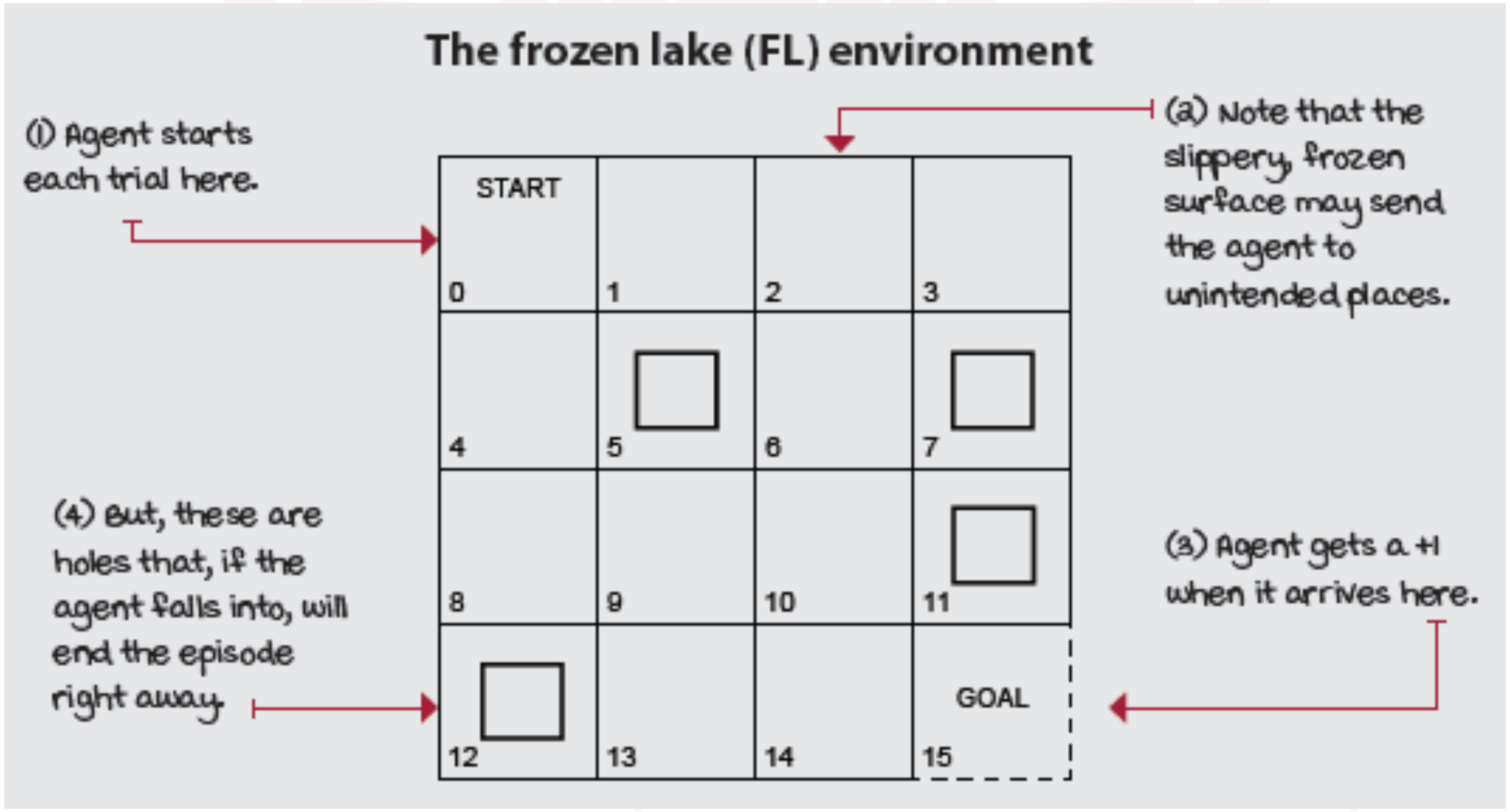

Bandit slippery walk graph
(1) Same as before: a hole, starting, and goal states
0
1
2
0.8
0
0.2
0.2
1
0.8
+1
+1
(2) But the transition function is different! With an 80% chance, we move forward, and with a 20% chance, we move backward!

BANDIT SLIPPERY WALK (BSW)

State	Action	Next state	Transition probability	Reward signal
0 (Hole)	0 (Left)	0 (Hole)	1.0	0
0 (Hole)	1 (Right)	0 (Hole)	1.0	0
1 (Start)	0 (Left)	0 (Hole)	0.8	0
1 (Start)	0 (Left)	2 (Goal)	0.2	+1
1 (Start)	1 (Right)	2 (Goal)	0.8	+1
1 (Start)	1 (Right)	0 (Hole)	0.2	0
2 (Goal)	0 (Left)	2 (Goal)	1.0	0
2 (Goal)	1 (Right)	2 (Goal)	1.0	0

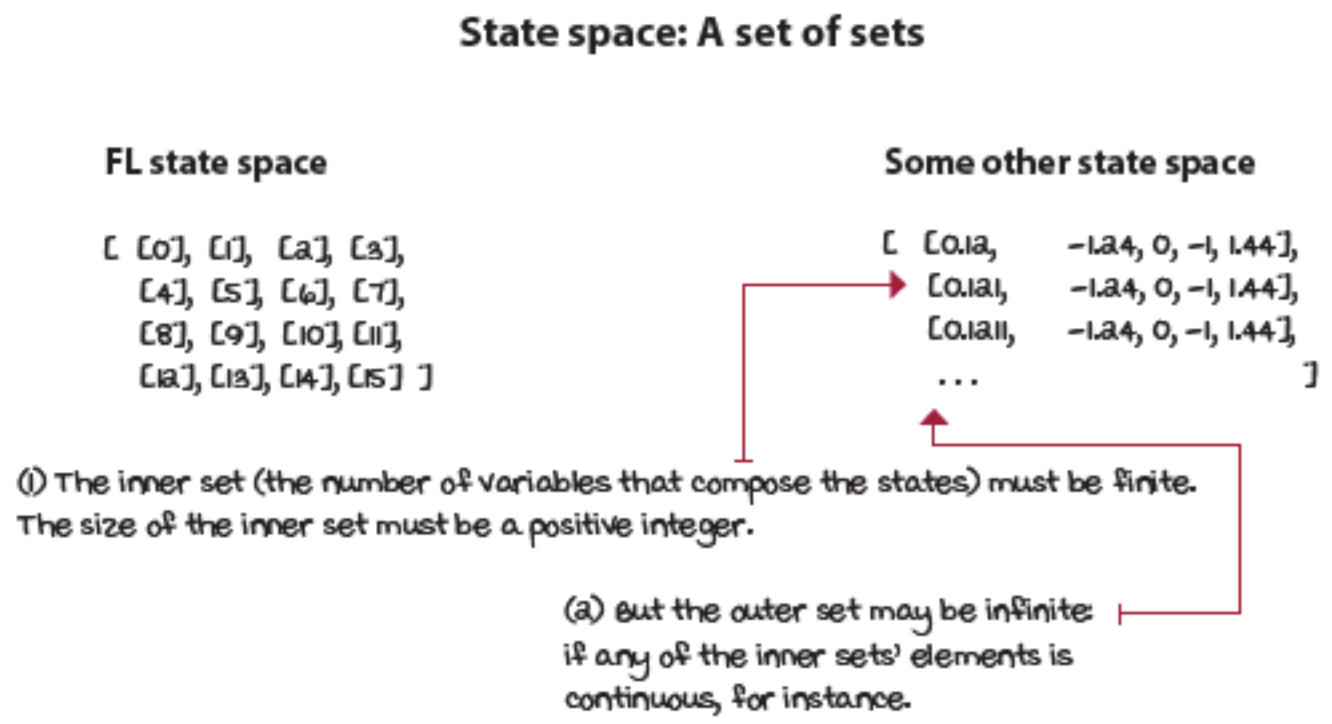
FROZEN LAKE

- It is a GW environment.
- The task in the FL environment is similar to the task in the BW and BSW environments: to go from a start location to a goal location while avoiding falling into holes.
- The challenge is similar to the BSW, in that the surface of the FL environment is slippery, it's a frozen lake after all. But the environment itself is larger.


The frozen lake (FL) environment
(1) Agent starts each trial here.
(2) Note that the slippery, frozen surface may send the agent to unintended places.
START
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
GOAL
(3) Agent gets a +1 when it arrives here.
(4) But, these are holes that, if the agent falls into, will end the episode right away.

