ch3. Finite Markov Decision Processes

야아쓰. 드디어 챕터 뜨리.
이번 챕터에선 finite Markov decision processes의 formal problem을 소개해줄게.
앞으로 MDPs 라고 부를 건가봐.
근데 이게 책의 나머지에서 풀려고 하는 문제네.
이 problem은 evaluative feedback과 연관이 있어. bandit에서처럼.
그리고 associative aspect 도 있지. different actions을 고르는겨. different situations에서.
MDPs 는 sequential decision making의 classical formalization이야.

이게 뭐냐면 actions이 immediate rewards 뿐만 아니라 subsequent situations, or states 에도 영향을 미치고, future rewards에도 영향을 미치는 거.
그래서 MDPs 는 delayed reward랑 관련이 있고
그러니까 당연히 immediate and delayed reward 간의 tradeoff 가 필요하겠지.

bandit problem에서는 $q_*(a)$를 각 action a 마다 estimate 했지?
MDP에서는 $q_*(s,a)$ 를 estimate 해. each action a in each state s 마다.
또는 optimal action selections이 주어졌을 때 $v_*(s)$ of each state 를 estimate 해.
이런 state-dependent quantities는 개별 action selections이 장기적으로 좋은 결과를 얻을 수 있게 해 준다.

MDPs 는 강화학습 문제에서 수학적으로 이상적인 폼이다. 이론적으로 정확한 statements 에 도움이 되지.
problem의 수학적 구조의 key elements를 소개해 줄게.
returns, functions, Bellman equations 같은 것들.
finite MDPs 로 공식화할 수 있는 응용 분야들 많이 알려줄게.

다른 인공지능도 마찬가지지만, 실제 적용이랑 수학적으로 다루는 거랑은 tension이 있어.
그 tension도 소개해 줄게. trade-off와 challenges 도.
MDPs를 넘어가는 건 17챕터에서 다룬다잉?

3.1 The Agent-Environment Interface

MDPs 는 goal 을 achieve 하기 위한 interaction을 통해 학습하는 문제에 대한 straightforward한 framing이야.
learner 와 decision maker를 agent라고 부르지.
얘가 interact 하는, agent의 outside를 구성하는 모든 것을 environment라고 해.

얘들은 interact continually야. agent가 selecting actions 하고 environment는 actions에 responding. 그리고 새로운 situations를 agent한테 presenting 하지.
environment는 rewards도 생기게 하고.
rewards가 뭐냐면, numerical values인데 agent가 시간이 지남에 따라 고르는 actions을 통해 maximize 하고싶어하는 거야.
(fig3.1)을 한 번 볼까?

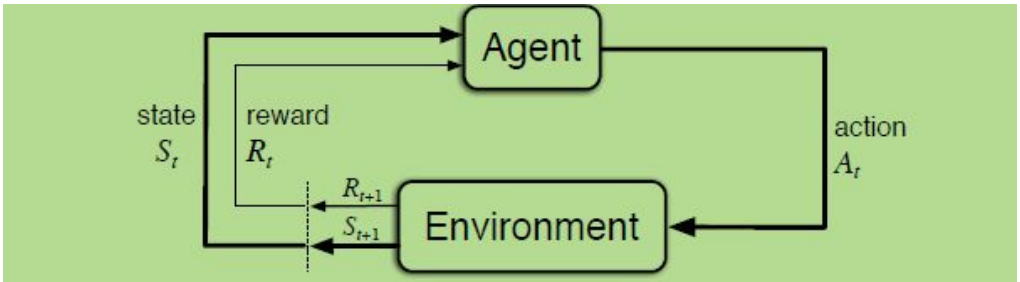


Figure 3.1: The agent–environment interaction in a Markov decision process.

더 구체적으로, agent와 environment는 매 discrete time steps sequence 마다 interact해.
at each of a sequence of discrete time steps.
그러니까 t = 0,1,2,3,... 마다.
매 time step t에서 agent는 어떤 representation을 받아. 어떤 representation?
environment's state의 representation.

$S_t \in \mathcal{S}$

이렇게 표현을 하네?
이 basis에서 select an action을 하고 그걸

$A_t \in \mathcal{A}(s)$

이렇게 표현을 하네?

한 스텝 더 가면, action의 결과로 agent는 numerical reward를 받아.

$R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

바로 이것.

