Homework 6 - 7

Menu

HW#6 Problem 1

(Exercise 2.2; 20 points)

(a) Use the languages $A = \{a^n b^n c^m \mid m, n \geq 0\}$ and $B = \{a^m b^n c^n \mid m, n \geq 0\}$, together with the fact that $\{a^n b^n c^n \mid m, n \geq 0\}$ is not context free, to show that the class of context-free languages is not closed under intersection.

(b) Use the preceding part and DeMorgan's law to show that the class of context-free languages is not closed under complementation.

HW#6 Problem 1 (a)

Transform languages A and B into the new forms:
$A = \{a^i b^j c^k \mid (i = j) \wedge (i, j, k \geq 0)\}$, and
$B = \{a^i b^j c^k \mid (j = k) \wedge (i, j, k \geq 0)\}$

The intersection of A and B
$= \{a^i b^j c^k \mid (i = j) \wedge (j = k) \wedge (i, j, k \geq 0)\}$, which is equal to
$\{a^n b^n c^n \mid m, n \geq 0\}$

We've known that A and B are context-free languages, but the intersection of A and $B = \{a^n b^n c^n \mid m, n \geq 0\}$ is not context free, so the class of context-free languages is not closed under intersection.

HW#6 Problem 1 (b)

DeMorgan's law: $A \cap B = \overline{\overline{A} \cup \overline{B}}$

We've known that the class of context-free languages is closed under union. Now suppose that the class of context-free languages is closed under complementation and A and B are two context-free languages:

A and B are context free.
$\Rightarrow$ $\overline{A}$ and $\overline{B}$ are context free.
$\Rightarrow$ $\overline{A} \cup \overline{B}$ is context free.
$\Rightarrow$ $\overline{\overline{A} \cup \overline{B}}$ is context free.
$\Rightarrow$ $A \cap B$ is context free.
$\Rightarrow$ **false**

HW#6 Problem 1 (b)

We've known that the class of context-free languages is not closed under intersection in problem 1 (a), contradiction.

So the class of context-free languages is not closed under complementation.

HW#6 Problem 2

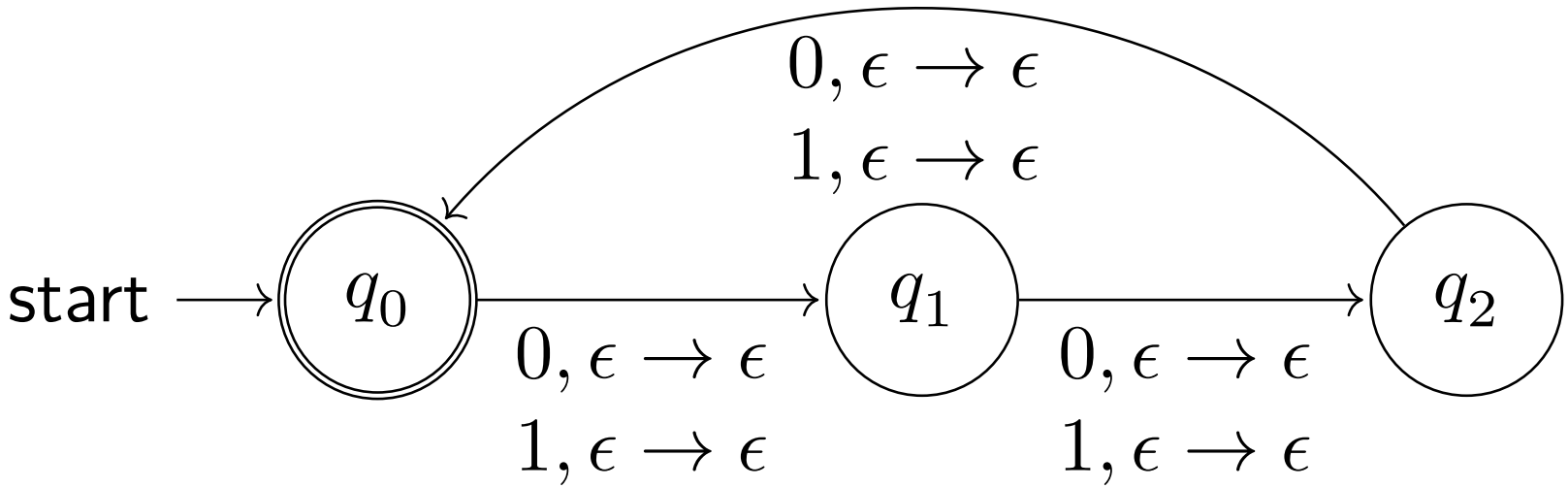
(Exercise 2.5; 20 points) Give informal descriptions and state diagrams of pushdown automata for the following languages. In all parts the alphabet Σ is $\{0, 1\}$.

(a) $\{w \mid \text{the length of } w \text{ is a multiple of } 3\}$

(b) $\{w \mid w \text{ is a palindrome, that is, } w = w^R\}$

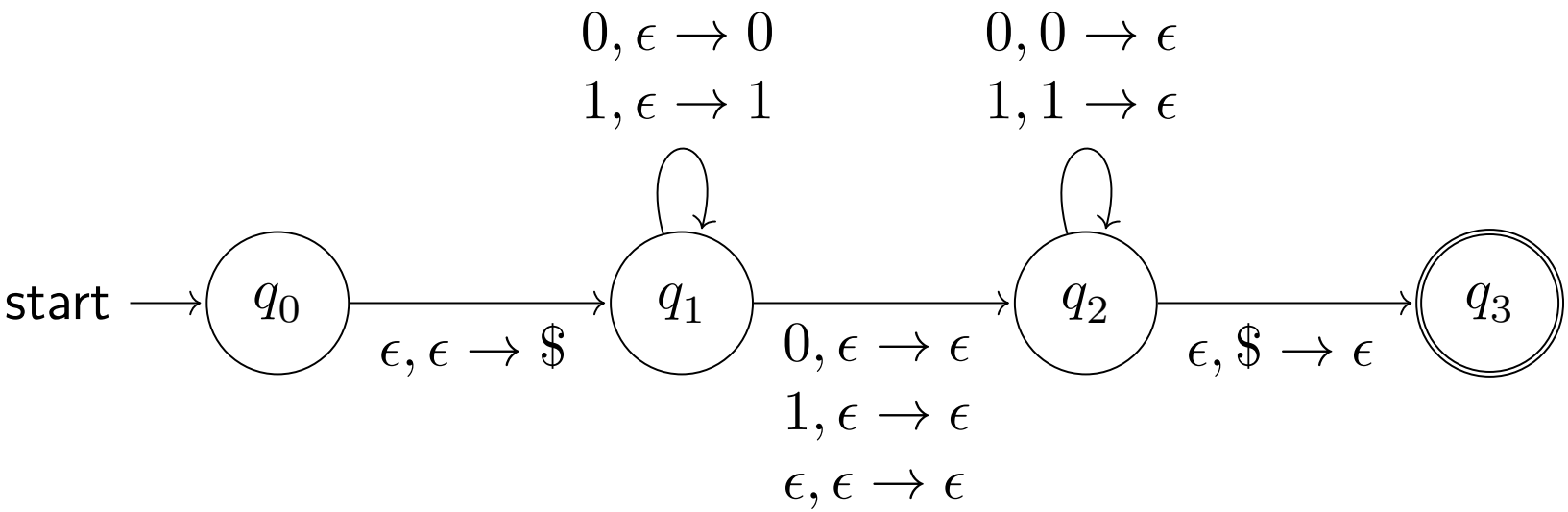
HW#6 Problem 2 (a)

$\{w \mid$ the length of w is multiple of 3$\}$



HW#6 Problem 2 (b)