FROZEN LAKE – STATES

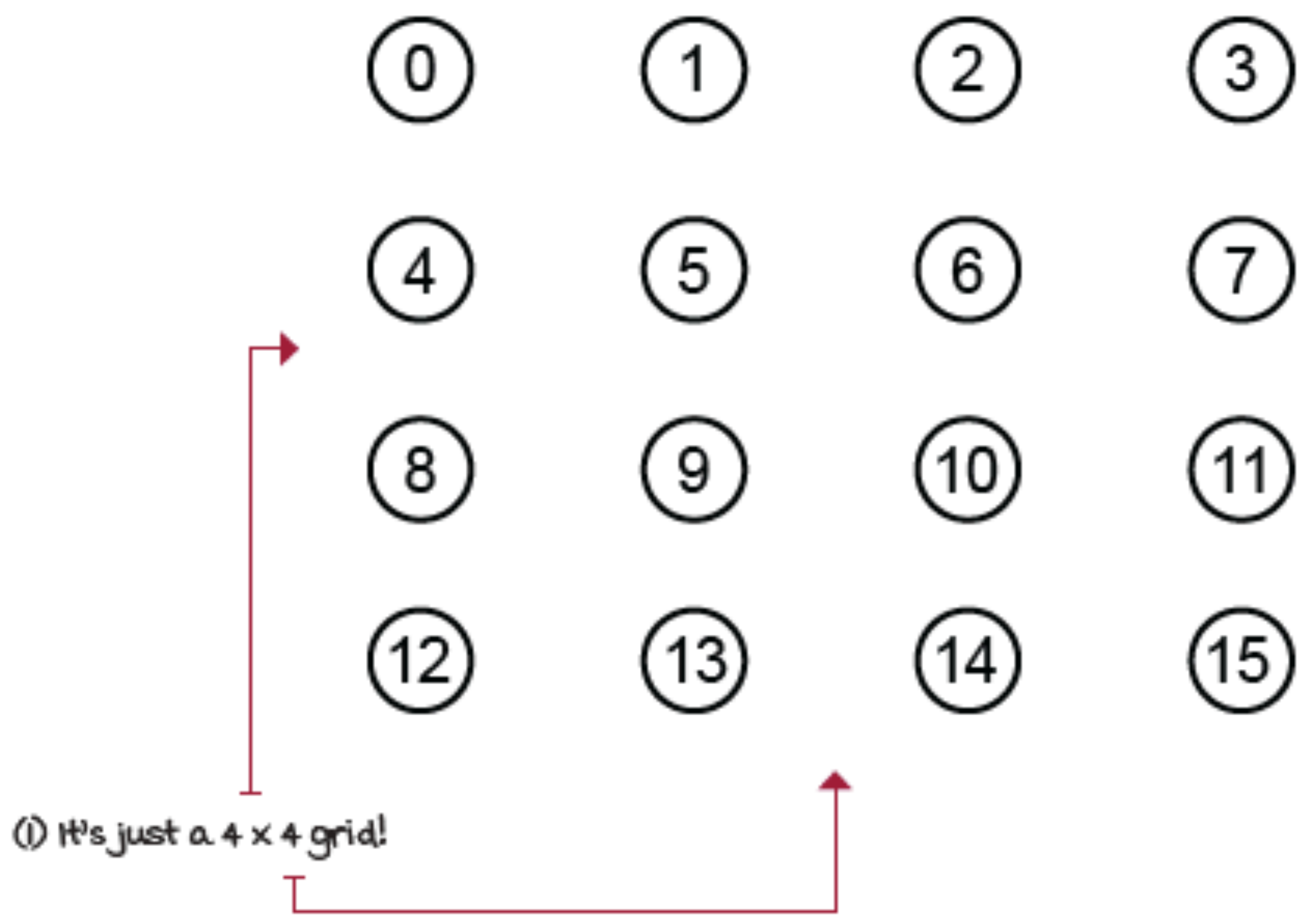
- A state is a unique and self-contained configuration of the problem.
- The set of all possible states, the state space, is defined as the set S. The state space can be finite or infinite.
- State space is different than the set of variables that compose a single state. This set should always be finite.
- In the end, the state space is a set of sets. The inner set must be of equal size and finite, as it contains the number of variables representing the states.
- The outer set can be infinite depending on the types of elements of the inner sets.


State space: A set of sets
FL state space
[[0], [1], [2], [3],
[4], [5], [6], [7],
[8], [9], [10], [11],
[12], [13], [14], [15]]
Some other state space
[[0.12, -1.24, 0, -1, 1.44],
[0.121, -1.24, 0, -1, 1.44],
[0.1211, -1.24, 0, -1, 1.44],
...]
(1) The inner set (the number of variables that compose the states) must be finite. The size of the inner set must be a positive integer.
(2) But the outer set may be infinite: if any of the inner sets' elements is continuous, for instance.

FROZEN LAKE – STATES

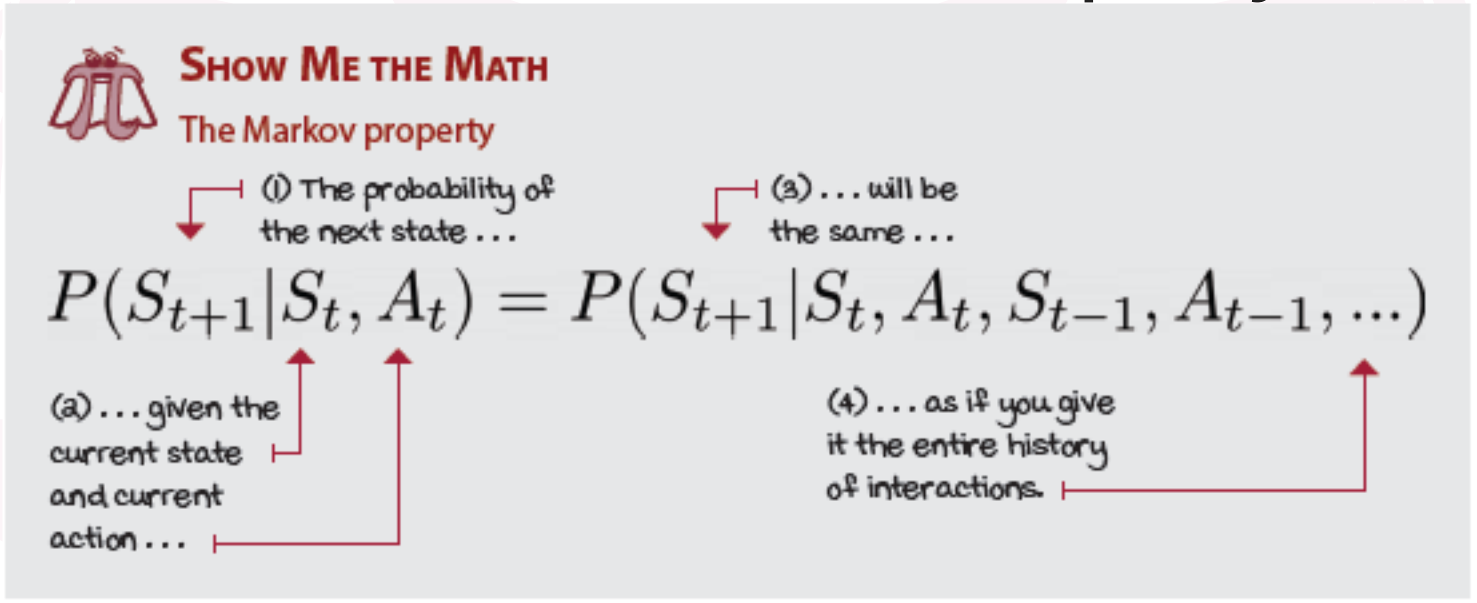
- For the BW, BSW, and FL environments, the state is composed of a single variable containing the id of the cell where the agent is at any given time.

