finite Markov Decision Process

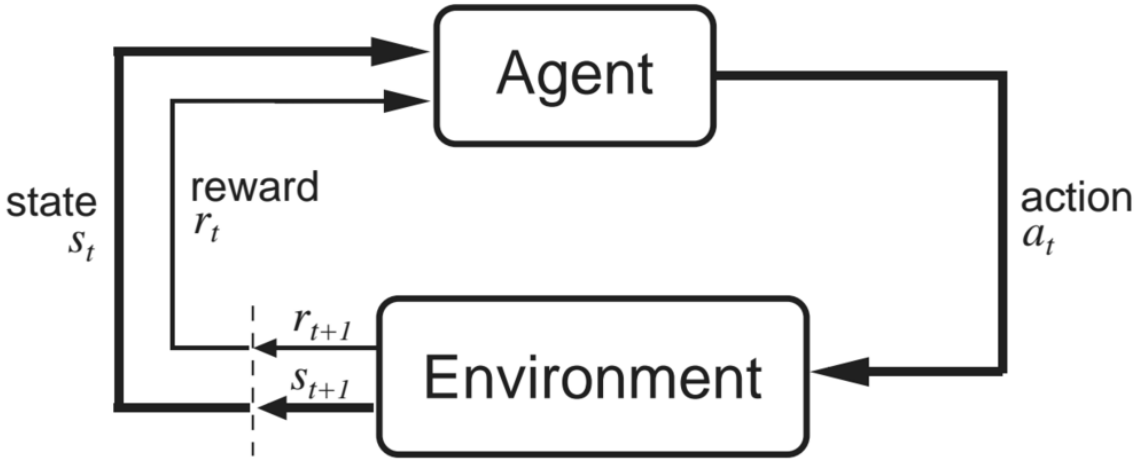
3.1 The Agent-Environment Interface

- **Agent**: 학습하는 대상, dicision maker (controller)
- enviroment: Agent 밖에서 agent와 interaction하는 모든것 (controlled system, plant)
- **State (S_t)**: step마다 받게되는 현재 상태로 상태에 따라 다른 action을 선택하게 된다.
  - enviroment의 모든것이 state의 요소로써 가치가 있거나 모두 사용되어야 하는 것은 아니다. 고려해야할 요소만 state로 넣는다.

- **Policy (π_t)**: state에서 각각의 action을 선택할 확률에 대한 정보
  - At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy.
  - $\pi_t(a|s)$: S_t일때 A_t일 확률

- **Action (A_t)**: agent가 어떤 state *s* 에서 선택할 수 있는 것. enviroment는 선택한 action에 대한 reward (R_{t+1})와 다음 step인 *t + 1* 에서의 state (S_{t+1})를 agent에게 준다.
- **Reward (R_t)**: action을 취했을때 받는 결과값으로 maximize해야하는 대상 (goal)
  - step *t* 에서의 어떤 action을 선택했을 때의 reward는 (R_{t+1}) 이다.

- task: reward, state, policy, action 등이 명확하게 갖추어져있는 Enviroment



- Example 3.1: Bioreactor

- Example 3.2: Pick-and-Place Robot
  로봇이 팔을 움직여 뭔가를 집고 어딘가 놓는것을 빠르고 부드럽게 학습시킨다고 했을때 집고 놓는것을 성공했다면 reward를 1을 줄 수 있는데 더 부드럽게 움직이는 것을 목표로 한다면 로봇이 움직임 움직임 마다 떨림이 있다면 그때마다 negative reward를 주는 것을 생각해볼 수 있다.

- Example 3.3: Recycling Robot
  캔을 주워서 쓰레기통에 버리는 로봇을 만드는데 action은 (1)'직접 캔을 찾기'와 (2)'캔을 가져다주기를 기다리기', (3)'배터리를 채우러 가기'가 있고 state는 (1)'배터리가 충분한 상태' (2)'배터리가 부족한 상태'가 있다. 그리고 reward는 빈캔을 주웠을때는 postive의 값을 받고 배터리가 없는 경우에는 negative 값을 받으며 나머지의 경우의 reward는 0으로 둘 수 있다.
  - '캔 찾기' action을 했을 때 캔을 찾지 못하는 경우는 생각하지 않는다.
    배터리가 소모되는 떨어지는 조건은 가정한다.