$\{w \mid w \text{ is a palindrome, that is, } w = w^R\}$

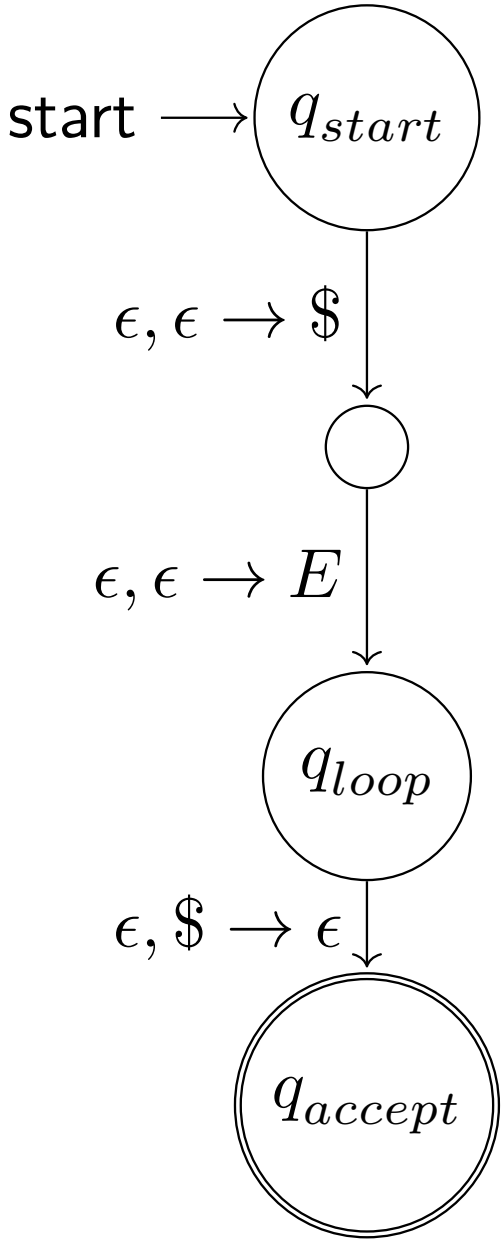


HW#6 Problem 3

(Exercise 2.12; 10 points) Convert the following CFG to an equivalent PDA, using the procedure given in Theorem 2.20.

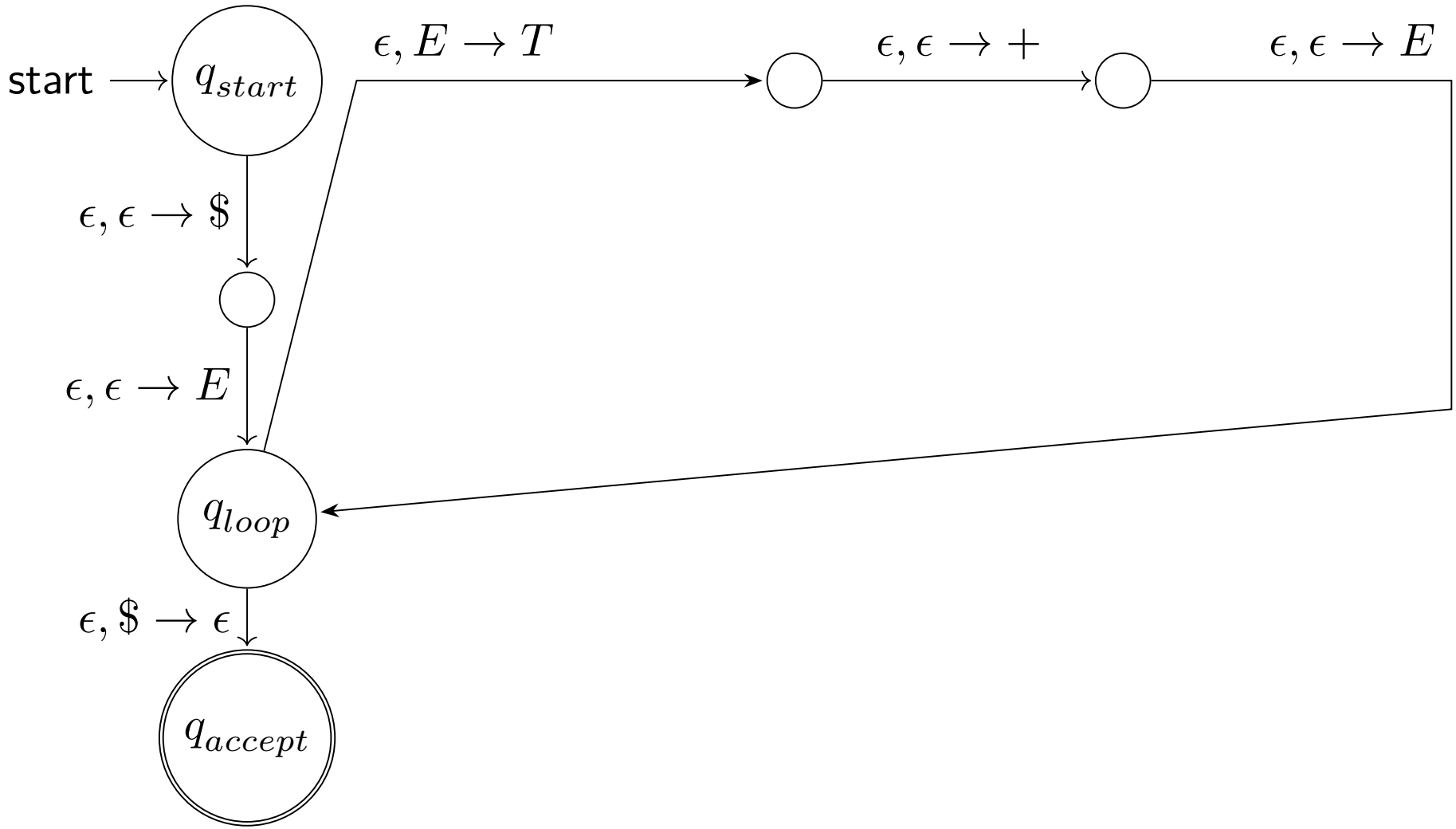
$$\begin{aligned} E &\to E + T \mid T \\ T &\to T \times F \mid F \\ F &\to (\ E \ ) \mid a \end{aligned}$$