States in the FL contain a single variable indicating the id of the cell in which the agent is at any given time step



FROZEN LAKE – STATES

- MDPs are fully observable - Observation and states are same
- Partially observable Markov decision processes (POMDPs) - Observation are the only thing the agent can see instead of the state.
- BW, BSW and FL are?? MDP
- Are BW, BSW, FL contain Markov Property ?? Yes

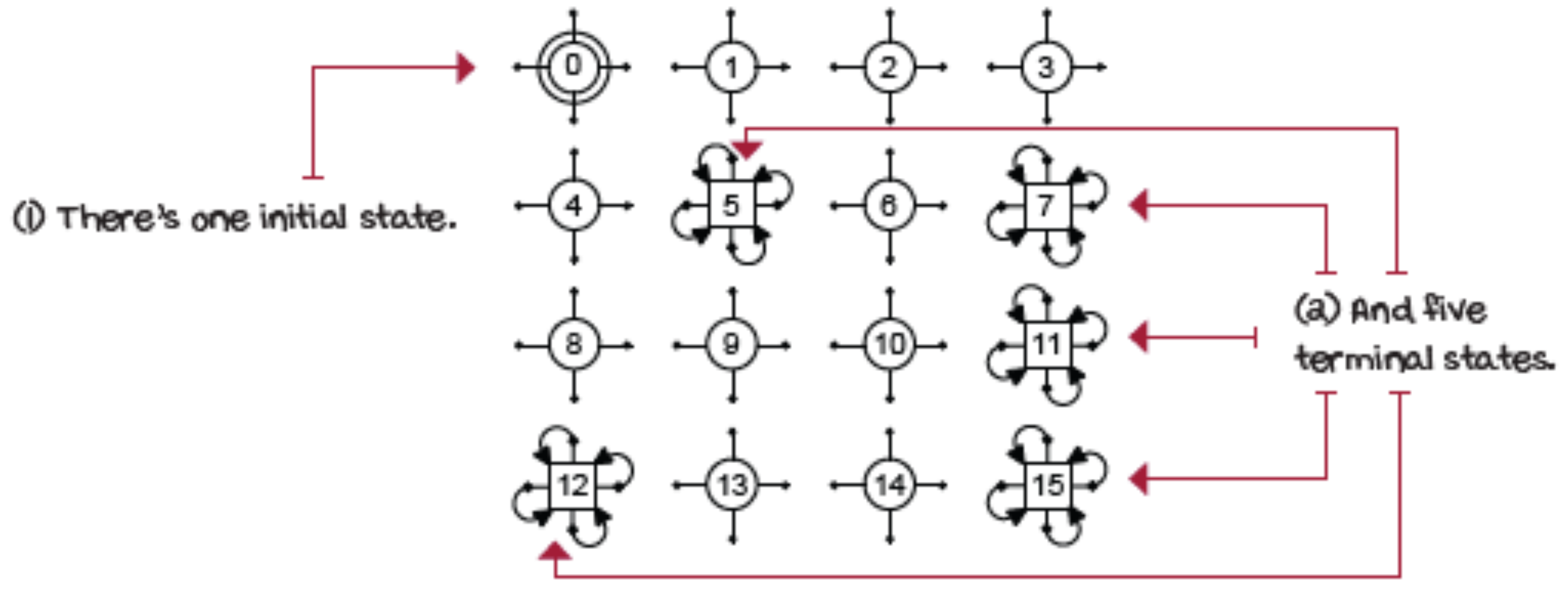

Show Me the Math
The Markov property
(1) The probability of the next state . . .
(3) . . . will be the same . . .
(2) . . . given the current state and current action . . .
(4) . . . as if you give it the entire history of interactions.

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ...)$$

FROZEN LAKE – STATES

- State Space: S^+
- S^i - Initial States - Subset of S^+
- S - Absorption State/Transition State - Subset of S^+
- A terminal state is a special state: it must have all available actions transitioning, with probability 1, to itself, and these transitions must provide no reward.

States in the frozen lake environment



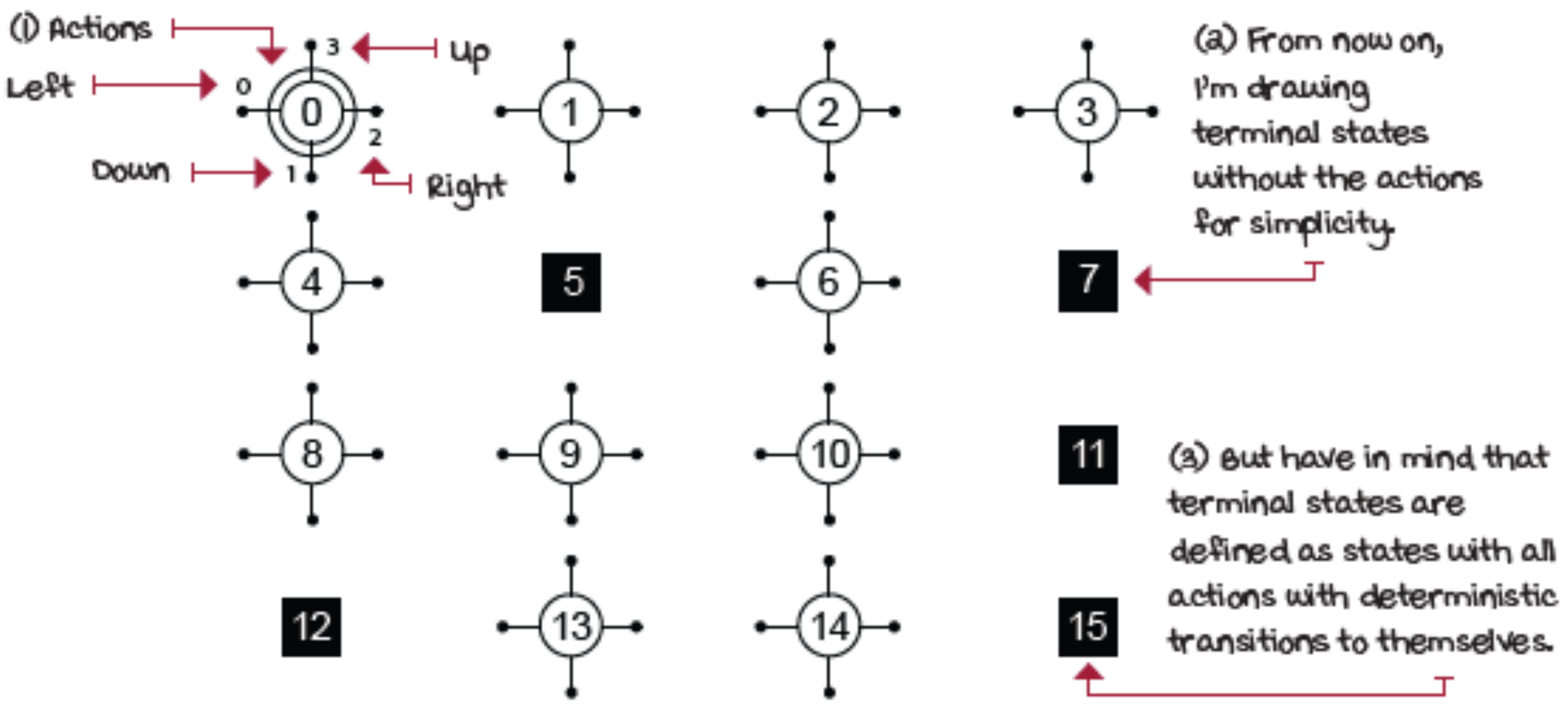
FROZEN LAKE – ACTIONS

- Action A(s) - A set of actions that depends on the state.
- If Needed A - A set of actions common to all the possible states.
- Unlike the number of state variables, the number of variables that compose an action may not be constant.
- Actions can be finite or infinite. The environment makes the set of all available actions known in advance. Agents can select actions either deterministically or stochastically.

FROZEN LAKE – ACTIONS

- In the BW, BSW, and FL environments, actions are singletons representing the direction the agent will attempt to move.
- In FL, there are four available actions in all states: Up, Down, Right, or Left. There's one variable per action, and the size of the action space is four.