- Exercise 3.2: Is the reinforcement learning framework adequate to usefully represent all goal-directed learning tasks? Can you think of any clear exceptions?

3.2 Goals and Rewards

- reward
  - agent가 어떤 state에서 어떤 action을 했을 때 받게 되는 값으로 reinforcement-learning에서는 reward를 최대화 하는것을 목표로 한다. reward가 높을 수록 잘 학습된 것으로 볼 수 있다.
    (That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).)

- reward의 예
  - make a robot learn to walk: step마다 균형잡힌 움직임을 보일 때 positive reward를 준다.
  - maze: 목적지를 벗어 날때마다 reward를 -1씩 할 수 있다.
  - recyling bot: 보통의 경우 0이지만, 비어있는 캔을 찾을때 마다 +1의

reward를 준다.

- reward를 어떻게 줄 것인가에 따라서 당연히 결과가 크게 차이가 날 수 있다.

- **What you want it to achieve, not how you want it achieved.**
  reward의 대상은 '어떻게' 얻는지보다는 '무엇을' 얻고자 하는지에 초점을 맞춰야한다. 예를들어 체스의 목적은 '이기는것'이지 '상대의 말을 제거'하는 것이 아니다. 수단을 reward로 두는 것은 잘못된 결과를 만들어낸다.

- ?? it should not be able, for example, to simply decree that the reward has been received in the same way that it might arbitrarily change its actions. ??

3.3 Returns

- sum of the rewards after step *t*
  $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$ (T *is a final step*)
- episodes
  - agent와 environment의 interaction을 순차적으로 나타낼 수 있을 때(끝을 나타낼 수 있을때(?)) 그것을 **episodes** 라고 부른다. (e.g. 5 such as plays of a game, trips through a maze)
  - episode: 어떻게 끝이 나든 처음부터 끝까지 종결이 된 것.
  - 이기든 지든 게임이 끝나는 경우 게임이 끝난 상태를 terminal state라고 본다. 하나의 episode는 terminal state에서 끝이난다.
  - episode마다 지나온 state들이 다르고 reward는 다를 수 있다.

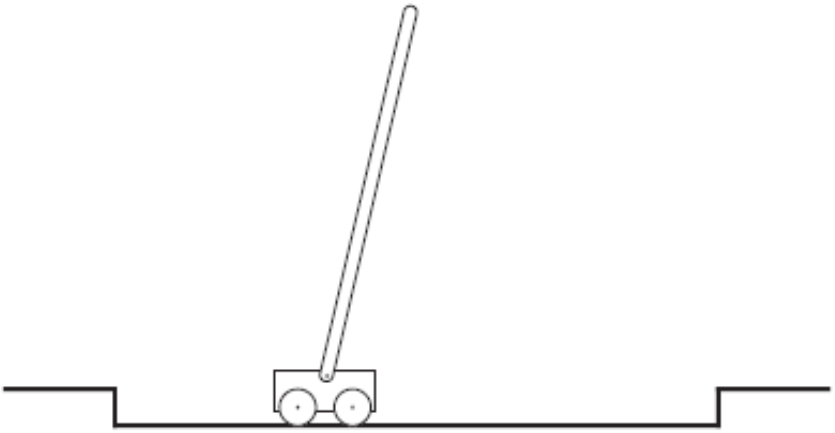
- **episodic-task**
  끝(terminal state)이 있어서 단계를 나타낼 수 있는 task

- **continuous-task** 와 **discouting rate**
  끝이 없이 계속 진행되는 task. (t = ∞) 이경우 일반적인 $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$ (T *is a final step*) 형태로 나타내기 어렵기 때문에 **discounting** 이라는 개념을 도입한다.
  A_t에서의 **discounted return**는

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

  로 나타낼 수 있고 여기서 γ(gamma) 를 **discouting rate** 라고 부른다. ($0 \le \gamma \le 1$)

- Example 3.4: Pole-Balancing



  - reward: 매번 step에서 넘어지지 않으면 +1, 쓰러질 시 -1을 준다.
  - 초기값으로 reset후 쓰러질때 까지를 하나의 episode로 볼 수 있다.