그리고 새로운 상태로 가는 거지.

S_{t+1}

요로케.
MDP와 agent는 함께 이런 sequence 또는 trajectory를 일으키지.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

finite MDP에서는, states, actions, rewards 요 세트가 전부 finite number of elements를 가져.

$(\mathcal{S},\ \mathcal{A},\ \text{and}\ \mathcal{R})$

바로 요것.

이 case에서는 random variables R_t, S_t 가 well defined discrete probability distributions을 따라. 이 distribution은 오직 preceding state와 action에만 dependent 하지.
그러니까, 이 random variables의 특정 values

$s' \in \mathcal{S}\ \text{and}\ r \in \mathcal{R}$

에 대해, time t 에 저 value들이 일어날 확률이 있다는겨.

preceding state와 action의 특정 values가 주어졌을 때.

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

$$\text{for all } s', s \in \mathcal{S},\ r \in \mathcal{R},\ \text{and } a \in \mathcal{A}(s)$$

바로 이 식인겨.
등호 위의 점은 함수 p에 대한 definition이다잉?

$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ 는 ordinary deterministic function이오. four arguments가 있는.
| 는 conditional probability인데, 여기선 그냥 p가 s와 a의 probability distribution라는 걸 나타낸다? 헐 그런건가. conditional 이 아니었나.

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

이렇다네.

확률 p는 completely characterize한다네. 뭘? dynamics of a finite MDP를.
거기서부터, 우리는 environment에 대해 알고 싶은 걸 다 계산할 수 있대.
state-transition probabilities같은 것들.

$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$

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

요렇게 표현되는 거지. 약간의 abuse of notation이 있댜.

또 expected rewards도 계산할 수 있어. state-action pairs의 expected rewards.

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a)$$

요렇게.

state-action-next_state 조합도 가능.

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

요렇게.

이 책에선 보통 four argument짜리를 쓸 거야. 근데 나머지들도 종종 쓰면 편리함.

MDP framework는 abstract하고 flexible해. 그리고 많은 서로 다른 문제에 다양한 방식으로 적용 가능. 예를 들어, time step 이 real time의 fixed interval일 필요가 없는 거지. 아무 연속적인 stages일 수 있어. decision making and acting stage.
actions은 low-level controls일 수 있어. 로봇 팔의 voltages 같이.
아니면 high-level decisions. 점심을 먹을지 또는 대학원을 갈지 같은?

비슷하게, states도 아주 다양한 form을 가질 수 있지.
low-level로는 direct sensor readings.
high-level, abstract 하게는 방에 있는 물체의 symbolic descriptions.
past sensations의 memory에 근거하거나 아예 mental 또는 subjective한 걸로 state를 구성할 수도 있어. 예를 들어, agent가 state에 있는데 object가 있는지 확실하지 않은 state인 거야.
아니면 clearly defined sense로 놀랄 수도 있지.

또 비슷하게, actions도 totally mental or computational 할 수 있어.
예를 들어, agent가 뭐에 대해 생각할지 control 할 수도 있고, attention에 focus할 수도 있다.

일반적으로,
actions은 어떻게 정할지 학습하고 싶은 어떤 decisions이든 가능.
states 는 decisions을 만드는 데 도움이 되는 어떤 것이든 가능.
actions can be any decisions we want to learn how to make, and the states can be anything we can know that might be useful in making them.

특히, agent와 environment간의 boundary는 일반적으로 로봇이나 동물의 body의 physical boundary와는 달라.
보통, boundary는 agent에 가깝게 그려져.
예를 들어, motors와 robot의 mechanical linkages, 그리고 sensing hardware는 보통 environment의 parts로 간주해야 해. agent의 parts 가 아니라.
비슷하게, MDP framework를 사람, 동물에 적용한다면, 근육, 해골, sensory organs는 environment part로 간주돼야지.
Rewards도 마찬가지로, physical bodies of natural and artificial learning systems 안에서 계산되지만, agent의 external이야.