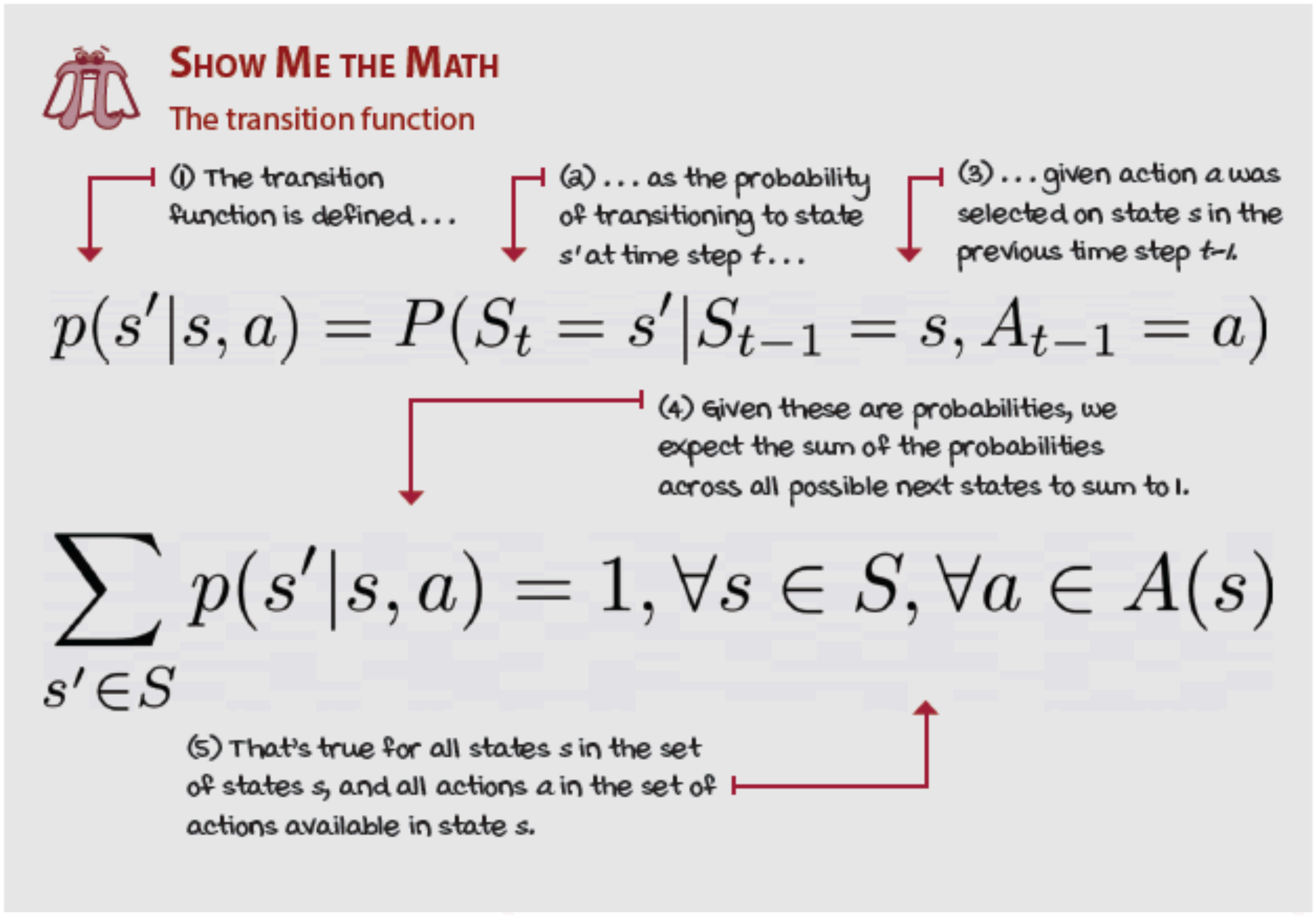
The frozen lake environment has four simple move actions



(1) Actions
Left
Down
Up
Right
0
1
2
3
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
(2) From now on, I'm drawing terminal states without the actions for simplicity.
(3) But have in mind that terminal states are defined as states with all actions with deterministic transitions to themselves.

FROZEN LAKE – TRANSITION FUNCTION

- It is the consequence of agent actions.
- The way the environment changes as a response to actions is referred to as the state-transition probabilities, or more simply, the transition function, and is denoted by T(s, a, s').



Show Me the Math

The transition function

(1) The transition function is defined . . .

(2) . . . as the probability of transitioning to state s' at time step t . . .

(3) . . . given action a was selected on state s in the previous time step t-1.

$$p(s'|s,a) = P(S_t = s' | S_{t-1} = s, A_{t-1} = a)$$

(4) Given these are probabilities, we expect the sum of the probabilities across all possible next states to sum to 1.

$$\sum_{s' \in S} p(s'|s,a) = 1, \forall s \in S, \forall a \in A(s)$$

(5) That's true for all states s in the set of states s, and all actions a in the set of actions available in state s.

FROZEN LAKE – TRANSITION FUNCTION

- One key assumption of many RL (and DRL) algorithms is that this distribution is stationary.
- That is, while there may be highly stochastic transitions, the probability distribution may not change during training or evaluation.
- In the FL environment, we know that there's a 33.3% chance we'll transition to the intended cell (state) and a 66.6% chance we'll transition to orthogonal directions. There's also a chance we'll bounce back to the state we're coming from if it's next to the wall.

	33.3%	
33.3%	S	33.3%
	0%	

Visualize it as the agent facing directly to the intended cell

FROZEN LAKE – TRANSITION FUNCTION

The transition function of the frozen lake environment

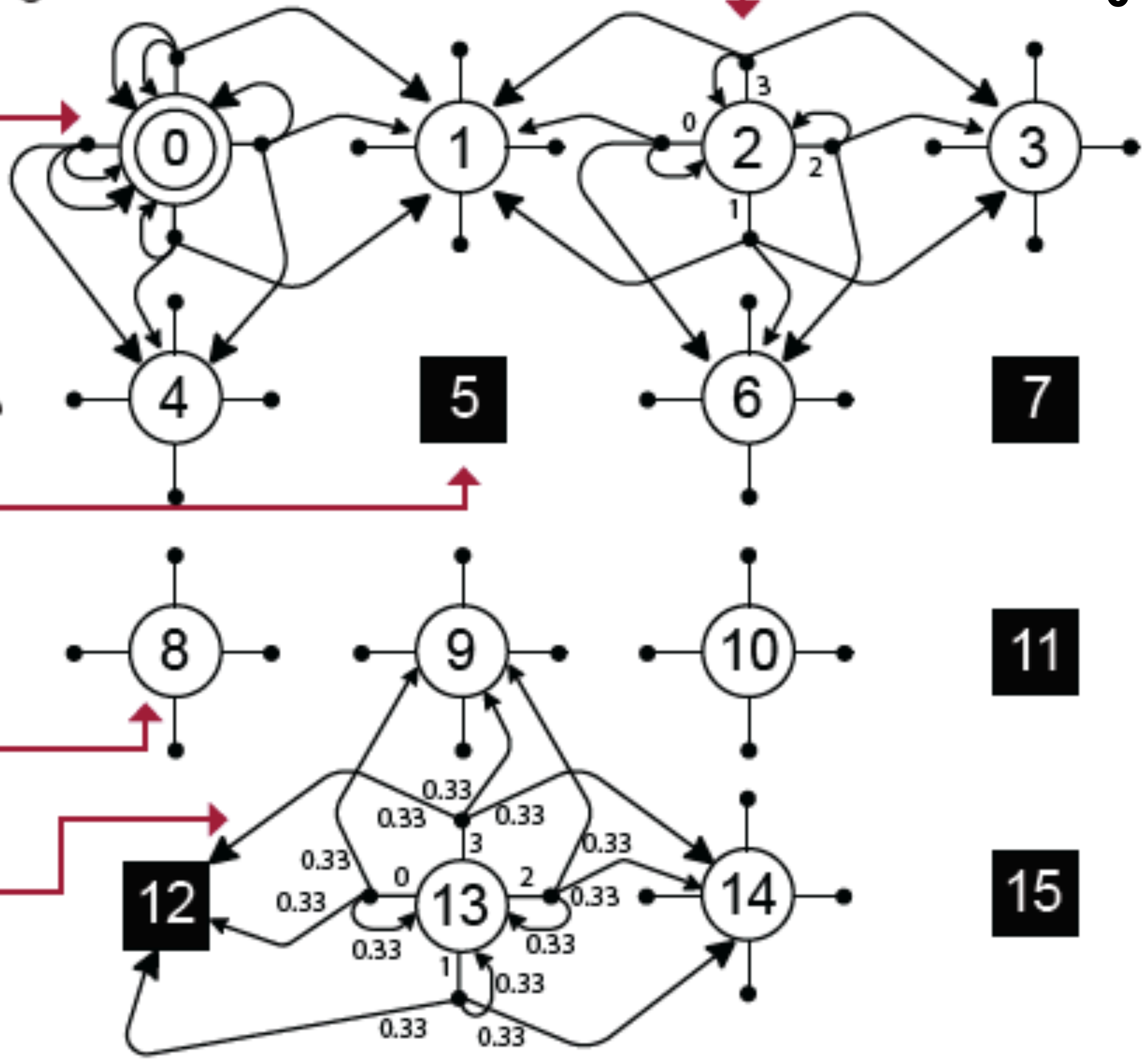
(1) Without probabilities for clarity

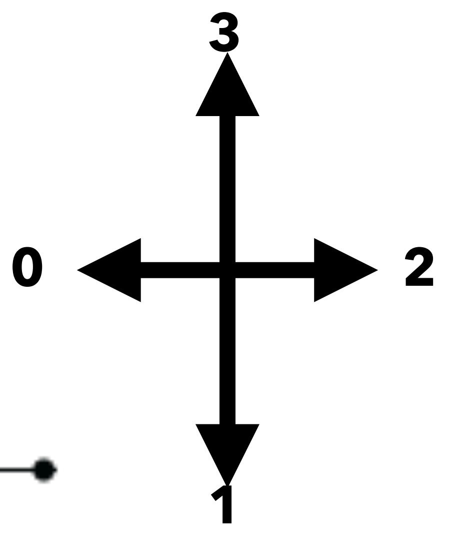
(2) Notice that the corner states are special. You bounce back from the horizontal and the vertical walls.