HW#6 Problem 3



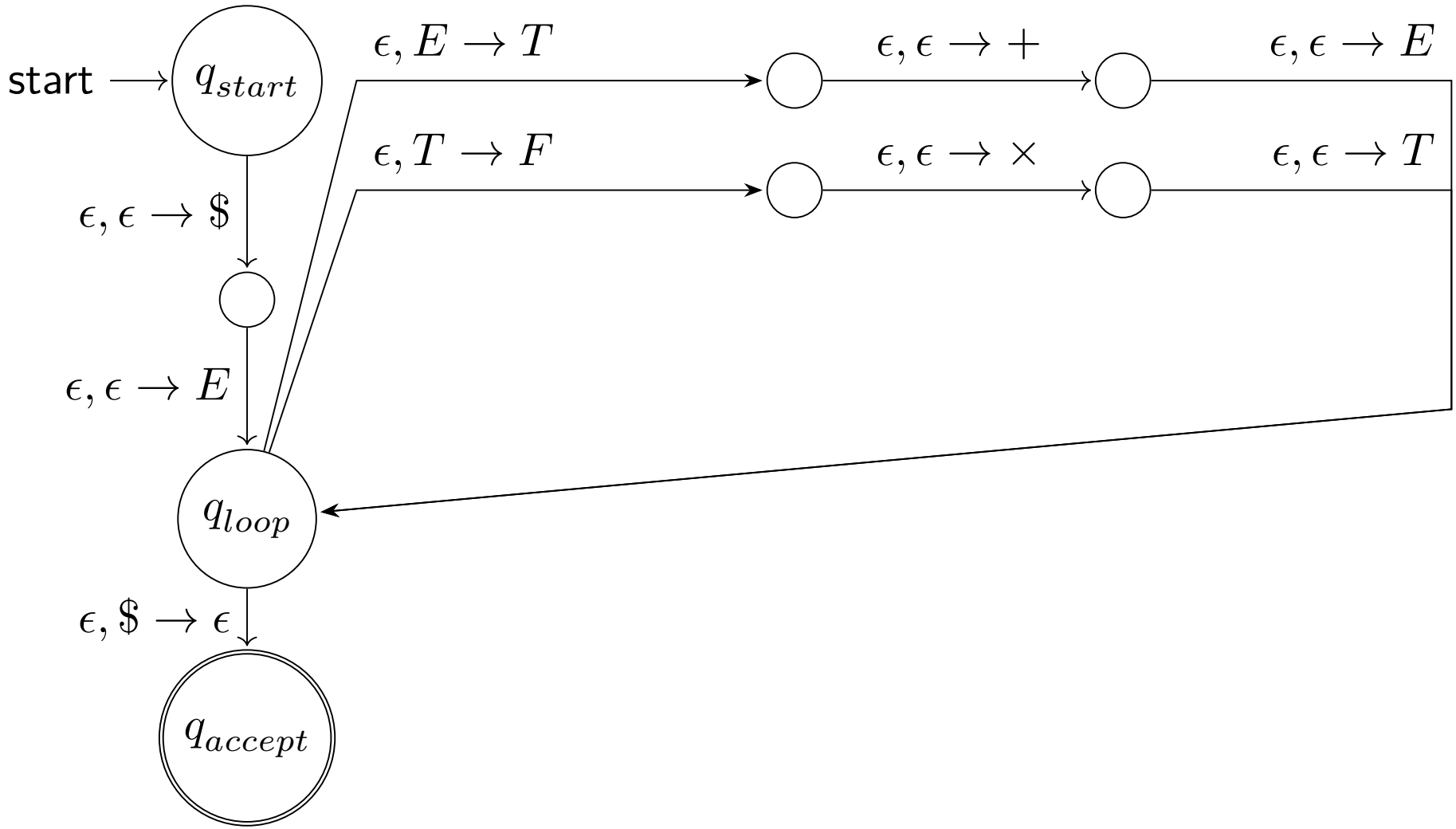
HW#6 Problem 3

$E \to E + T$



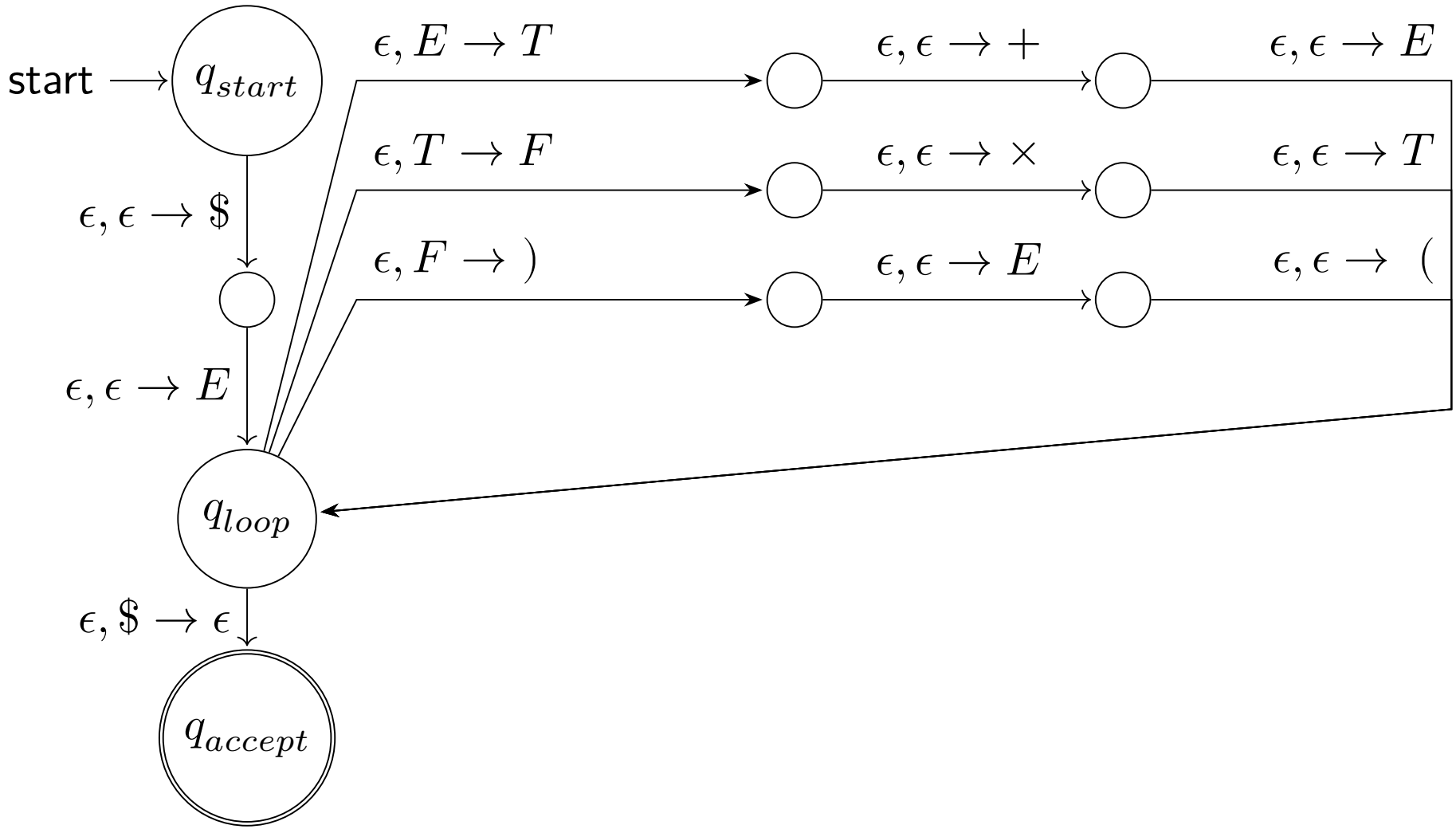
HW#6 Problem 3

$T \to T \times F$



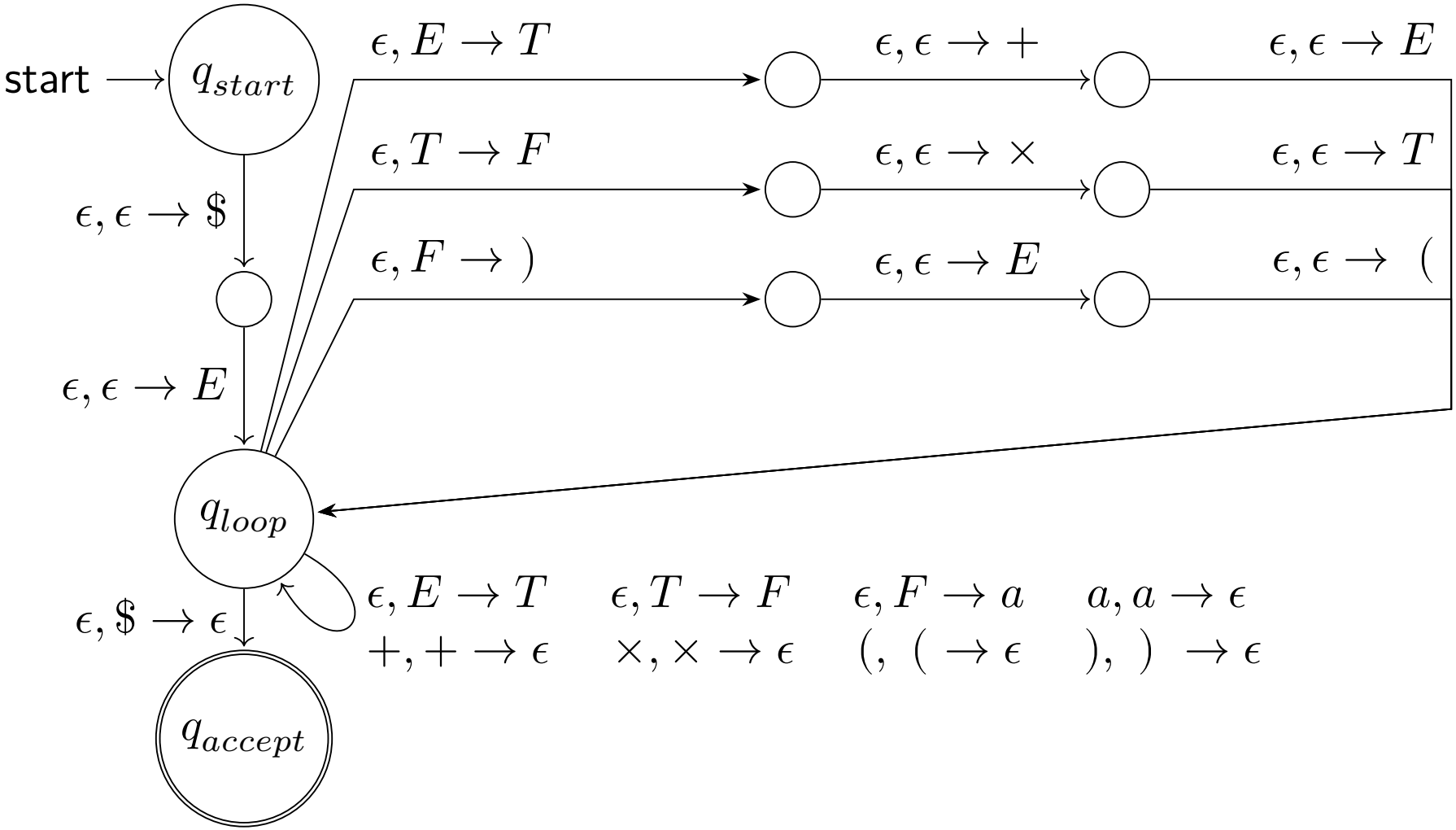
HW#6 Problem 3

$F \rightarrow \ (\ E \ )$



HW#6 Problem 3

Remaining grammar



HW#6 Problem 4

(Problem 2.39; 20 points) Let $G = (V, \Sigma, R, \langle \text{STMT} \rangle)$ be the following grammar.