일반적인 rule은, agent가 맘대로 바꿀 수 없는 모든 것은 agent의 outside로 보고, 따라서 environment의 aprt라는 거지.
environment의 모든 것을 agent가 알고 있다고 보진 않아.
예를 들어, agent는 종종 reward가 어떻게 계산되는지 꽤 많이 알고 있을 때가 있어. actions, states로 이루어진 함수로 계산되는 rewards 말이야. 하지만 우린 항상 reward computation은 agent의 external이라고 간주해야해. 왜냐면 rewards는 agent를 향하는 task를 정의하고, 따라서 agent가 임의로 바꿀 수 없어야하거든.
사실, 경우에 따라서 agent가 environment가 어떻게 돌아가는지 모든 걸 알 때가 있어. 근데 그래도 어려운 강화학습 문제가 있지. 마치 Rubik's cube의 작동 방식을 정확히 알지만 풀 수 없는 것처럼.
agent-environment boundary는 한계를 나타내. 어떤 한계? agent's absolute control의 한계. knowledge의 한계가 아니라.

agent-environment boundary는 서로 다른 목적으로 서로 다른 장소에 놓을 수 있어.

복잡한 robot에선, 많은 agents들이 한 번에 작동해. 각자의 boundary를 가지고.
예를 들어, 한 agent가 states의 부분을 구성하는 high-level decisions making을 할 수 있지.
근데 그 states는 high-level dicision을 구현하는 lower-level agent가 만들어내는 거지.
원문을 보면서 읽으세유. 번역이 잘 되고 있는지 모르겄습니다.
실제에선, agent-environment boundary가 정해지는 때가 언제냐면, particular states, actions, rewards 를 선택했을 때야. 그래서 specific decision making task of interest를 특정했을 때지.

MDP framework는 매우 추상적이야. goal-directed learning from interaction 문제에서.
sensory, memory, control apparatus의 details 이 뭐든 간에,
objective one이 달성하려 노력하는 게 뭐든 간에,
learning goal-directed behavior 문제를 agent와 environment 사이에서 앞뒤로 전달되는 3 signals로 줄일 수 있다고 하거든.
actions: agent가 한 choices를 나타내는 signal
states: choice가 만들어진 basis를 나타내는 signal
rewards: agent의 goal을 define하는 signal

이 framework는 모든 decision-learning problems을 represent 하는 데 충분하진 않겠지만, widely useful and applicable함이 증명됨.

오브코오스, 특정 states, actions는 task마다 크게 다르고, 그들이 어떻게 represent 되느냐에 따라 performance에 영향이 커. 강화학습에선, 다른 learning과 마찬가지로, 고런 representational choices는 과학이 아니라 아트임.
이 책에서 몇 가지 advice와 examples을 보여줄게. 어떤? representing states and actions 하는 좋은 방법들.
하지만 primary focus는 representations이 정해졌을 때 어떻게 behave할지 배우는 general principles야.

(Example3.1) Bioreactor
강화학습을 bioreactor에 적용한다고 치자.
moment-by-moment temperature와 stirring rates를 determine하는거지.

actions은 target temperature, stirring rates가 될 거야.
걔네들은 lower-level control systems에 전달됨.
그 systems은 attain 하기 위해 heating elements와 motors를 activate 하는 애들.

states는 thermocouple과 sensory readings가 될 거고, filtered, delayed 될 거야 아마.
그리고 vat 안의 ingredients와 target chemical을 나타내는 symbolic inputs도 포함.

rewards는 useful chemical이 생성되는 rate의 moment-by-moment measure야.

각 state는 sensor readings와 symbolic inputs의 list, vector이고,
각 action은 target temperature, stirring rate 의 vector야.

요런 구조가 아주 전형적이라는군.
rewards는 항상 single numbers래.

(Example3.2) Pick-and-Place Robot
이번엔 로봇팔 동작 control 하는거야. 반복적으로 pick-and-place 하는겨.
빠르고 부드러운 움직임을 학습하고 싶으면, learning agent 가 motors를 직접적으로 control 할 수 있어야 할 거고, 현재 mechanical linkages의 positions과 velocities 정보를 low-latency로 받을 수 있어야겠지.

actions은 voltages야. 각 motor와 joint에 적용된 voltages.
states는 joint angles, velocities. latest readings. 가장 최근 수치.
reward는 각 object마다 pick, place가 성공하면 +1.

smooth한 움직임을 encourage하기 위해, 매 time step마다 small, negative reward를 줄 수 있어. 순간순간 이상한 움직임의 함수로.