(3) Remember that terminal states have all transitions from all actions looping back to themselves with probability 1.

(4) I'm not drawing all the transitions, of course. This state, for instance, isn't complete.

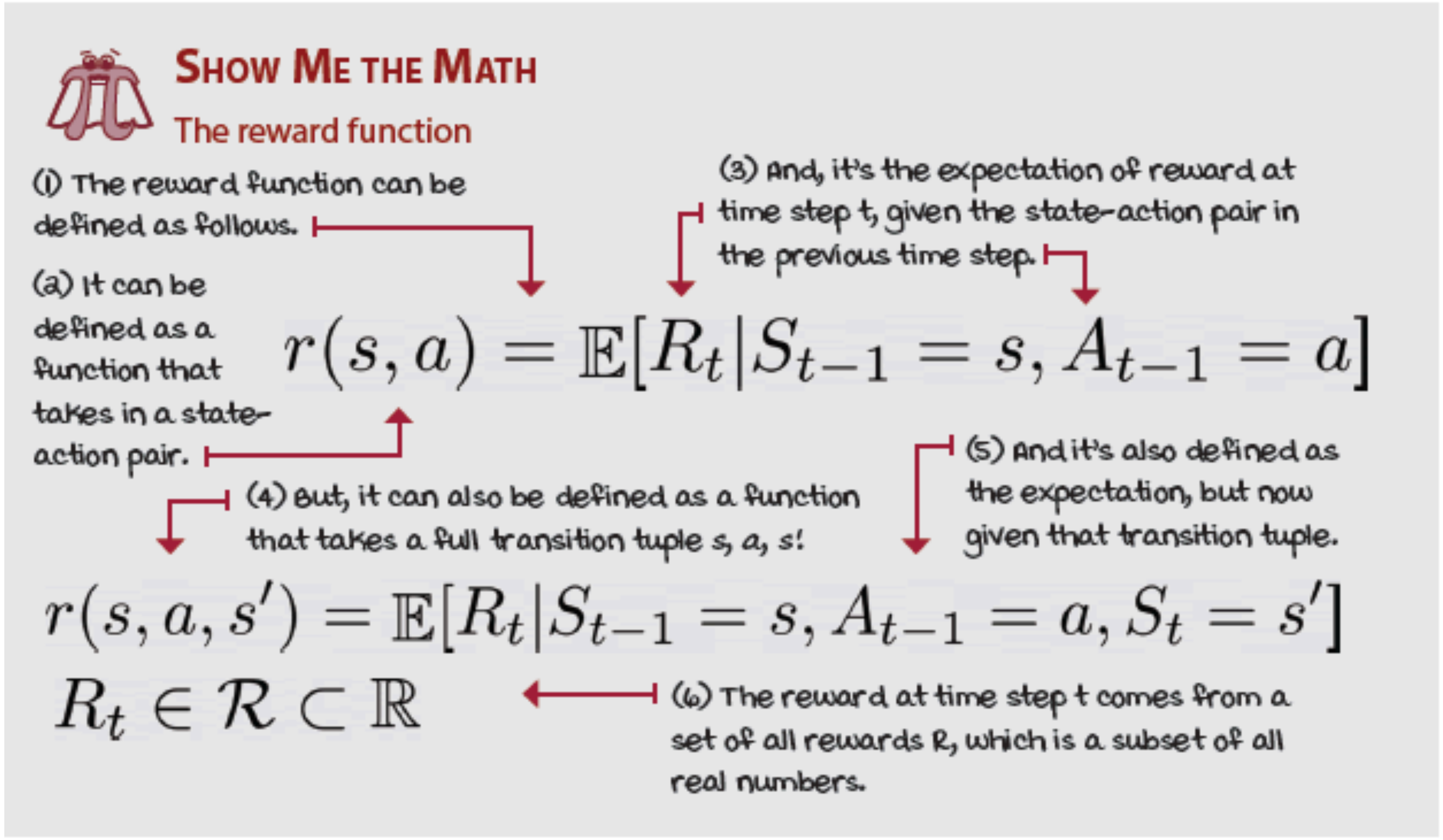
(5) This environment is highly stochastic!





FROZEN LAKE - REWARDS

- The reward function R maps a transition tuple s, a, s' to a scalar.
- The reward function gives a numeric signal of goodness to transitions. When the signal is positive, we can think of the reward as an income or a reward.
- rewards can also be negative, and we can see these as cost, punishment, or penalty.



SHOW ME THE MATH

The reward function

(1) The reward function can be defined as follows.

(2) It can be defined as a function that takes in a state-action pair.

(3) And, it's the expectation of reward at time step t, given the state-action pair in the previous time step.

$$r(s,a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$$

(4) But, it can also be defined as a function that takes a full transition tuple s, a, s!

(5) And it's also defined as the expectation, but now given that transition tuple.

$$r(s,a,s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$$

$$R_t \in \mathcal{R} \subset \mathbb{R}$$

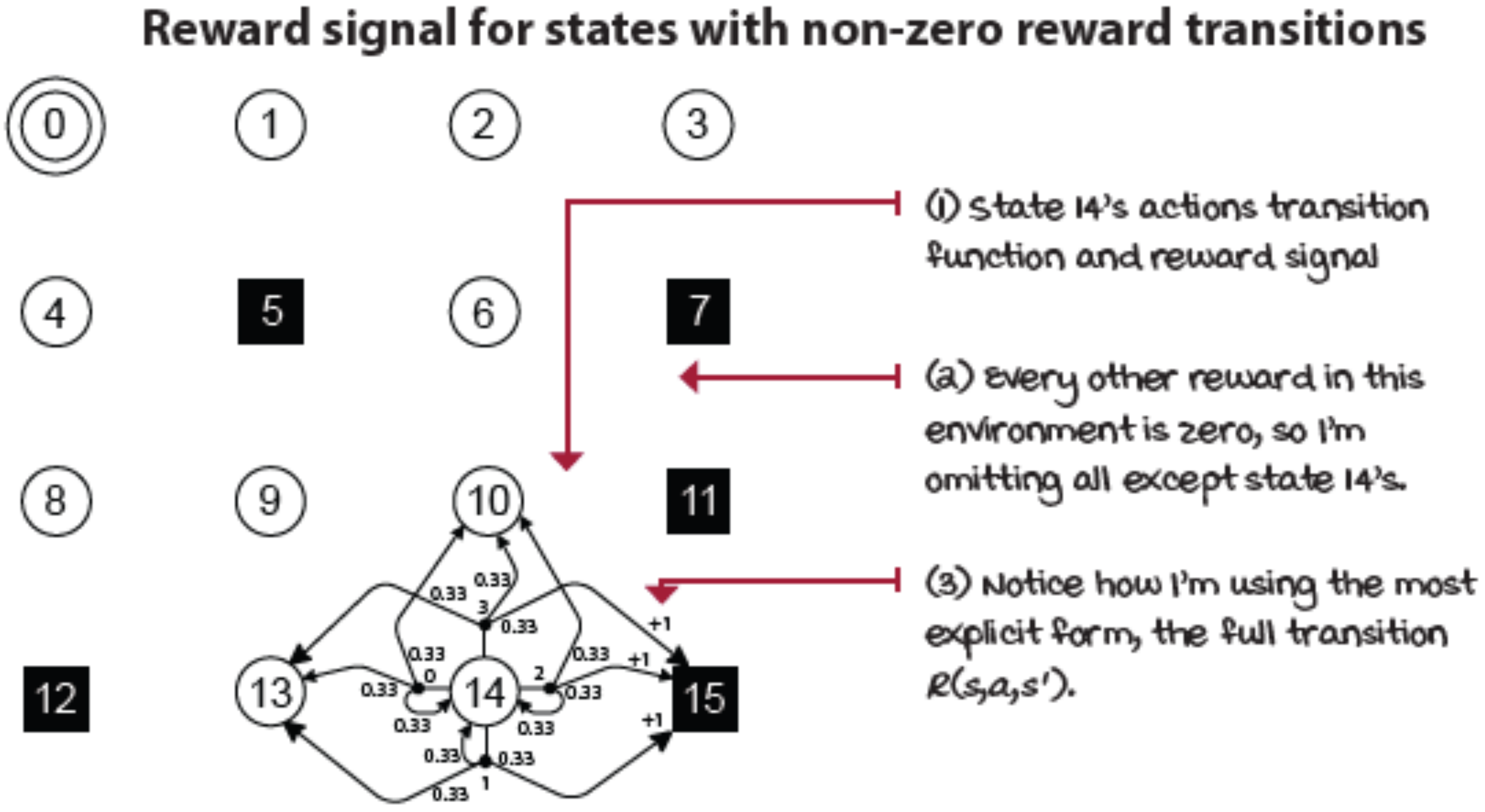
(6) The reward at time step t comes from a set of all rewards R, which is a subset of all real numbers.