$$
\begin{aligned}
\langle \text{STMT} \rangle &\rightarrow \langle \text{ASSIGN} \rangle \mid \langle \text{IF-THEN} \rangle \mid \langle \text{IF-THEN-ELSE} \rangle \\
\langle \text{IF-THEN} \rangle &\rightarrow \texttt{if condition then } \langle \text{STMT} \rangle \\
\langle \text{IF-THEN-ELSE} \rangle &\rightarrow \texttt{if condition then } \langle \text{STMT} \rangle \texttt{ else } \langle \text{STMT} \rangle \\
\langle \text{ASSIG} \rangle &\rightarrow \texttt{a := 1}
\end{aligned}
$$

$$\Sigma = \{\texttt{if}, \texttt{condition}, \texttt{then}, \texttt{else}, \texttt{a := 1}\}$$

$$V = \{\langle \text{STMT} \rangle, \langle \text{IF-THEN} \rangle, \langle \text{IF-THEN-ELSE} \rangle, \langle \text{ASSIG} \rangle\}$$

G is a *natural-looking* grammar for a fragment of a programming language, but G is ambiguous.

(a) Show that G is ambiguous.

(b) Give a new unambiguous grammar for the same language.

HW#6 Problem 4 (a)

Counterexample:
`if condition then if condition then a:=1 else a:=1`

There are two ways to obtain this language:

1.
⟨STMT⟩
⇒ ⟨IF-THEN⟩
⇒ `if condition then` ⟨STMT⟩
⇒ `if condition then` ⟨IF-THEN-ELSE⟩
⇒ `if condition then if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
⇒ `if condition then if condition then a:=1 else a:=1`

HW#6 Problem 4 (a)

2.
⟨STMT⟩
⇒ ⟨IF-THEN-ELSE⟩
⇒ `if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
⇒ `if condition then` ⟨IF-THEN⟩ `else` ⟨STMT⟩
⇒ `if condition then if condition then` ⟨STMT⟩ `else`
⟨STMT⟩
⇒ `if condition then if condition then a:=1 else a:=1`

So G is ambiguous.

HW#6 Problem 4 (b)

⟨STMT⟩ → ⟨ASSIGN⟩ | ⟨IF-THEN⟩ | ⟨IF-THEN-ELSE⟩
⟨IF-THEN⟩ → `if condition then` ⟨STMT⟩
⟨IF-THEN-ELSE⟩ → `if condition then` ⟨STMT⟩ `else` ⟨STMT⟩
⟨ASSIGN⟩ → `a:=1`

The problem of the original grammar G is that when ⟨IF-THEN-ELSE⟩ appears, we expect that the `if` and `else` in it should be matched, but the ⟨STMT⟩ in front of the `else` may have a unmatched `if` which may wrongly match the `else`.

To solve the problem, we need to guarantee that all `if` and `else` between the `if` and `else` in ⟨IF-THEN-ELSE⟩ should already be matched.

HW#6 Problem 4 (b)

A new unambiguous grammar G':

⟨STMT⟩ → ⟨ASSIGN⟩ | ⟨IF-THEN⟩ | ⟨IF-THEN-ELSE⟩

⟨IF-THEN⟩ → `if condition then` ⟨STMT⟩

⟨IF-THEN-ELSE⟩ → `if condition then` ⟨STMT-M⟩ `else` ⟨STMT⟩

⟨STMT-M⟩ → ⟨ASSIGN⟩ | ⟨IF-THEN-ELSE-M⟩

⟨IF-THEN-ELSE-M⟩ → `if condition then` ⟨STMT-M⟩ `else` ⟨STMT-M⟩

⟨ASSIGN⟩ → `a:=1`

We guarantee that all `if` and `else` in -M variables have already been matched.

HW#6 Problem 5

(Problem 2.32; 20 points) Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that, if A is context free and B is regular, then A/B is context free.

HW#6 Problem 5

建構 A/B 對應的 PDA
令 PDA $M_A = (Q_A, \Sigma, \Gamma, \delta_A, q_{0A}, F_A)$
DFA $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$
建構 $M_{A/B}$ 的思路是這樣子:
第一階段: 先把 w 餵給 M_A
第二階段: 再「假裝」輸入一些 symbols，同時跑在 M_A 與 M_B 上
可以想像，第一階段並不需要理會 M_B，所以只需要記錄 M_A 的狀態
而第二階段就需要同時記錄兩台機器的狀態
當兩台機器都待在 accepting state 時就行了

HW#6 Problem 5

$M_{A/B} = (Q, \Sigma, \Gamma, \delta, q_0, F)$
$Q = Q_A \cup (Q_A \times Q_B)$

左邊的 Q_A 是第一步使用的 state，右邊的 $Q_A \times Q_B$ 是第二步用的

$q_0 = q_{0A}$
$F = F_A \times F_B$

HW#6 Problem 5

第一階段：模擬 M_A
對於 $q_A \in Q_A, b \in \Gamma$
若 $a \in \Sigma$，直接模擬 M_A 的動作
$\delta(q_A, a, b) = \delta_A(q_A, a, b)$

若是 $a = \epsilon$，有兩種路：模擬 M_A 的動作，或者跳轉到第二步驟
跳轉瞬間不該消耗 stack 裡頭的符號，因為這並不是 M_A 的動作，所以會把消耗掉的 b 還回去
跳轉後 M_A 應該待在本來的狀態，而 M_B 從 q_{0B} 開始
$\delta(q_A, \epsilon, b) = \delta_A(q_A, \epsilon, b) \cup \{((q_A, q_{0B}), b)\}$

HW#6 Problem 5

第二階段：同步模擬 M_A, M_B
對於 $q_A \in Q_A, q_B \in Q_B, b \in \Gamma$
若是 $a \in \Sigma$
實際輸入 symbols 的程序應該在第一階段完結，第二階段不能吃。
$\delta((q_A, q_B), a, b) = \{\}$

若 $a = \epsilon$
雖然實際上這機器不會再接收輸入，但我們還是得「假裝」有 symbols 打進去
要怎麼模擬? 把此時輸入每個 symbol 或 ϵ 的可能性彙總起來
$\delta((q_A, q_B), \epsilon, b) =$
$\{((q'_A, q'_B), c) \mid \exists a \in \Sigma.(q'_A, c) \in \delta_A(q_A, a, b) \wedge q'_B \in \delta_B(q_B, a)\}$

$\cup$

$\{((q'_A, q'_B), c) \mid (q'_A, c) \in \delta_A(q_A, \epsilon, b) \wedge q'_B = q_B\}$

HW#6 Problem 5

Therefore, A/B is context free.

HW#6 Problem 6

(10 points) Prove (using the pumping lemma) that $\{a^m b^n c^{m \times n} \mid m, n \geq 1\}$ is not context free.

HW#6 Problem 6

Let $A = \{a^m b^n c^{m \times n} \mid m, n \geq 1\}$
Prove that A is not context free.
Use the pumping lemma.
Let s be $a^p b^p c^{p \times p}$, where p is the pumping length for A.