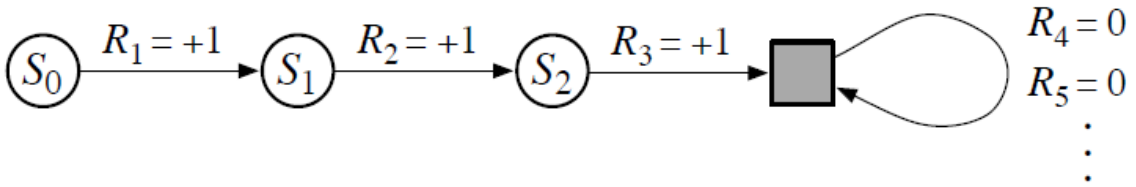
- Exercise 3.4
  failure에 대한 reward를 -1대신 0을 사용하면 어떻게 될까? -1을 사용한 경우와 어떻게 다를까? (수렴하는데 더 많은 시간이 걸리지 않을까?)

- Exercise 3.5
  maze의 경우에서 출구로 통과하는 것에 1의 reward를 주고 그밖에 나머지에는 0을 주기로 하고 실제로 이대로 학습을 시켜보면 학습이 잘 되지 않는데 이유가 무엇일까?

3.4 Unified Notation for Episodic and Continuing Tasks

- 실제로 S_t 보다는 $S_{t,i}$ (i는 episode)로 표기하는 것이 정확하나 일단 책에서는 항상 single episode의 상황에서 설명하고 있으므로 S_t로 표기

- episodic task나 continuing task나 아래의 그림으로 설명이 가능하다.



  사각형 box는 terminal state를 말하는데 episodic task의 경울 $G_t = R_1 + R_2 + R_3 + ... + R_t$으로 가능하다. 반면 continuing task의 경우 *t*, *t+1*, .. *t+k* 가 *k* 가 커짐에 따라 진행됨에 따라 각 γ의 *k* 승이 되므로 $k = \infty$ 은 0에 가까워진다. 따라서 두 task모두 아래와 같이 나타낼 수 있다. (chapter 10에서 더 자세히 다룸)

$$G_t \doteq \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

3.5 The Markov Property

막연히 현재를 *t* 라고 볼때 *t+1* 의 state인 *s'*, reward를 *r* 를 추정한다면 과거의 히스토리를 생각하여 아래처럼 나타낼 수 있다.

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$$

지나온 과거의 모든 state와 그때의 action 그리고 받은 reward들을 가지고 *t+1* 에서의 *s'* 와 *t* 를 구할 수있다고 생각할 수 있다. 하지만 실제로는 모든 과거의 데이터를 진행되는 모든 순간 계속하여 가지고 있는 것은 어렵기 때문에 여기서 **markov property** 를 사용한다.

Markov Property

> 과거와 현재 상태가 주어졌을 때의 미래 상태의 조건부 확률 분포가 과거 상태와는 독립적으로 현재 상태에 의해서만 결정된다는 것을 뜻한다.
> (A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process.)

state signal이 markov property를 사용한다고 가정하면

$$p(s', r|s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

의 형태로 나타내어 *s'* 와 *r* 을 추정할 수 있다. markov property는 특정 시점의 상태(state)가 해당 시점 이전의 상태들을 대신할수 있다고 가정하기 때문이다.
(the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.)

- ?? Markov state 가 정확히 무엇인지 (Markov property를 충족하는 state)??

- ?? Even when the state signal is non-Markov, it is still appropriate to think of the state in reinforcement learning as an approximation to a Markov state. ??
    - state signal이 non-Markov라도 해당 state를 reinforcement-learning에서

의 대략적인 markov state로 두고 생각하는 것이 좋다...?
    - 아주 완벽하게 markov state가 아니더라도 markov property를 사용하는 것이 문제를 해결하는데에 큰 역할을 한다.

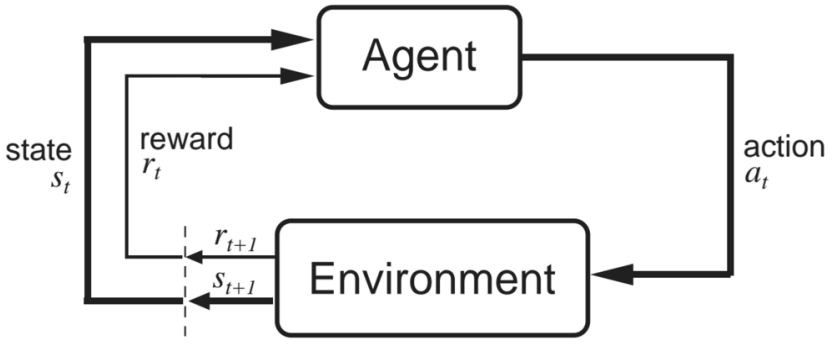
- 최대한 markov state를 완성할수록 reinforcement-learning의 결과는 좋아진다.