FROZEN LAKE – REWARDS

- In the FL environment, the reward function is +1 for landing in state 15, 0 otherwise.
- There are only three ways to land on 15.
  - (1) Selecting the Right action in state 14 will transition the agent with 33.3% chance there (33.3% to state 10 and 33.3% back to 14).
  - (2) selecting the Up
  - (3) the Down action from state 14 will unintentionally also transition the agent there with 33.3% probability for each action.

Transition

table ??



Reward signal for states with non-zero reward transitions
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0.33
+1
0
1
2
3
(1) State 14's actions transition function and reward signal
(2) Every other reward in this environment is zero, so I'm omitting all except state 14's.
(3) Notice how I'm using the most explicit form, the full transition R(s,a,s').

FROZEN LAKE – TABLE

State	Action	Next state	Transition probability	Reward signal
0	Left	0	0.33	0
0	Left	0	0.33	0
0	Left	4	0.33	0
0	Down	0	0.33	0
0	Down	4	0.33	0
0	Down	1	0.33	0
0	Right	4	0.33	0
0	Right	1	0.33	0
0	Right	0	0.33	0
0	Up	1	0.33	0
0	Up	0	0.33	0
0	Up	0	0.33	0
1	Left	1	0.33	0
1	Left	0	0.33	0
1	Left	5	0.33	0
1	Down	0	0.33	0
1	Down	5	0.33	0
1	Down	2	0.33	0
1	Right	5	0.33	0
1	Right	2	0.33	0
1	Right	1	0.33	0

… and So on

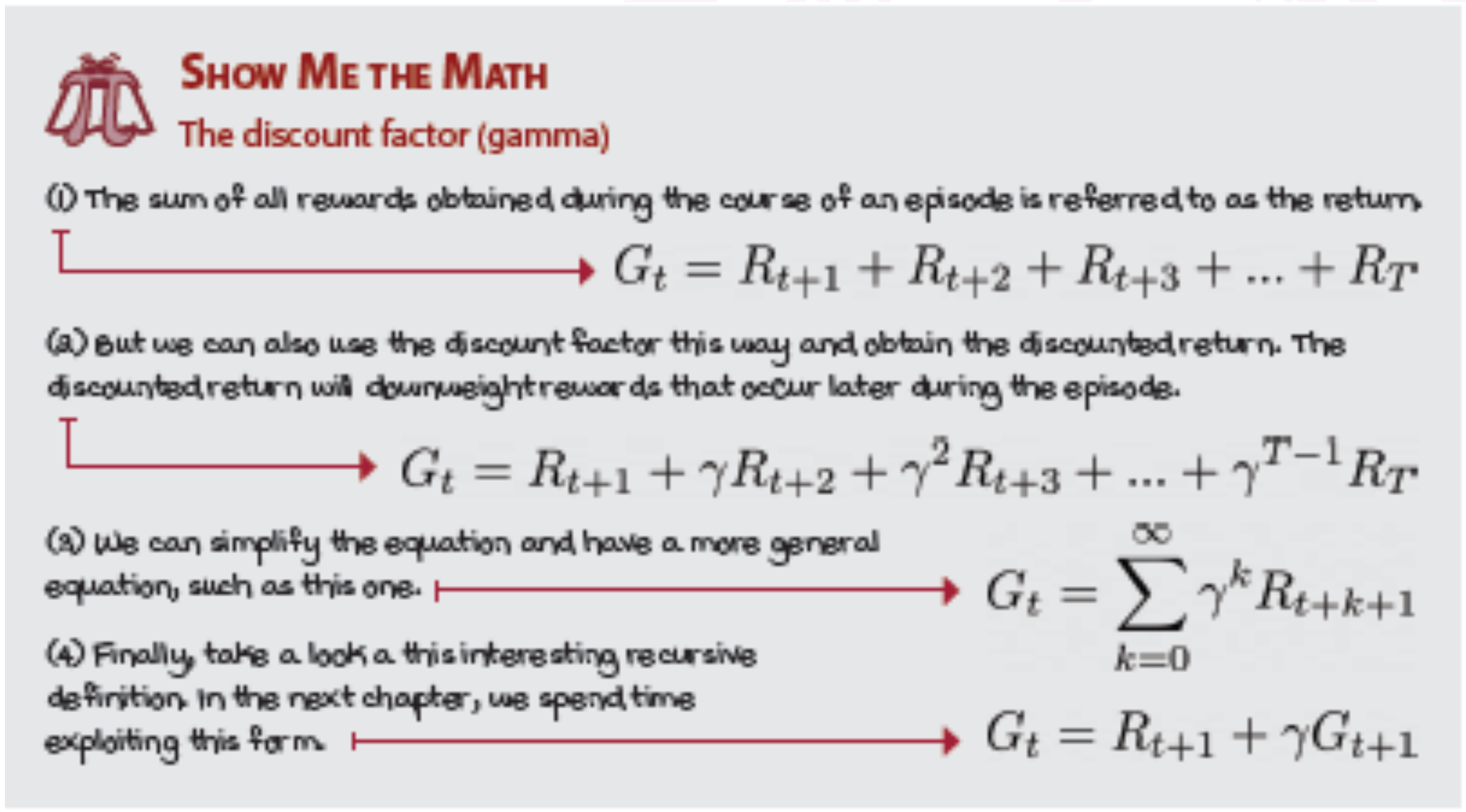
FROZEN LAKE – TIME STEP

- A time step, also referred to as epoch, cycle, iteration, or even interaction, is a global clock syncing all parties and discretizing time.
- Episodic and Continual tasks.
- Episodic and continuing tasks can also be defined from the agent's perspective. We call it the planning horizon.
    - Finite Horizon - Agent knows the task will terminate in a finite number of time steps. Eg: Frozen Lake
    - Greedy Horizon - If agent is forced to terminate with 15 time steps in FL
    - Infinite Horizon - Agent plans for infinite time steps but a manual terminal conditions will be give.
- FL is an indefinite planning horizon; the agent plans for infinite number of steps, but interactions may stop at any time.
-

FROZEN LAKE – DISCOUNTED REWARD

- Because of the possibility of infinite sequences of time steps in infinite horizon tasks, we need a way to discount the value of rewards over time; that is, we need a way to tell the agent that getting +1's is better sooner than later.
- We commonly use a positive real value less than one to exponentially discount the value of future rewards. The further into the future we receive the reward, the less valuable it is in the present.
- This number is called the discount factor, or gamma. The discount factor adjusts the importance of rewards over time.

FROZEN LAKE – DISCOUNTED REWARD



Show Me the Math

The discount factor (gamma)

(1) The sum of all rewards obtained during the course of an episode is referred to as the return.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

(2) But we can also use the discount factor this way and obtain the discounted return. The discounted return will downweight rewards that occur later during the episode.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