(Exercise 3.1)
MDP framework에 맞는 example tasks를 만들어볼래? 세 개만?
states, actions, rewards를 다 identify 하고.
세 case 다 최대한 다르게 해 봐.
framework는 abstract하고 flexible하며 다양한 방식으로 적용 가능.
그러니 한계를 늘려보렴?

(Exercise 3.2)
MDP framework가 모든 goal-directed learning tasks에 적합할까?
어떤 예외가 있을까?

(Exercise 3.3)
문제가 많아. driving 문제를 보자.
actions을 define 할 수 있겠지? accelerator, steering wheel, brake 가 떠오르지 않냐.
그러니까 너의 몸이 기계와 만나는 곳들.
좀 더 참신하게 갈 수도 있겠다. 타이어 고무가 바닥과 만나는 곳.
actions이 tire torques가 되는거지.
아니면 네 뇌가 네 몸을 만나는 곳이라면, actions은 팔을 움직이는 근육의 경련이 되겠지.
엄청 high-level로 가면, actions이 어디로 갈지 정하는 선택이 될 수도 있고.
뭣이 right level, right place일까? agent와 environment의 경계를 긋기에.
어떤 기준으로 뭐가 더 좋다고 해야하지?
fundamental reason이 있니? 아님 free choice?

(Exercise 3.4)
current state St 에서 actions이 stochastic policy π에 따라 선택된다고 해 봐.
그러면 R_{t+1} 의 기대값은 어떻게 되니?
π와

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

여기에서의 argument 4개짜리 함수 p 관점에서.

(Exercise 3.5)

아래 example 3.3같은 table을 줘 봐. 위에 함수 $p(s', r|s, a)$ 의 경우로 만들어서.

s, a, s', r, $p(s', r|s, a)$ columns 이 있어야하고, row로는 $p(s', r|s, a)$>0인 경우의 every 4-tuple이 있어야 해.

Example 3.3 Recycling Robot
office에서 빈 깡통 모으는 mobile robot이 있어. 인트로가 아주 흥미롭군.
can을 detect하는 sensor가 달려있고 arm & gripper가 있어서 주워서 bin 에 넣을 수 있다.
rechargeable battery 쓰고요.
control system은 sensory information을 받을 수 있고, navigating 할 수 있고, arm&gripper를 control 할 수 있는 components들을 갖고 있지.
cans을 어떻게 찾을지 결정하는 High-level decisions 은 강화학습하는 agent가 현재 battery charge level로 결정해.
간단하게 하기 위해, charge level이 두 개만 있다 치자.
state set S = {high, low}야.
각 state에서 agent는 결정하지. 뭘?
(1) 일정 시간동안 actively search 할지,
(2) stationary 하게 있으면서 누가 can을 가져다줄 때까지 기다릴지,
(3) home base로 가서 recharge 할지.

energy level이 high면, recharge할 필요가 없지? 그래서 이 state에 대한 actions set은 뺀다.
action set은 A(high)={search, wait}, A(low)={search, wait, recharge}.

rewards는 most of the time, 0이야.
robot 이 empty can을 확보하면 positive가 돼.
근데 battery 꺼지면 크게 negative.
can을 찾기 위해선 actively search 해야하지만, battery 다 닳을거야. waiting 하면 안 닳지.
robot이 searching 할 때마다 battery가 남아있을 확률이 팍팍 줄어들겠지.
그러면 robot은 shut down해야하고 구조될 때까지 기다려야해. low reward 받겠지.

energy level이 high면 active search의 period는 항상 complete 될 거야. battery 방전 걱정 없이.
high energy level로 시작한 period of searching은, 끝나고 나서 α 확률로 energy level을 high로, 1-α 확률로 low로 만들어.

반면 low level일 때 period of searching은 β 확률로 low를 남기고, 1-β 확률로 deplete.
deplete the battery 하면 로봇이 rescue돼야겠지. 그리고 battery는 high로 recharge 될 거야.