- Example 3.5: Pole-Balancing State
    - state를 모두 Markov state로 만들어서 모든 정보를 정확하게 안다면 이 문제를 이상적으로 해결할 수 있겠지만 실제 세계에서는 센서 측정 딜레이나 온도, joint의 마모된 정도등 다양한 변수들이 있기때문에 실제로 완벽하게 해결하는 것은 어렵지만 단순히 state signal을 오른쪽, 왼쪽, 가운데로 non-Markov한 state로 두어도 reinforment-learning을 통하여 효과적으로 Pole-Balancing문제를 해결할 수 있다. (markov state를 완벽하게 정의하지는 못했지만 괜찮은 성과가 나온 예)

- Exercise 3.6: Broken Vision System
    - 스스로가 자신이 vision system이라고 생각해보자. 카메라로 어떤 이미지가 들어왔을때 많은것을 볼 수 있지만 가려져 있는 것등 모든 사물을 볼 수는 없다. 첫번째 프레임을 본 후의 상황을 markov state로 볼 수 있는가? 카메라가 고장이 나서 아무것도 볼 수 없다면 이 경우에도 markove state를 적용해 볼 수 있는가?

3.6 Markov Decision Process

reinforment-learning에서는 **Markov property** 를 만족하는 task를 **Markov Decision Process** 라고 말한다. 그중에서도 state와 action이 유한한 task가 **finite Markov Decision Process** 다.



Markov property를 나타내는

$$p(s', r|s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

위의 표현은

rewards for state-action 의 관점에서는

$$r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

state-transition probabilities 의 관점으로는

$$p(s'|s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

rewards for state-action-next state 로는

$$r(s, a, s') \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r|s, a)}{p(s'|s, a)}$$

로 표현한다.

- Example 3.7: Recycling Robot MDP

$$\begin{aligned} \mathcal{A}(\texttt{high}) &\doteq \{\texttt{search}, \texttt{wait}\} \\ \mathcal{A}(\texttt{low}) &\doteq \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}. \end{aligned}$$

  when energy high
  - the energy level *high* with probability = α
  - the energy level *low* with probability = 1-α

  when energy low
  - the energy level *high* with probability = β
  - the energy level *low* with probability = 1-β

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

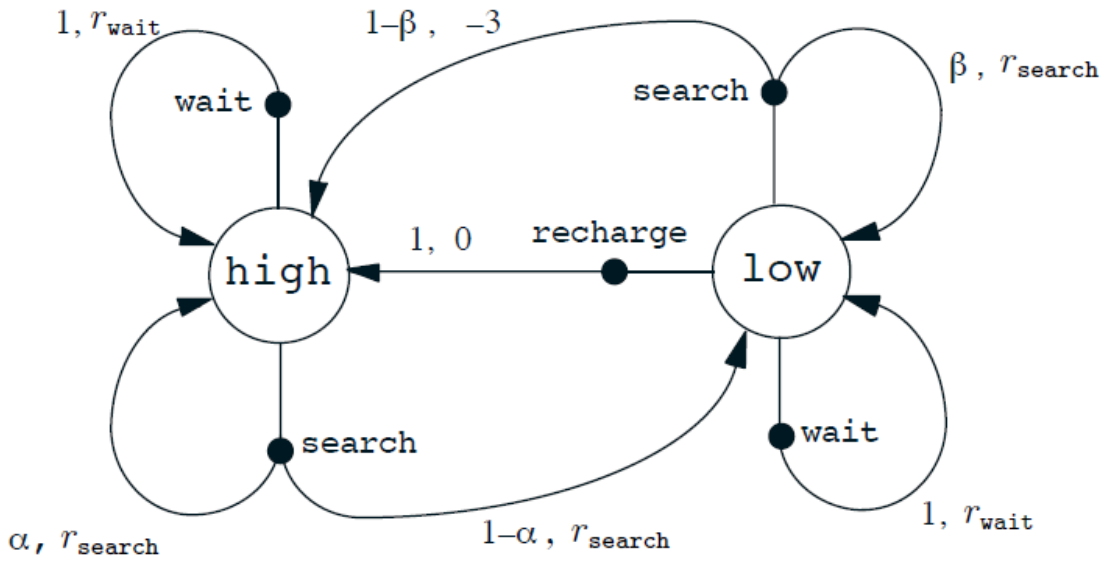


Figure 3.3: Transition graph for the recycling robot example.