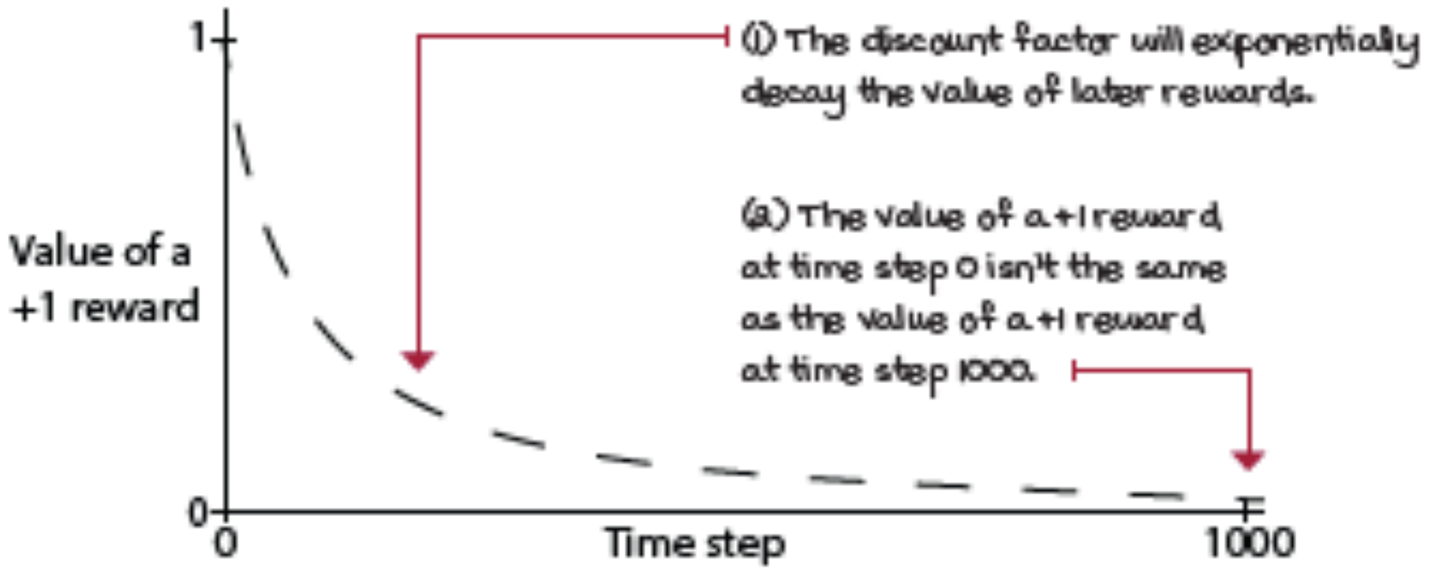
(3) We can simplify the equation and have a more general equation, such as this one.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(4) Finally, take a look a this interesting recursive definition. In the next chapter, we spend time exploiting this form.

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Effect of discount factor and time on the value of rewards



(1) The discount factor will exponentially decay the value of later rewards.

(2) The value of a +1 reward at time step 0 isn't the same as the value of a +1 reward at time step 1000.

POLICIES AND VALUE FUNCTIONS

- Function of states (or of state-action pairs) that 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).
- The value of a state is the expected return starting from that state. It depends on the agent's policy
- State-value function for policy π

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{for all s} \in \mathcal{S}$$

POLICIES AND VALUE FUNCTIONS

- The value of taking an action in a state under policy π is the expected return starting from that state, taking that action, and thereafter following π
- Action-value function for policy π

$$q_\pi(s,a) \doteq \mathbb{E}[G_t \,|\, S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a\right]$$

- v_π and q_π can be estimated from experience
  - Calculating and maintaining average for each state and actual returns overtime the average will converge to $v_\pi(s)$ and $q_\pi(s,a)$
  - With manu states, averaging will be in not practical, instead the agent will maintain v_π and q_π as parameterize functions with less parameters than states

POLICIES AND VALUE FUNCTIONS

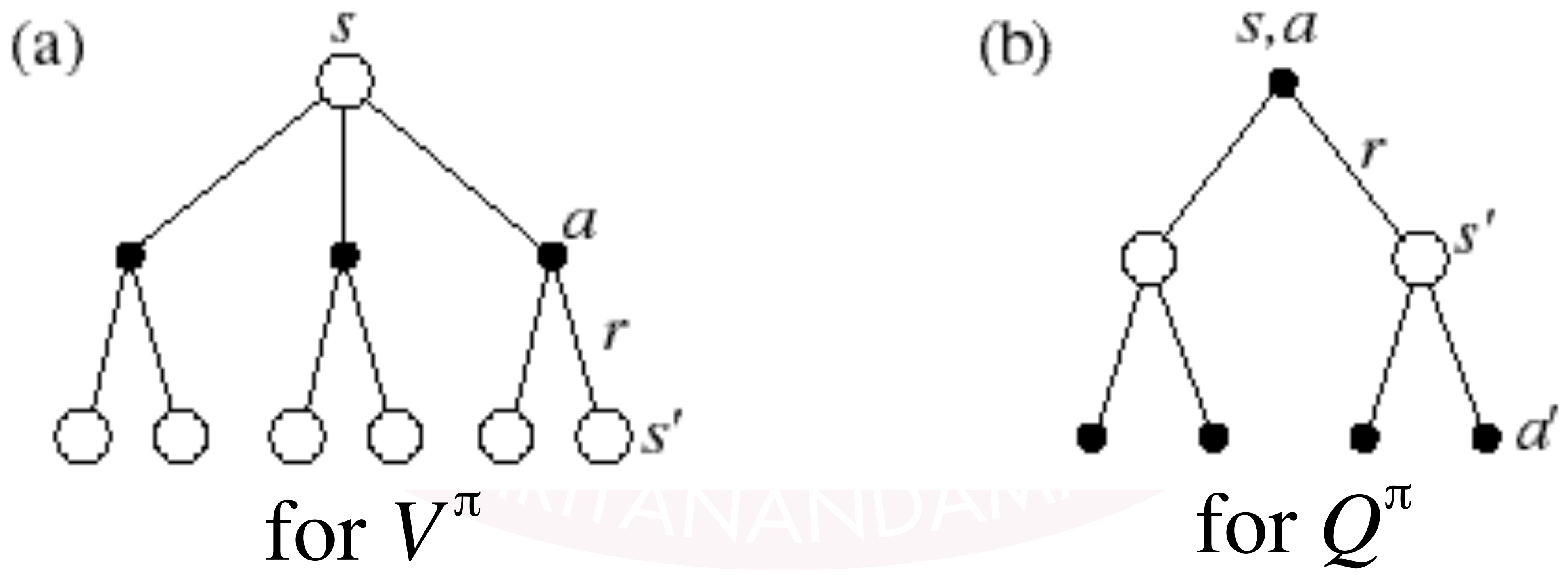
- Value functions satisfy recessive relationships similar to the returns function.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big], \text{for all } s \in \mathcal{S}
\end{aligned}
$$

Bellman equation

BELLMAN EQUATION

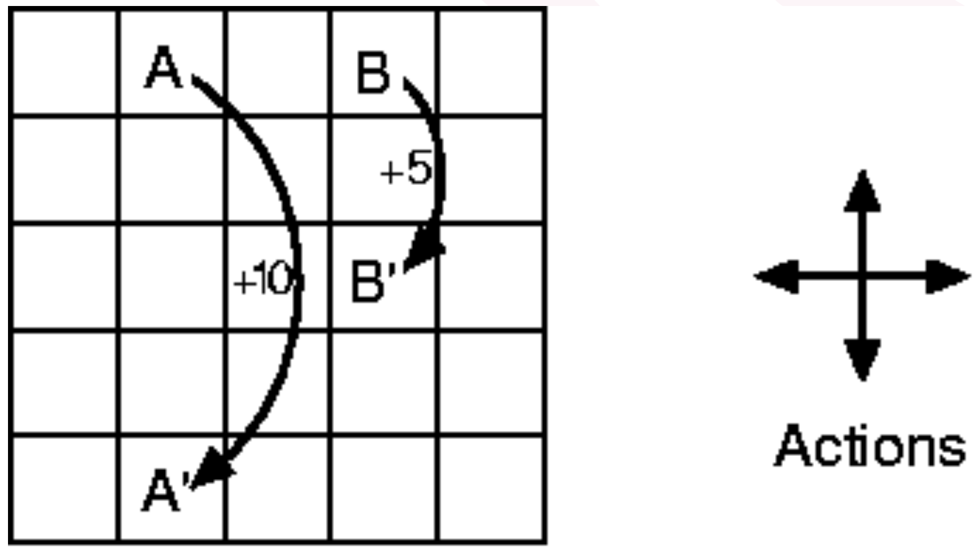
- It expresses a relationship between the value of a stateand the values of its successor states.
- Think of looking ahead from a state to its possible successor states.



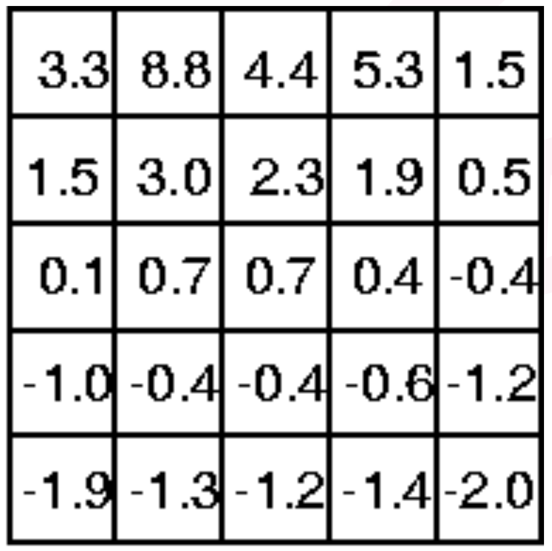
Backup diagrams

GRID WORLD

- Actions: north, south, east, west; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

OPTIMAL POLICIES AND OPTIMAL VALUE FUNCTIONS

- Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run.