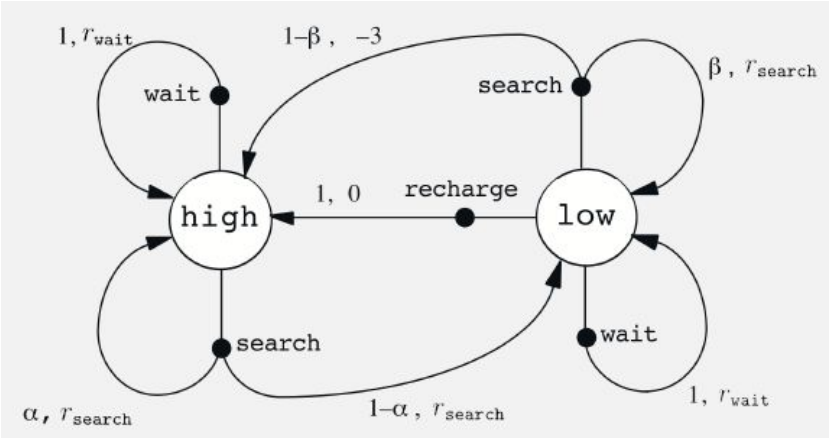
로봇이 주운 can 각각은 unit reward로 count 될 거고,
로봇이 구조돼야할 상황이 오면 -3 reward.

r_{search} 와 r_{wait} 는, 우선 $r_{search} > r_{wait}$ 이고,
각각 robot이 searching할 때, waiting 할 때 주울 cans의 expected number(그래서 expected reward).

마지막으로 recharge를 위해 집으로 돌아갈 땐 can 안 주움.
battery 가 depleted 됐을 때도 can 안 주움.

이 system은 그러면 finite MDP 이다. 그럼 transition probabilities, expected rewards를 dynamic하게 적을 수 있겠지?

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



table에 row를 보면, current state s, action a, next state s' 의 모든 조합이 다 있어.

finite MDP의 dynamics를 summarize하는 또 다른 좋은 방법은 transition graph야.

두 가지 nodes가 있는 거 보이니. state nodes 와 action nodes야.
모든 가능한 state에 대해 state node가 있어. high, low 보이니?
각 state-action pair에 대해 action node가 있지. 작은 까만 동그라미야. action 이름 붙은 거 보이니? state node에 줄이 연결돼있지.

state s 에서 시작해 action a를 하면 state node s에서 action node (s,a) 로 라인을 따라 움직여.
그러면 environment는 next state's node로 옮겨가. action node (s,a) 화살표를 따라.
모든 화살표는 triple (s, s', a)에 대응하고,
화살표엔 transition probability p(s' | s,a) 를 labeling.
expected reward for that transition r(s,a,s')도 labeling.
action node를 나가는 transition probabilities의 합은 항상 1이야.
이제 exercise 3.5 해야지?후후

3.2 Goals and Rewards

강화학습에선 agent의 purpose나 goal이 reward 로 공식화돼. reward는 environment로부터 agent로 전달되지.
매 time step마다 reward는 걍 실수야.
agent의 goal은 받는 total amount of reward를 maximize하는 것.
immediate reward를 maximize하는 게 아냐. cumulative reward 를 maximize 하는 것.
한 마디로 정리하면,
reward hypothesis:
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).
굳이 이럴 필요가 있었을까.

goal을 이렇게 reward signal 로 formalize한다는 게 강화학습의 특징 중 하나.
이렇게 하는 게 limiting 처럼 보일 수 있는데, 실전에선 매우 유용하고 flexible함이 증명됨.
예제를 볼까?
robot이 걷는 걸 배우게 만들기 위해, 연구원들이 매 time step 마다 reward를 주지. robot의 forward motion에 비례해서.
robot이 미로를 탈출하는 법을 배우게 할 땐, 탈출 전 모든 time step에서 reward는 -1이야.
이러면 agent는 최대한 빨리 탈출하기 위해 용 쓰지. 좀 가혹한데.
can 좁기 시킬 때는, 대부분 reward 0, can 제대로 모았을 때 +1.
뭔 일이 발생하면 negative rewards를 줄 수도 있지.
chess 배울 땐 winning에 +1, losing 에 -1, drawing과 nonterminal positions 에 0.