3.7 Value Function

value function은 agent가 어떤 state에서 그 state에서의 policy를 이용하여 좋은 값의 reward를 예측해 내는게 하는가를 결정하는 역할을 한다. 여기서 **policy는 어떤 state에서 어떤 action을 선택할 확률** 이며π(a|s) 와 같이 표현한다.

- **state-value function for policy π, v_π**

어떤 state에서 어떤 policy(π)를 따를때 나오는 return(G_t)들의 합

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

- **action-value function for policy π, q_π**
  어떤 action을 어떤 state에서 어떤 policy(π)를 따를때 나오는 return들(G_t)의 합

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

  - Q-value로 표현하기도함. deep Q-network에서 Q가 action-value function을 말한다.

- Monte Carlo methods: v_π, q_π를 random sampleing하여 구함 (5장에서 다룬다.)

- **Bellman equation for v_π**
  어떤 state(s)의 value와 그 다음 state(s')의 value에 대한 관계를 나타낸 것
  (action value function을 구하는 방법)

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

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

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

?? For example, if an agent follows policy and maintains an average, for each

state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, v(s), as the number of times that state is encountered approaches infinity.

- **backup diagram**

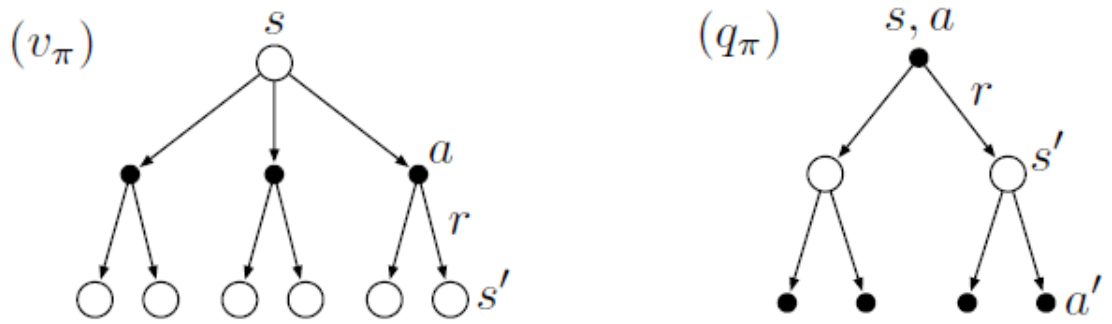


Figure 3.4: Backup diagrams for v_π and q_π.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S}, \qquad (3.12)
\end{aligned}
$$

3.8 Optimal Value Functions

optimal policy를 적용했을 때 얻게 되는 값. optimal policy는 discounted factor를 이용해 나중의 reward(G_t)까지를 고려한 최대의 reward를 내는 어떤 state s에서의 action a 를 선택하는 것을 말한다.

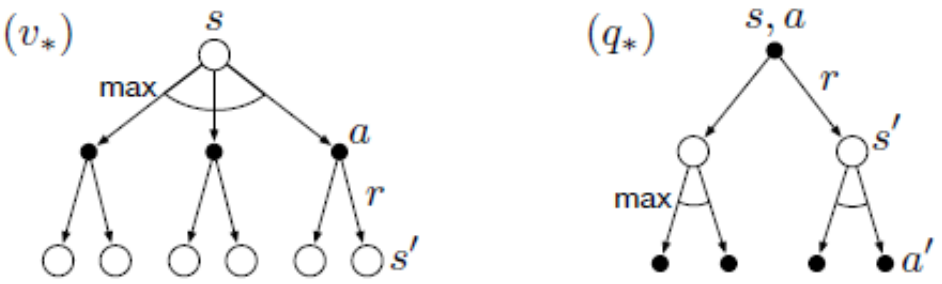
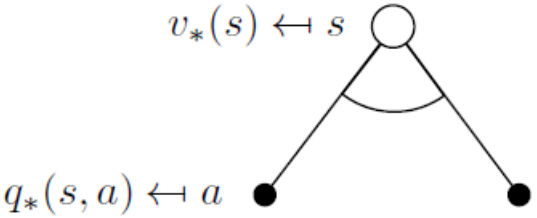