HW#6 Problem 6

Cases of dividing s as $uvxyz$ (where $|vy| > 0$ and $|vy| \leq p$):

- Both v and y contain only one type of symbol, e.g., $\overbrace{a \cdot \underbrace{\cdots}_{v} \cdot a}^{p} \overbrace{b \cdot \underbrace{\cdots}_{y} \cdot b}^{p} \overbrace{c \cdots c}^{p \times p}$, in which case, uv^2xy^2z will have wrong number of c's which is not equal to :
  the number of a's $\times$ the number of b's,
  $a^{p+i}b^{p+j}c^{p \times p}$
  or
  $\overbrace{a \cdots a}^{p} \overbrace{b \cdot \underbrace{\cdots}_{v} \cdot b}^{p} \overbrace{c \cdot \underbrace{\cdots}_{y} \cdot c}^{p \times p}$, in which case, uv^2xy^2z will also have wrong number of c's which is not equal to :
  the number of a's $\times$ the number of b's,
  $a^{p}b^{p+i}c^{p \times p+j}$

HW#6 Problem 6

Cases of dividing s as $uvxyz$ (where $|vy| > 0$ and $|vy| \leq p$):

- Either v or y contains more than one type of symbol, e.g., $a \cdot \underbrace{\cdots}_{v} \underbrace{\cdots}_{x} \underbrace{\cdot ab \cdot}_{y} \cdots \cdot bc \cdots c$, in which case, uv^2xy^2z will have some a's and b's out of order and so is not in A.

HW#6 Problem 7

(Problem 2.57; 10 points) Let $A = \{wtw^R \mid w, t \in \{0,1\}^* \text{ and } |w| = |t|\}$. Prove that A is not context free.

HW#6 Problem 7

Use the pumping lemma.
Let s be $1^p0^p(01)^p0^p1^p$, where p is the pumping length.
Divide s as $uvxyz$ such that $|vy| > 0$ and $|vxy| \leq p$.
Note also that the $(2p+1)$-th symbol of s is a 0, while the last $(2p+1)$-th symbol is a 1; similarly, the $(2p+2)$-th symbol is a 1, while the last $(2p+2)$-th symbol is a 0.

HW#6 Problem 7

Case 1: vxy falls (entirely) within the substring 1^p0^p.
If either v or y saddles on the middle point and contains both 1 and 0, then when we pump down, the first p symbols will contain some trailing 0s and cannot be the reverse of 1^p at the end of the resulting string (which is of length at least $3p$).
Otherwise, either v contains some 1s but no 0s or both v and y contain only 0s.
In the first case, when we pump up, the first $2p$ symbols will have more 1s than 0s and hence cannot be the reverse of 0^p1^p.
In the second case, when we pump up, the $(2p+1)$-th symbol will remain a 0 and the last $(2p+1)$-th symbol will also remain a 1 and hence the resulting string (of length greater than $6p$) cannot be of the form wtw^R (with w of length at least $2p+1$).

HW#6 Problem 7

Case 2: vxy falls within $0^p(01)^{\frac{p}{2}}$
In this case, no matter what v and y contain, when we pump up, the $(2p+1)$-th symbol will remain a 0, while the last $(2p+1)$-th symbol will remain a 1, and hence the resulting string (of length greater than $6p$) cannot be of the form wtw^R (with w of length at least $2p+1$).

Case 3: vxy falls within $(01)^p$. This is analogous to Case 2.

HW#6 Problem 7

Case 4: vxy falls within $(01)^{\frac{p}{2}}0^p$.

Case 4-1: Both v and y are in $(01)^{\frac{p}{2}}$. When we pump up, the $(2p+1)$-th symbol will remain a 0 and the last $(2p+1)$-th symbol will remain a 1 and hence the resulting string (of length greater than 6p) cannot be of the form wtw^R (with w of length at least $2p+1$).

Case 4-2: v is within $(01)^{\frac{p}{2}}$ and y saddles on the middle point. The same argument for Sub- case (a) applies, when we pump up.

HW#6 Problem 7

Case 4-3: v is within $(01)^{\frac{p}{2}}$ and y is within 0^p. If y is nonempty, then when we pump up $(i = 2)$, the last $(2p + 2)$-th symbol will become a 0, while the $(2p + 2)$-th symbol will remain a 1; otherwise (y is empty), when we pump up, the same argument for subcase 4-1 applies.

Case 4-4: v saddles on the middle point and y is within 0^p. This is analogous to subcase 4-3.

Case 4-5: both v and y are within 0^p. This is also analogous to subcase 4-3.

HW#6 Problem 7

Case 5: vxy falls within 0^p1^p. This is analogous to Case 4-3.

By pumping lemma, $A = \{wtw^R \mid w, t \in \{0,1\}^* and |w| = |t|\}$ is not context free.

HW#7 Problem 1

(Exercise 3.1; 10 points) Consider the Turing machine for $\{0^{2^n} \mid n \geq 0\}$ discussed in class. Give the sequence of configurations (using the notation uqv for a configuration) that the machine goes through when started on the input 000000.

HW#7 Problem 1

q_1 0 0 0 0 0 0
␣ q_2 0 0 0 0 0
␣ x q_3 0 0 0 0
␣ x 0 q_4 0 0 0
␣ x 0 x q_3 0 0
␣ x 0 x 0 q_4 0
␣ x 0 x 0 x q_3
␣ x 0 x 0 q_5 x
␣ x 0 x q_5 0 x
␣ x 0 q_5 x 0 x
␣ x q_5 0 x 0 x
␣ q_5 x 0 x 0 x
q_5 ␣ x 0 x 0 x

␣ q_2 x 0 x 0 x
␣ x q_2 0 x 0 x
␣ x x q_3 x 0 x
␣ x x x q_3 0 x
␣ x x x 0 q_4 x
␣ x x x 0 x q_4
␣ x x x 0 x ␣ q_{reject}

HW#7 Problem 2

(20 points) Give a *formal* description (with a state diagram) of a Turing machine that decides the language $\{w \in \{0,1\}^* \mid w$ is nonempty and contains an equal number of 1s and 0s$\}$.

HW#7 Problem 2

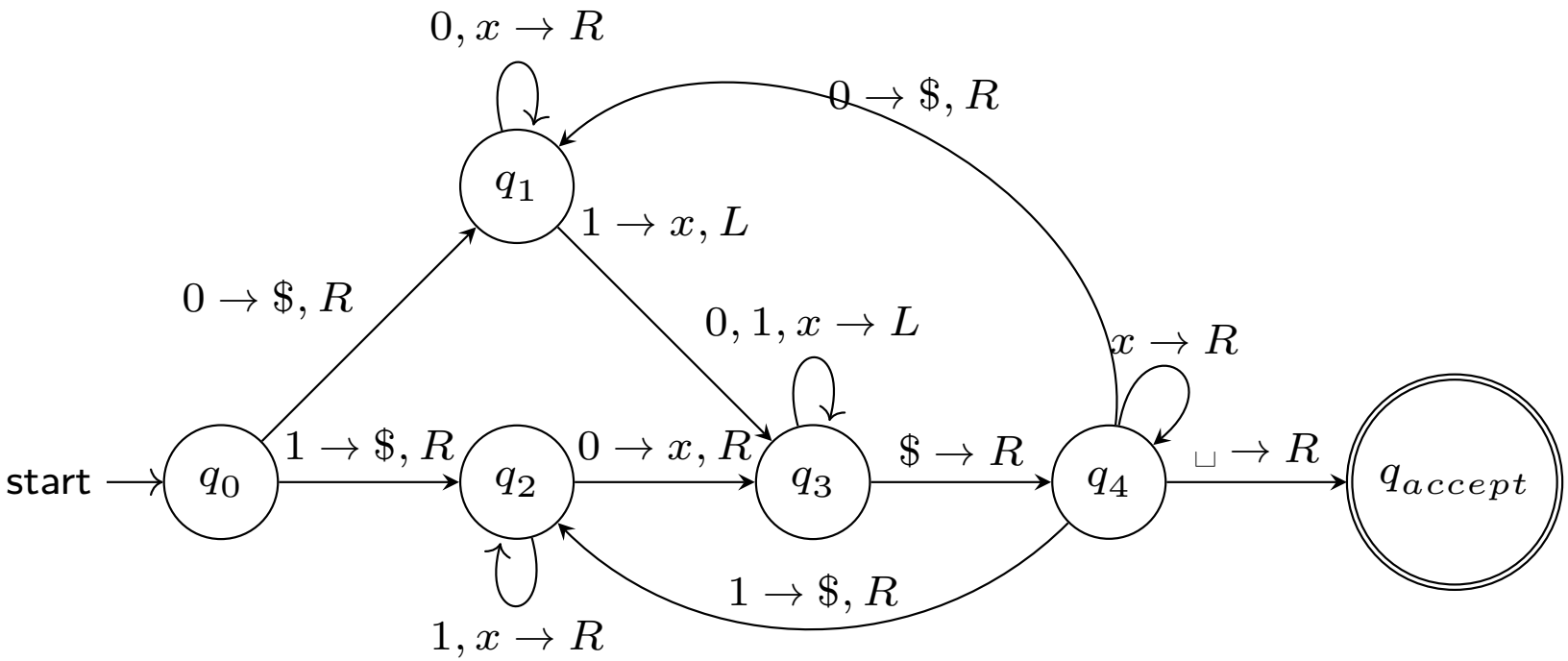
$L = \{\omega \in \{0,1\}^* \mid \omega$ is nonempty and contains equal number of 1s and 0s $\}$.