뭔 소린지 알겠지?
agent는 reward를 maximize하도록 학습한다.
얘가 우릴 위해 뭔가 해주길 바란다면, reward를 그쪽 방향으로 줘서 우리 목표를 달성하도록 만들면 되겠지.
그래서 reward를 우리가 달성하고자 하는 걸 가리키게 하는 게 매우 중요.

reward는 사전 지식을 agent에게 주는 게 아냐. 어떤 사전 지식? 우리가 원하는 것을 달성하는 방법에 대한 사전 지식.
예를 들어, chess playing agent는 winning에만 보상을 받지. 상대 말을 얼마나 뺏는지, 중앙을 얼마나 장악하고 있는지에 대한 subgoal에 대해선 보상을 받지 않아.
이런 subgoal들에 보상을 주면 실제 goal인 winning이 아니라 subgoal을 달성하기 위한 길을 찾을거야. winning 보다 상대 말을 더 많이 뺏는 법을 익히는거지.

reward signal은 robot과 communicating하는 길이야.
무엇을 달성하게 만들지를.
어떻게 달성하는지가 아니라.
어떻게는 알아서 찾는다고 볼 수도 있겠네.

3.3 Returns and Episodes

여태 학습의 objective에 대해 논의했는데 넘나 informally했지?
agent의 goal은 long run 에서 받는 cumulative reward를 maximize 하는거라 했다.
이걸 어떻게 formally 정의할까?

t 시점 이후로 받은 reward를 R_{t+1}, R_{t+2}, R_{t+3},... 이렇게 표현하면, 우리가 maximize하고자 하는 건 이 sequence의 어떤 부분일까?
일반적으로, expected return을 maximize하길 원한다.
return 은 G_t 로 쓰고, reward sequence의 특정한 함수로 정의해.
가장 간단한 case로, return은 rewards의 sum이지.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

T가 final time step이고.

요런 approach는 실제 적용에 나쁘지 않아. final time step이 있는 경우에.
그러니까, agent-environment interaction이 subsequences로 자연히 쪼개지는 경우.
이걸 episodes라고 불러.
game이나 미로 탐색, 또는 어떤 repeated interaction.

각 episode는 terminal state에서 끝나.
starting state로 reset되거나, starting states의 standard distribution의 sample로 reset 돼.
episodes가 winning, losing 등 다양한 방식으로 끝나도 next episode가 독립적으로 시작돼.
이전 episode가 어떻게 끝났든지 상관없이.

그러니 모든 episodes는 different outcome에 대해 different rewards의 동일한 terminal state에서 끝난다고 볼 수 있어.
이런 episodes에서의 task는 episodic tasks 라고 불러.
episodic tasks에선, 모든 nonterminal states set 을 구분할 필요가 있지.
S: set of all nonterminal states.
S^+: set of all states plus the terminal state.

T: time of termination. episode마다 normally 달라지는 random variable.

반면, agent-environment interaction이 identifiable episodes로 자연스럽게 쪼개지 않으면서 무한하게 계속 진행되는 경우도 많아.
예를 들어, on-going process-control을 formulate 한다거나, 수명이 긴 로봇에 적용한다거나.
이런 걸 continuing tasks 라고 불러.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

이런 return 공식은 continuing tasks 에는 문제가 있지.
final step T가 무한대니까.
maximize하고 싶은 return 은 무한대로 가겠지.
매 time step +1 받는다고 생각해봐.
그래서 이 책에서는 return의 정의로 개념적으론 좀 복잡하지만 수학적으로 훨씬 간단한 걸 써보도록 하자.

여기서 discounting 이라는 개념이 좀 필요함.
이 approach에선 agent 가 시간에 따라 discount 된 rewards의 합을 maximize하는 action을 선택하려고 해.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3.8)$$

앞으로 받을 expected discounted return을 maximize하는 action A_t를 선택.
γ 는 [0,1], discount rate라고 불러.

discount rate는 미래 rewards의 현재 가치를 결정한다.
k time step 미래에 받을 reward는 지금 받았으면 γ^{k-1} 곱한 것만큼의 가치를 가진다.
$\gamma < 1$이면 (3.8) 의 infinite sum은 reward sequence $\{R_k\}$ bounded돼 있는 한 finite value가 될 거다.
$\gamma = 0$ 이면, agent는 myopic, 근시가 될 거야. immediate rewards만 maximizing 하겠지.
이 경우 objective는 R_{t+1} 만 maximize 하는 A_t를 선택하는 방법을 배우는거야.

혹시라도 만약 각 agent의 actions가 오직 immediate reward에만 영향을 미친다면 agent는 각 immediate reward를 개별적으로 maximizing 해서 (3.8)을 maximizing 할 거야.
하지만 일반적으로, immediate reward를 maximize 하기 위해 acting 하면 future rewards에 대한 access가 줄어 return 이 줄어들어.
γ이 1에 가까울수록, return objective는 future rewards는 더 강하게 고려한다는 뜻이고 agent가 더 미래지향적이 되겠지.