Figure 3.7: Backup diagrams for v_* and q_*

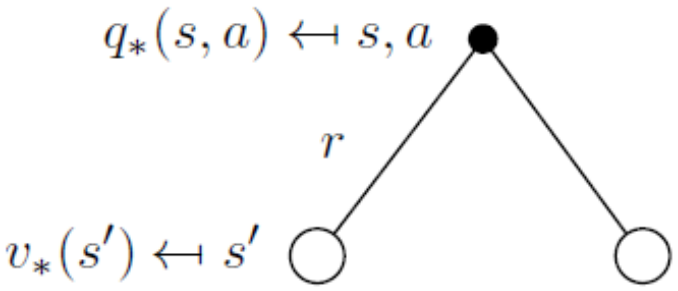
- optimal state-value function $V^*(s)$
  어떤 state에서 gready하게 action-value function이 최대(q*(s,a))가 되는 action을 선택하는 것.
  The optimal value functions are recursively related by the Bellman optimality equations:



$$v_*(s) = \max_a q_*(s,a)$$

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r|s, a)\big[r + \gamma v_*(s')\big].
\end{aligned}$$

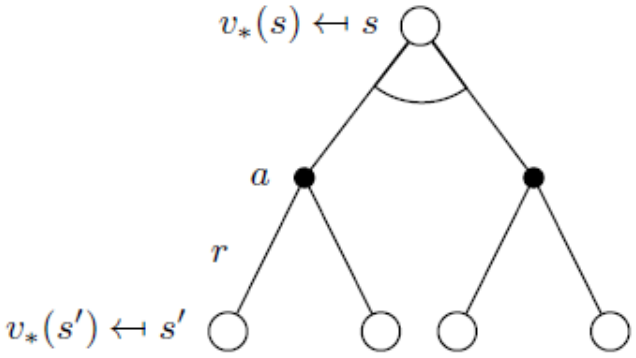
- optimal action-value function $q_*(s, a)$
  어떤 action에서 gready하게 state-value function이 최대($V^*(s)$)가 되는 state를 선택하는 것.



$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

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

두개를 합치면 optimal value function이 된다.



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

결국 reward function과 state transition probability를 알면 MDP를 완성하고 이것이 곧 reinforcement learning을 문제해결하는 것이라고 볼 수 있는데 interative하게 reward function과 state transition probability(policy)를 구해내는 것은 다음장인 Dynamic Programming에서 다룬다.

- Example 3.11: Bellman Optimality Equations for the Recycling Robot
  - actions: s (search), w (wait), re (recharge)
  - state: l (low), h (high)

$$\begin{aligned} v_*(\mathtt{h}) &= \max \left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l}) + \gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l}) + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \alpha[r_\mathtt{s} + \gamma v_*(\mathtt{h})] + (1-\alpha)[r_\mathtt{s} + \gamma v_*(\mathtt{l})], \\ 1[r_\mathtt{w} + \gamma v_*(\mathtt{h})] + 0[r_\mathtt{w} + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} r_\mathtt{s} + \gamma[\alpha v_*(\mathtt{h}) + (1-\alpha) v_*(\mathtt{l})], \\ r_\mathtt{w} + \gamma v_*(\mathtt{h}) \end{array} \right\}. \end{aligned}$$

    state가 high일때는 action이 search, wait이므로 두가지 linear function에서 max 값을 구한다.

$$v_*(\mathtt{l}) = \max \left\{ \begin{array}{l} \beta r_\mathtt{s} - 3(1-\beta) + \gamma[(1-\beta) v_*(\mathtt{h}) + \beta v_*(\mathtt{l})] \\ r_\mathtt{w} + \gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}$$

    state가 low일때는 action이 search, wait, recharge이므로 세가지 linear function에서 max 값을 구한다.