TM_L is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where

- $Q = \{q_0, q_1, q_3, q_4, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, 1, \sqcup, \$\}$,
- q_0 is the initial state,
- q_{accept} is the accept state,
- q_{reject} is the reject state, and

HW#7 Problem 2

- $\delta =$ (all undescribed transitions lead to q_{reject})



HW#7 Problem 3

(Exercise 3.7; 10 points) Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "The input is a polynomial p over variables $x_1, \ldots, x_k$:

(a) Try all possible settings of $x_1, \ldots, x_k$ to integer values.

(b) Evaluate p on all of these settings.

(c) If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

HW#7 Problem 3

不是合法的 Turing machine，原因是並沒有寫出 (a) 步驟列舉所有 $x_1, ..., x_k$ 的方法
$=>$ 不同的列舉方法可能產生不同的結果!

HW#7 Problem 4

(Problem 3.16; 10 points) Show that the collection of decidable languages is closed under *concatenation*.

HW#7 Problem 4

做出一台圖靈機 decide 兩個 decidable language
假設兩個 decidable language A, B 對應到的 Decider M_A, M_B
做出 M = "On input w,
1. Divide w into xy ($|w|+1$ different division)
2. Input x to M_A and y to M_B (try any possible with $|w|+1$ division)
3. Repeat Step 1 and 2, if both M_A M_B accept on some x y, accept, otherwise, reject."
由於 w 是有限長度字串，它的分割法只有 $|w|+1$ 種
而且 Decider M_A 與 M_B 都會停機
所以 M 也一定會在有限時間停機，M decides the concatenation of A and B

HW#7 Problem 5

(Problem 3.18; 10 points) A ***Turing machine with a doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape, as it moves left. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

HW#7 Problem 5

要說明兩端無限長的圖靈機，辨識效果和一般圖靈機相同
首先我們用兩端無限長的圖靈機來模擬一般圖靈機
概念上很簡單，就是封印左邊的紙帶不用
實作上就是先在輸入的左方放一個標記，之後只要踩到這個標記
就往右退回去

HW#7 Problem 5

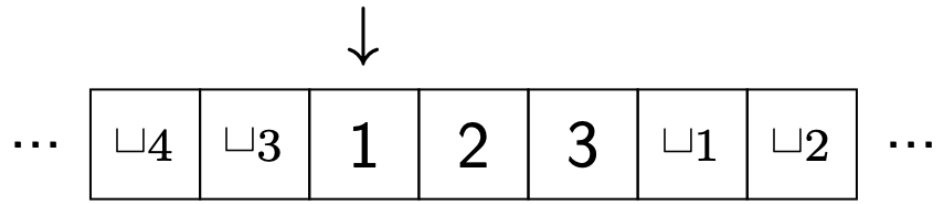
接下來要說明一般圖靈機有辦法模擬兩端無限長的圖靈機
依照課堂上講過的定理，對兩條紙帶的圖靈機都有一個一般圖靈機與之等價
所以這邊我們利用一台有兩條紙帶的圖靈機去模擬
效力等同於拿一個一般圖靈機去模擬它

HW#7 Problem 5

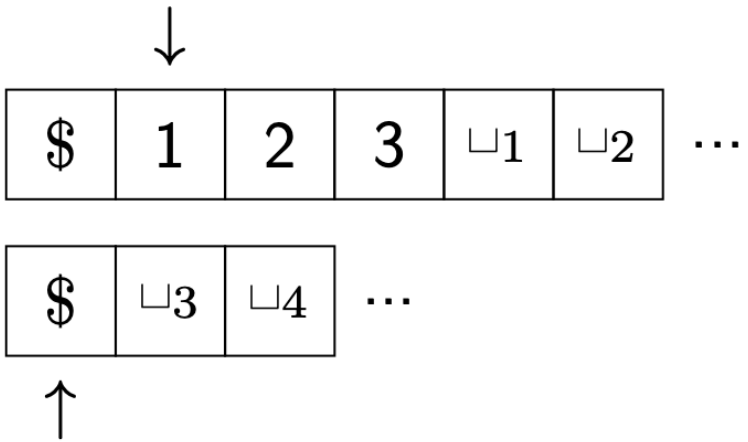
兩條紙帶的第一條代表兩端無限長圖靈機的右半邊紙帶，第二條代表左半邊
輸入的字串都是放在第一條紙帶上
第一條用來模擬右半邊的運算，第二條則模擬左半邊

HW#7 Problem 5

為方便理解，補充實作上的細節
首先將第一條紙帶的輸入往右移並在開頭加上標記
第二條紙帶也在開頭加上標記
比如這樣的輸入（空格加上編號以方便看出位置上的關聯）



在兩條紙帶的圖靈機上會先轉換成這樣再操作

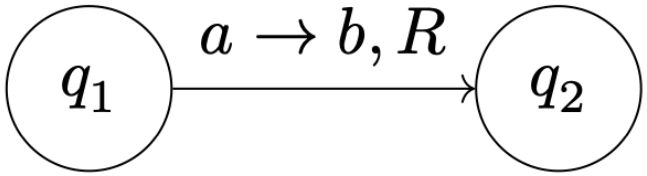


HW#7 Problem 5

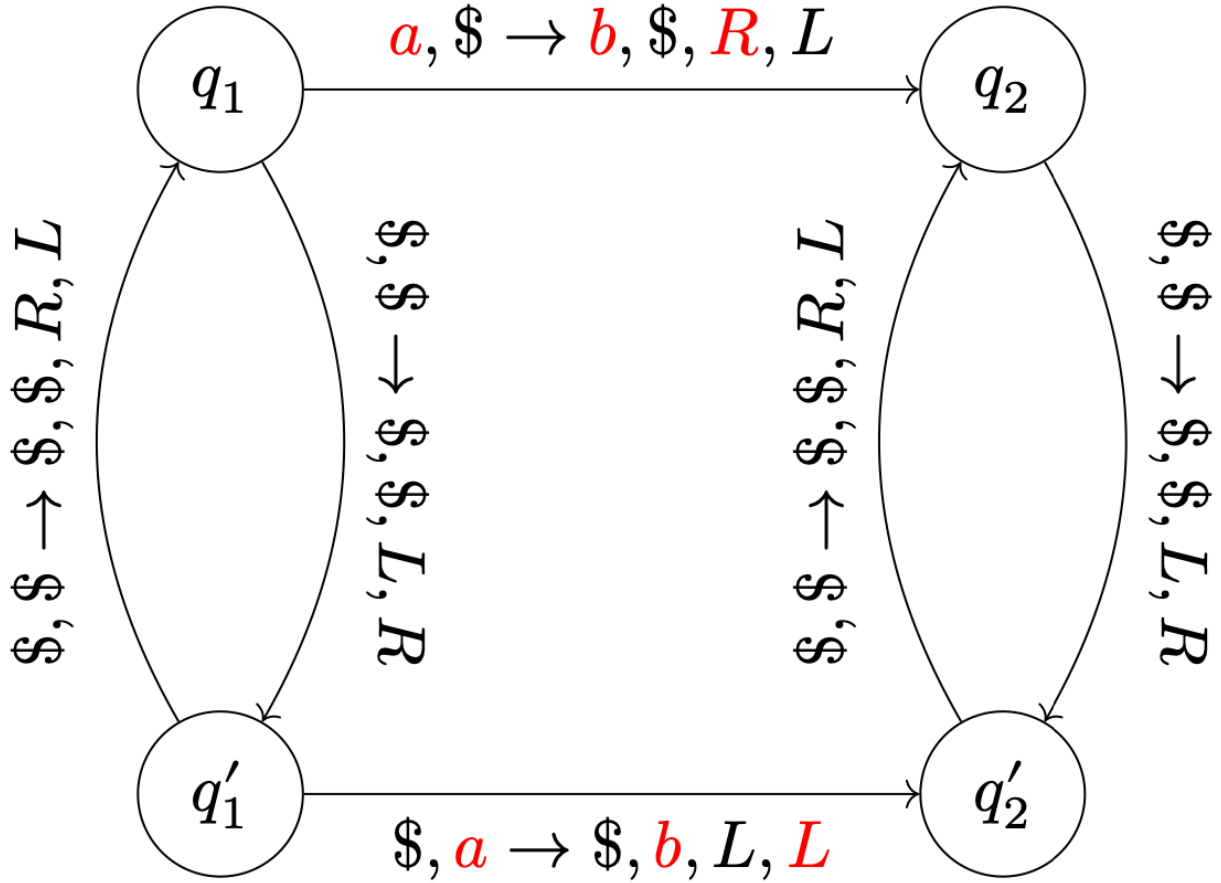
將 state diagram 複製出一份並將 LR 顛倒
原本的 diagram 給第一條紙帶用，以模擬右半邊紙帶，另一個以此類推
在左半右半切換時 (對應到碰上 $ 時) 移動到另一個 state diagram