연속적인 time steps에서의 returns 은 서로 연결돼있어. 이거 중요.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

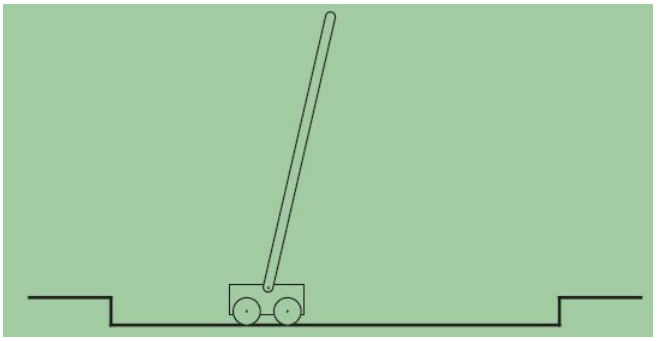
ch2의 increment 생각난다잉.
이건 모든 time steps $t < T$ 에 대해 성립.
$G_T = 0$ 으로 정의했을 때 t+1에서 termination이 발생하더라도 성립.

(3.8)의 return 이 infinite number of terms의 합이라해도, reward가 nonzero 이고 constant 이면 finite 이다. $\gamma < 1$ 인 경우에 말이지.
예를 들어, reward가 constant +1 이면, return은

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

요로코롬.

(example 3.4) Pole-Balancing



그림을 보면, 카트에 달린 막대가 있쥬? 카트는 트랙따라 움직이고용.
카트에 힘을 줘서 저 막대가 쓰러지지 않게 하는 게 이 task의 objective유.
막대가 수직에서 특정 각도를 넘어가거나, 카트가 트랙을 벗어나면 failure.
실패할 때마다 vertical로 리셋.
episodic task로 취급할 수도 있겠지. natural episodes가 pole을 balance하는 반복적인 시도라고 하고.
reward는 failure가 발생하지 않은 모든 time step에서 +1이라 할까?
매 time, return은 실패하기까지의 number of steps 가 될 거야.
이 경우에, 영원히 balancing을 성공하면 return은 infinity가 돼.

continuing task로 취급할 수도 있어. discountin을 이용하는.
이 경우에, reward는, failure일 때 -1, 다른 때는 다 0일 거야.
return at each time 은 $-\gamma^K$ 와 관련이 있겠지. K는 failure 전의 number of time steps야.

어떤 case든 pole balancing을 최대한 유지해서 return이 maximize 돼.

(Exercise 3.6)
3.1에서의 equation들은 continuing case 일 때 적용되는 것들이었지.
episodic tasks에 적용하려면 약간 수정돼야해.

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \,|\, s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

(3.3) 식인데 이거 한 번 바꿔볼래?

(Exercise 3.7)
pole-balancing을 episodic task로 생각해보자.
근데 discounting도 쓰고.
대신 failure 일 때만 -1이고 나머지는 rewards가 0.
그럼 매 time마다 return 이 어떻게 될까?
return이 discounted, continuing formulation이랑은 어떻게 다를까?

(Exercise 3.8)
미로찾기 로봇을 디자인한다고 해 보자.
+1 reward를 탈출할 때 주고 다른 모든 time 에는 reward가 0으로 했다 쳐.
task는 자연스럽게 episodes로 쪼개지겠지. 연속적으로 미로찾는 episodes.
그래서 episodic task로 취급하기로 했어.
goal은 expected total reward

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

를 maximize 하는 거.
learning agent 가 한동안 running 하고, 탈출하는 데 전혀 improvement가 없었어.
뭣이 잘못됐는가? agent가 achieve하길 바라는 게 뭔지 잘 communicate 했는가?

(임시답안)

(Exercise 3.9)
γ = 0.5 야. R_1 = -1, R_2 = 2, R_3 = 6, R_4 = 3, R_5 = 2, T = 5 야.
G_0, G_1, …, G_5 는 어떻게 될까?
Hint: Work backwards.

(Exercise 3.10)
γ = 0.9, R1 = 2, 그 뒤의 reward sequence는 7로 무한히 이어져.
G_1, G_0는 뭘까?

(Exercise 3.11)

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

이거 증명.