- Example 3.12: Solving the Gridworld

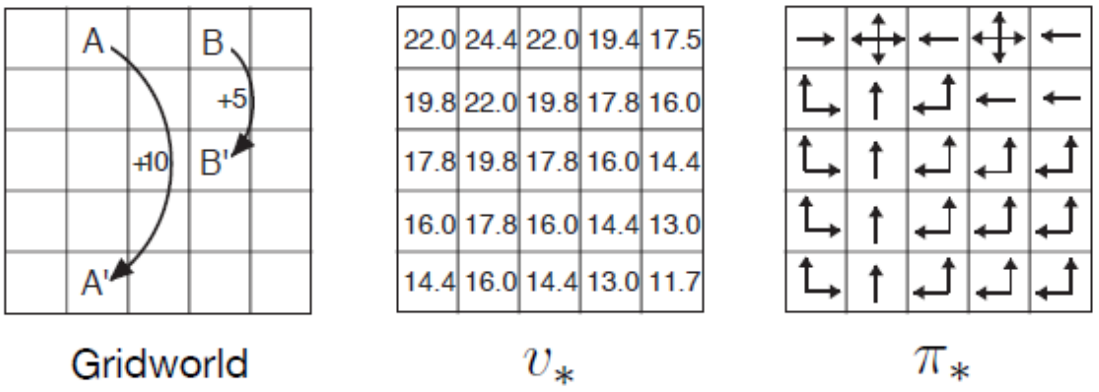


Figure 3.8: Optimal solutions to the gridworld example.

bellman optimal equation으로 grid world을 풀게되면 위와 같이 optimal value function과 optimal polices을 구할 수 있다. 하지만 실제의 경우에는 엄청나게 많은 경우에 대한 연산이 필요하게 되므로 완벽한 opimal을 구하기 어렵다. 결국, reinforcement-learning에서 말하는 opitmal solution은 approximate solution이라고 볼 수 있으며 이 책에서는 이러한 관점(대략적으로 ballman opimal equation을 푸는 것)에서 몇가지 방법을 배울것이다.

3.9 Optimality and Approximation

- agent는 optimal policy아래서 매우 잘 동장하지만 실제로 opitmal한 값을 구하려면 엄청난 연산비용이 들기 때문에 이러한 경우는 매우 드물다.
- 그리고 value function, police, models등을 저장할 엄청난 메모리 공간을 필요로 하기도 한다. 작은 task인 finite state set의 경우에는 전체의 state-action에 대한 경우를 array나 table에 저장할 수 있는데 이것을 tabular case라고한다. 그러나 이것 역시 실제의 경우는 tabular case에 해당하는 경우는 드물기 때문에 모든경우를 저장하는 것은 어렵다.
- 결국 reinforcement learning problem은 근사값을 구하는것이다.
- The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

3.10 Summary

- reinforcement-learning은 goal을 달성하기위해 agent가 environment와 여러 time steps에 걸쳐 어떻게 상호작용 해야하는지 학습시키는 것이다. 여기서 그 도구로서 사용되는 것이 action과 action을 선택할 기준이되게 하는 state다. 그리고 reward는 state에 대한 action의 evalution기능을 한다. policy는 agent가 특정 state에서 선택할 action을 고를 때에 대한 rule이다.
- return(G_t)은 현재 state s부터 terminal state까지를 보았을때 reward를 구하는 function이다. undiscounted formulation($\gamma = 1$)은 agent와 enviroment의 상호작용은 episodes로 나타낼 수 있다. 이 경우 끝이 있는 경우는 *episodic task* 라고 한다.
- discounted formulation(discounting rate를 사용하는)은 쪼갤 수 없는 지속적인 경우은 *continuing task* 에 적합하다.
- Markov property는 state signal이 과거에 대한 정보를 나타낼 수 있을때 성립한다.
- Markov property가 적용 될 수 있는 task의 경우 그것을 Markov decision process라고 부른다.
- finite MDP는 finite state와 action set에 대한 MDP를 말한다.
- value function은 각각의 state와 state-action, policy를 bellman-equation을 이용하여 구한다. 여기서 최대가 되는 value function을 optimal value function이라고 부르고 그때의 policy를 optimal policy라고 부른다. optimal policy는 여러 개가 될 수도 있지만 optimal value functions는 유일하다.
- 이상적인 것은 agent가 완벽하고 정확한 model을 갖는 것이지만 그것은 연산 비용이나 메모리 때문에 불가능한 경우가 많다. 완벽한 optimal solution을 찾는 것은 대단히 어렵지만 그것에 거의 가까운 값을 구하는 것이 reinforcement-learning이다.