HW#7 Problem 5

也就是說原本這樣的 transition



會變成



HW#7 Problem 6

(Problem 3.20; 20 points) A ***Turing machine with stay put instead of left*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, S\}$$

At each point the machine can move instead its head right or let it stay in the same position. Show that this Turing machine variant is *not* equivalent to the usual version. What class of languages do these machines recognize?

HW#7 Problem 6

一台只能往右與停在原地的圖靈機，它的辨識能力究竟爲何？

有一個關鍵是在與這個圖靈機沒辦法往回讀取它在左邊所寫過的玩意
這個永遠不回頭的特性，和某種東西好像有點像...
PDA 能夠把前面的內容塞到 stack，所以似乎不是 PDA
NFA/DFA 呢？

HW#7 Problem 6

我們先用這種圖靈機去模擬一台 DFA
很簡單，把 DFA 的 transition 加上指針不寫入且往右移
並在原本的 accepting states 加入一條讀取空格則跑到 q_{accept} 的 transition 即可

HW#7 Problem 6

那麼要如何用 NFA 模擬這種圖靈機？
這個地方稍微有點難懂

當這台圖靈機往右走的時候，其實我們不需要理會它在原本那格寫了什麼，因爲它永遠不回頭
但若這台圖靈機在某格一直待著，我們勢必要記錄現在這格到底內容是什麼
我們會利用 **states** 去記錄
也就是說，除了原本圖靈機的狀態集 Q 之外，我們還需要 $Q \times \Gamma$，包含同時存著圖靈機狀態與紙帶當前字元的 pairs

HW#7 Problem 6

那麼要怎麼把輸入字串餵給 NFA
當圖靈機第一次來到某一格，因爲永遠不回頭，這格若不是空格就是輸入字元
如果是輸入字元，就對應到 NFA 輸入字元的動作
其餘地方都是 ϵ-transition，利用 state 本身記錄紙帶內容

那若是往右遇到空格了呢？
我們把一開始在 q 狀態遇到空格記爲一個 pair $[q, \sqcup]$
Q 與 Γ 都是有限集合，持續走 $|Q| \times |\Gamma|$ 步之後必然會遇到相同的 pair (遇到相同 pair 時的環境都一樣：右邊都是無盡的空格)
就可以確認這機器不會停機，也就是這機器不接受這個字串
反過來如果從 q 持續走若干步之後到達 q_{accept}，則將 q 狀態當成 accepting state
若 NFA 輸入完字串後停在這裡，就相當於接受了這個字串
我們把符合條件的這種 q 收集起來做成集合 F

HW#7 Problem 6

那麼 NFA 裡頭包含 q_{accept} 的狀態要怎麼處理？
因爲圖靈機是碰到 q_{accept} 就直接停機
所以 NFA 的這些狀態應該會有一條 ϵ-transition 連到一個永遠待在原地的 accepting state
令這個 accepting state 爲 q'_{accept}
那麼 NFA 的 accepting states 就是 $F \cup \{q'_{accept}\}$

HW#7 Problem 6

假設原本圖靈機的 transition function 爲 δ，而 NFA 的 transition relation 是 $\delta'(q' \in \delta'(q_a))$
若是圖靈機在 q 狀態接收到一個 $a \in \Sigma$，而下述的 $X \in \Gamma$
$\delta(q, a) = (q', X, R)$，因爲往右走所以不需要理會 X，所以
$(q, a, q') \in \delta'$
$\delta(q, a) = (q', X, S)$，因爲停下來了，需要記錄 X，所以
$(q, a, (q', X)) \in \delta'$
這邊會實際吃掉輸入字元

若是圖靈機在 q 狀態接收到一個 $X \in \Gamma$，而下述的 $Y \in \Gamma$
$\delta(q, X) = (q', Y, R)$，因爲往右走所以不需要理會 Y，所以
$((q, X), \epsilon, q') \in \delta'$
$\delta(q, X) = (q', Y, S)$，因爲停下來了，需要記錄 Y，所以
$((q, X), \epsilon, (q', Y)) \in \delta'$
這邊會用 ϵ-transition 去模擬紙帶的運作

HW#7 Problem 6

NFA 模擬 TM 在 q 狀態接收一個 $a \in \Sigma$，而下述的 $X \in \Gamma$：
$(q, a, q') \in \delta'$
$(q, a, (q', X)) \in \delta'$
NFA 模擬 TM 在 q 狀態接收到一個 $X \in \Gamma$，而下述的 $Y \in \Gamma$
$((q, X), \epsilon, q') \in \delta'$
$((q, X), \epsilon, (q', Y)) \in \delta'$

NFA 模擬 TM 處理 accepting state:
$(q_{accept}, \epsilon, q'_{accept}) \in \delta'$
$((q_{accept}, X), \epsilon, q'_{accept}) \in \delta'$ for all $X \in \Gamma$
$(q'_{accept}, a, q'_{accept}) \in \delta'$ for all $a \in \Sigma$

HW#7 Problem 6

弄了這麼長，結論就是我們能用這種圖靈機模擬 DFA，也能用 NFA 模擬這種圖靈機
因爲 DFA 與 NFA 的辨識能力是相同的，所以這種圖靈機的辨識能力也和它們相同
所以這種圖靈機能辨識的語言就局限於 regular languages

HW#7 Problem 7

(Problem 3.22; 20 points) Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognizing a larger class of languages) than 0-PDAs.

(a) Show that 2-PDAs are more powerful than 1-PDAs.

(b) Show that 3-PDAs are *not* more powerful than 2-PDAs. (Hint: simulate a Turing machine tape with two stacks.)