- For finite MDPs policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an optimal policy. We denote them all π'.

- Optimal policies share the same optimal state-value function:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

- Optimal policies also share the same optimal action-value function: $Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$ for all $s \in S$ and $a \in A(s)$

■ This is the **expected return for taking action a in state s and thereafter following an optimal policy**.

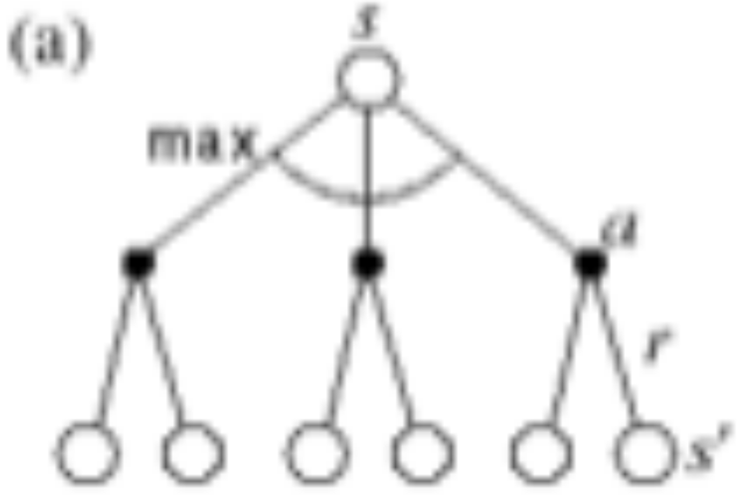
$$Q^{*}(s,a) = \mathbb{E}[R_{t+1} + \gamma v_{*}(S_{t+1}) \mid S_t = s, A_t = a]$$

BELLMAN OPTIMALITY EQUATION FOR V*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\ &= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\ &= \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right] \end{aligned}$$

The relevant backup diagram:

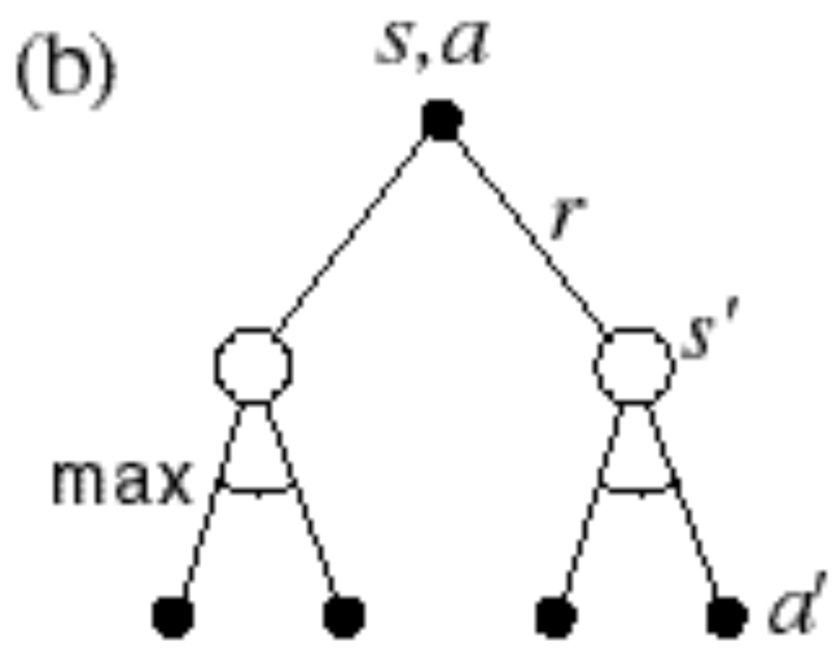


V^* is the unique solution of this system of nonlinear equations.

BELLMAN OPTIMALITY EQUATION FOR Q*

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]. \end{aligned}$$

The relevant backup diagram:



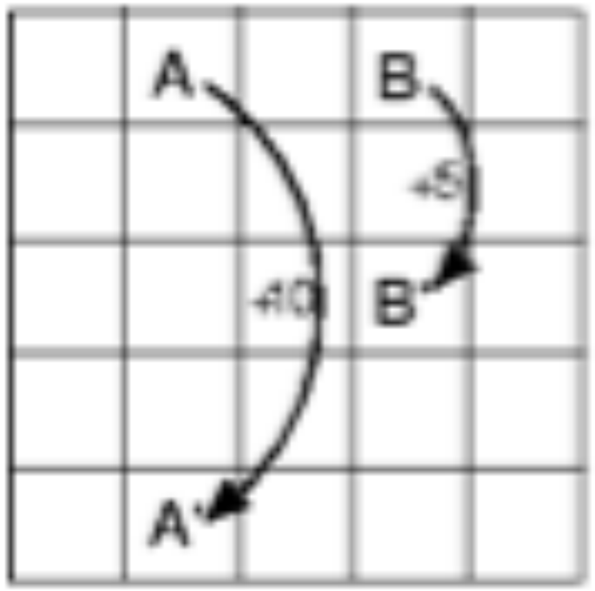
Q^* is the unique solution of this system of nonlinear equations.

WHY OPTIMAL STATE-VALUE FUNCTIONS ARE USEFUL

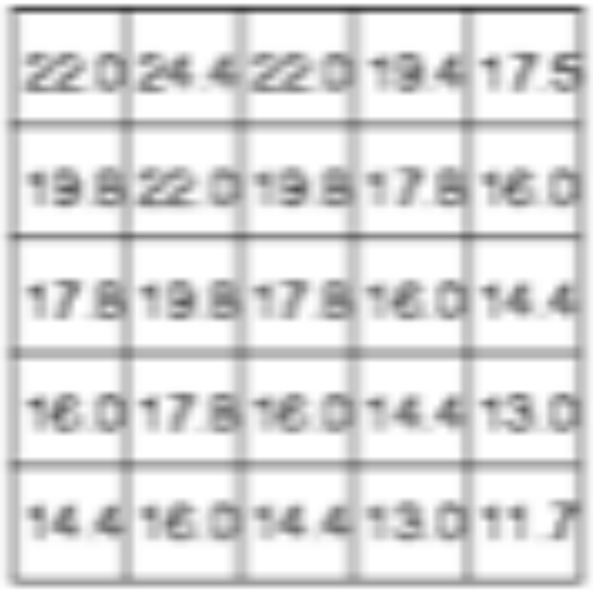
Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

E.g., back to the grid world:

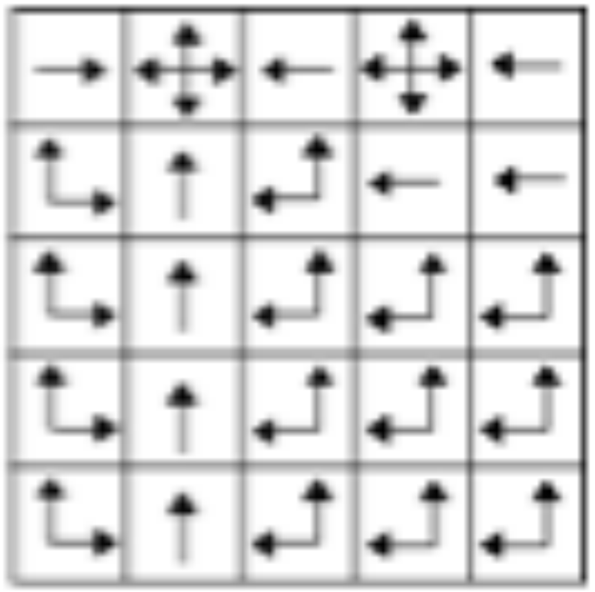


a) gridworld



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

SOLVING THE BELLMAN OPTIMALITY EQUATION

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space and time to do the computation;
  - the Markov Property.
- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods; see later),
  - BUT, number of states is often huge (e.g., backgammon has about 1020 states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

A SUMMARY

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards the agent tries to maximize
- Episodic and continuing tasks
- Markov Property
- Markov Decision Process
  - Transition probabilities
  - Expected rewards
- Value functions
  - State-value function for a policy
  - Action-value function for a policy
  - Optimal state-value function
  - Optimal action-value function
- Optimal value functions
- Optimal policies
- Bellman Equations
- The need for approximation

THANK YOU !!!