HW#7 Problem 7

第一小題要說明 2-PDA 比 1-PDA 強
很明顯 2-PDA 當然能夠模擬 1-PDA
那要如何證明 1-PDA 無法模擬 2-PDA？
那我們就找一個 language，可以被一個 2-PDA 給辨識，但卻不是 context-free
我們挑 $0^n1^n0^n1^n$，已知它不是 context-free

HW#7 Problem 7

流程大致是這樣：
在兩個 stacks 當中塞入一個識別符號 $
吃若干個 0 放入第一個 stack
吃 1 並把 0 從第一個 stack pop 掉並把 1 塞入第二個 stack
吃 0 並把 1 從第二個 stack pop 掉並把 0 塞入第一個 stack
吃 1 並把 0 從第一個 stack pop 掉
當兩個 stacks 的頂端都是 $ 則跳到 accepting state (若還有字沒輸入完成，輸進去就會爆掉)
這樣我們就建構出一個能辨識這個語言的 2-PDA

HW#7 Problem 7

注意，兩個方向都要給出說明 2-PDA 可以模擬 1-PDA，但 1-PDA 沒辦法辨識某些 2-PDA 能辨識的語言
這樣才能說明 2-PDA 能辨識的語言集合嚴格大於 1-PDA 的

HW#7 Problem 7

第二小題，說明 3-PDA 的辨識能力與 2-PDA 一樣
這邊往另外一個方向，證明這兩種機器都與另外一種機器具有一樣的能力

HW#7 Problem 7

首先我們要用圖靈機去模擬 2-PDA
用 3-tape TM 來模擬
一號紙帶代表 PDA 的輸入，在有實際輸入時才向右
二號三號分別代表兩個 stacks
指針指到的字代表 stack 頂的內容，一開始先填充識別符號代表 stack 爲空
PDA 有 pop 代表會偵測指針指到的字
如果有 pop 且沒有塞字進去，則代表填入空格且向左
有 pop 也有塞入，則代表填入塞入的字且停在原地
沒有 pop 也沒有塞字，則停在原地
沒有 pop 而有塞字，則向右並在右邊這格寫入 (一個 R 再一個 S)

HW#7 Problem 7

那麼如何用 2-PDA 去模擬 TM？
首先先在 stack 底部填上識別符號
再來吃入所有輸出到一個 stack 裡頭
此時 stack 頂會是最尾巴的字，我們看不到開頭是什麼
所以就把所有字倒到另一個 stack
基於 stack 的性質，現在另一個 stack 的頂端就會是第一個字元了
我們把這個 stack 的頂端當成圖靈機的指針指向的位置
這個 stack (暫時稱爲 1 號 stack) 代表指針與其右方，另外一個 (2 號 stack) 代表左方，距離頂端越遠，代表離指針越遠

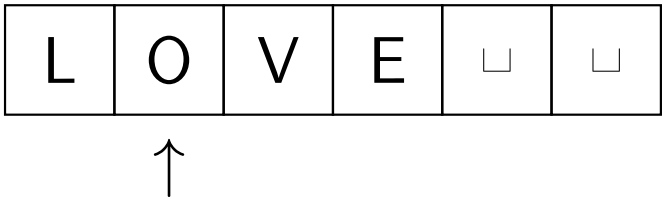
HW#7 Problem 7

圖靈機指針向右，就是從 1 號 stack pop 掉並把圖靈機寫入的字元塞到 2 號 stack
當 1 號 stack 看到識別符號，代表圖靈機指針初次跑到一個很右的地方
因爲在那之前圖靈機還沒到過，所以這裡自然會是空格，所以就先把空格塞入 1 號 stack 再算下去

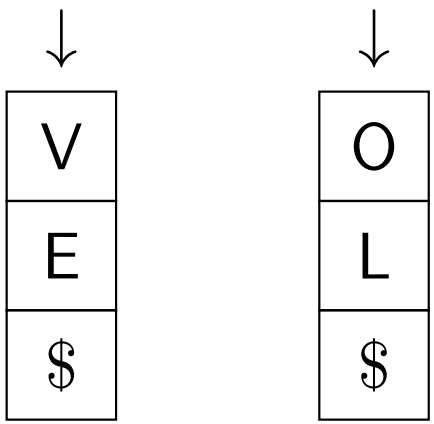
圖靈機指針向左，就是先對 1 號 stack 做 pop，塞入 TM 所寫入的字，再從 2 號 stack pop 字元塞入 1 號 stack
若是 2 號 stack 已經到底了，代表圖靈機跑到左邊的端點，上述的 pop 動作就不會執行

HW#7 Problem 7

想像一台圖靈機現在處於這樣的狀態

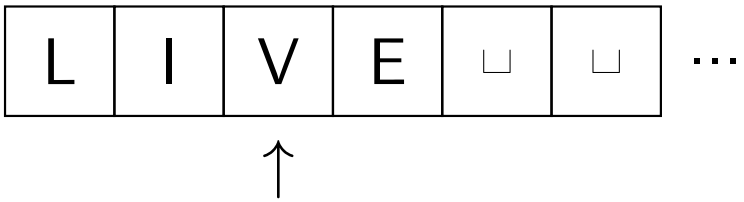


那麼 2-PDA 就可能 (依照處理過程不同，1 號 stack 的底部可能有更多東西) 是這樣：



1 號 stack　2 號 stack

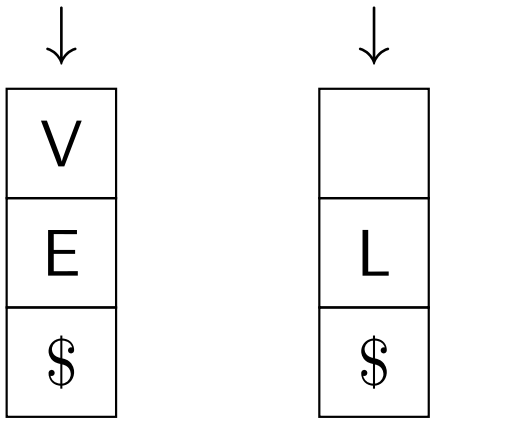
現在的指針正指在 O 的位置
假設我們想要做一個 $O \to I, R$ 的操作
在 TM 上的結果呈現會是：



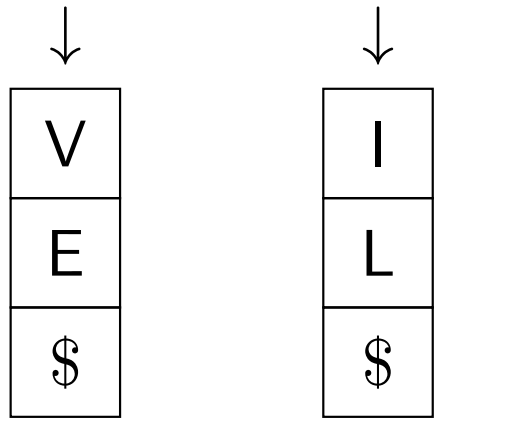
而想要用 2-PDA 呈現，其步驟會是：

1. 把 O 從 stack 2 pop 出來
2. 把 I push 進 stack 2
3. 對指針做操作：把 V 從 stack 1 pop 出來
4. 把 V push 進 stack 2

HW#7 Problem 7

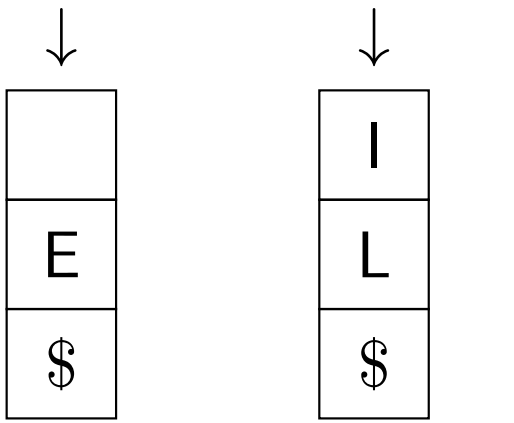


1 號 stack 2 號 stack

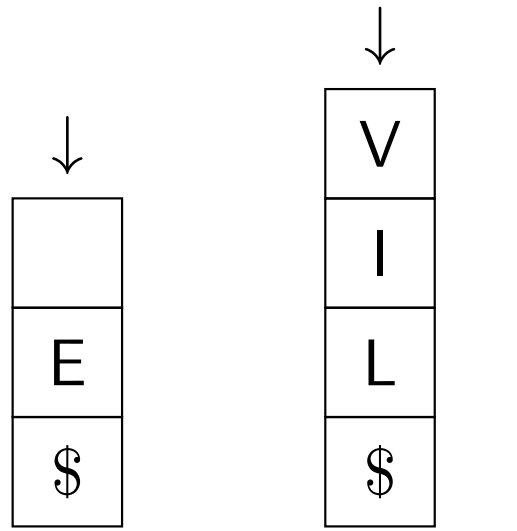


1 號 stack 2 號 stack

HW#7 Problem 7



1 號 stack 2 號 stack



1 號 stack 2 號 stack

HW#7 Problem 7

因爲 3-tape TM 可以模擬 2-PDA，2-PDA 可以模擬 TM，而 3-tape TM 與 TM 的能力一樣，結論就是 2-PDA 與圖靈機的辨識能力一樣
而這些事情代入到 3-PDA 也能成立 (4-tape TM 模擬 3-PDA、3-PDA 用其中兩個 stack 就能模擬 TM)
所以 3-PDA 與 2-PDA 的辨識能力都與圖靈機相同，兩者能力相等