Homework 6 - 10

曾守瑜

Menu

Hw6 Problem1

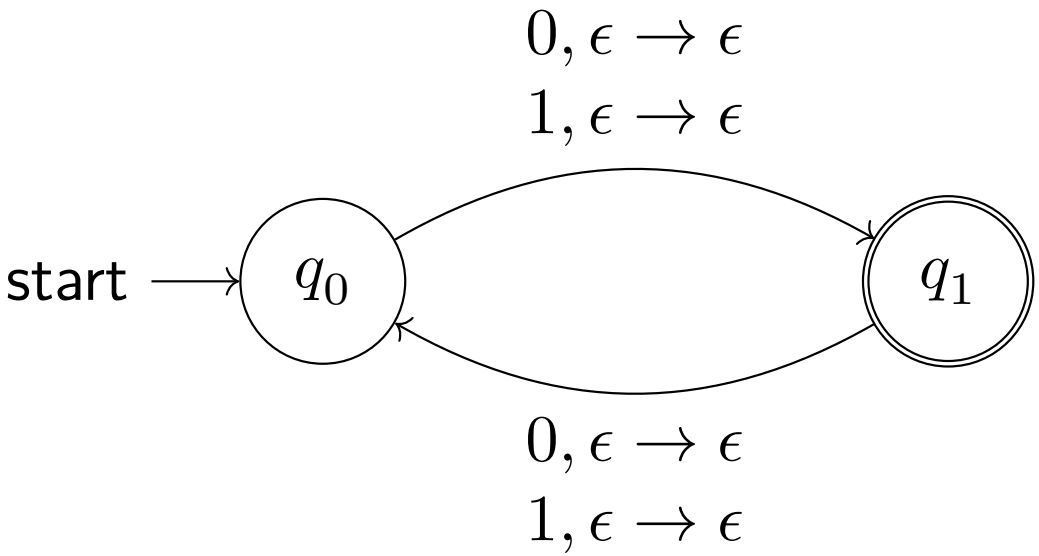
將兩個語言的定義稍微改寫成等價的形式：
$A = \{a^i b^j c^k \mid (i = j) \wedge (i, j, k \geq 0)\}$
$B = \{a^i b^j c^k \mid (j = k) \wedge (i, j, k \geq 0)\}$
那麼兩個語言的交集就會是
$C = \{a^i b^j c^k \mid (i = j) \wedge (j = k) \wedge (i, j, k \geq 0)\}$
這與題目給的 $\{a^n b^n c^n \mid n \geq 0\}$ 是等價的
已知 A 與 B 各自是 context-free，而兩者的交集 C 卻不是
可以得知 the class of context-free languages 並不封閉於交集

Hw6 Problem1

上一小題已經得知兩個 context-free 語言的交集不見得會是 context-free
而根據笛摩根定律，$A \cap B = \overline{\overline{A} \cup \overline{B}}$
已知兩個 context-free 語言的聯集也是 context-free
假設 context-free 語言的 complement 也是 context-free
那麼 $\overline{A}$、$\overline{B}$、$\overline{A} \cup \overline{B}$ 與 $\overline{\overline{A} \cup \overline{B}}$ 都一定是 context-free
但 $A \cap B$ 卻不一定是 context-free，矛盾
所以 the class of context-free languages 並不封閉於 complement

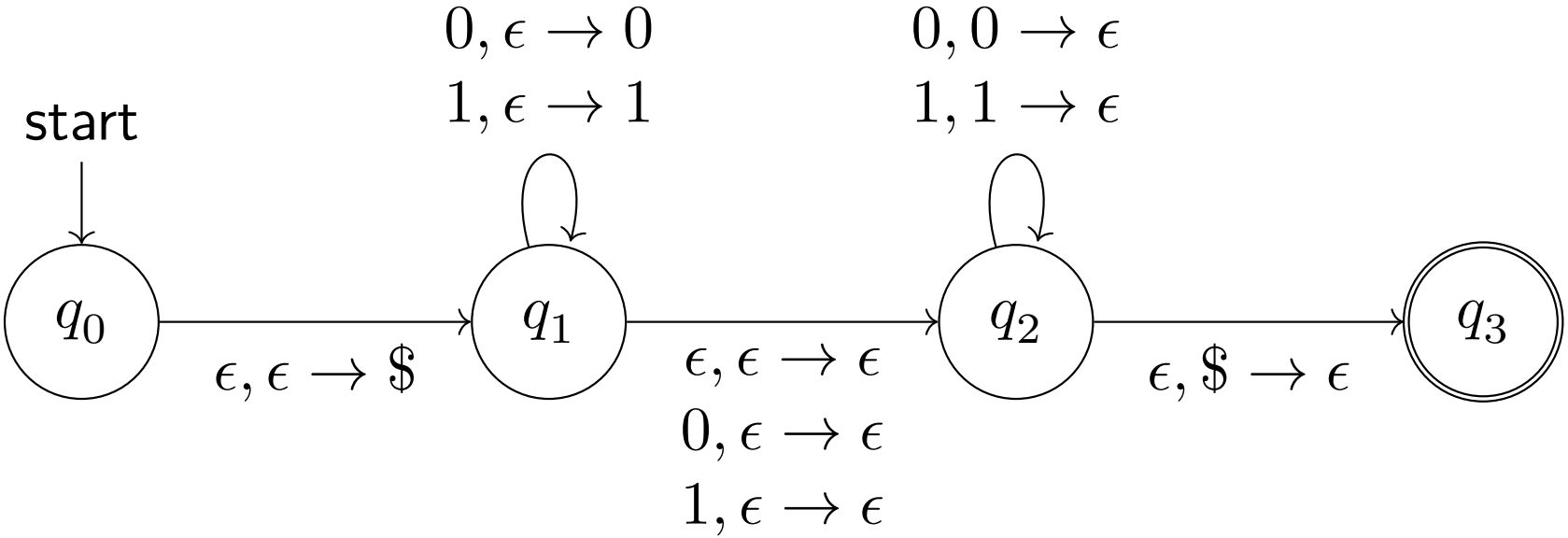
Hw6 Problem2

第一小題 $\{w \mid \text{the length of } w \text{ is odd}\}$
這個本身就是 regular language，直接把它的 DFA 轉成 PDA



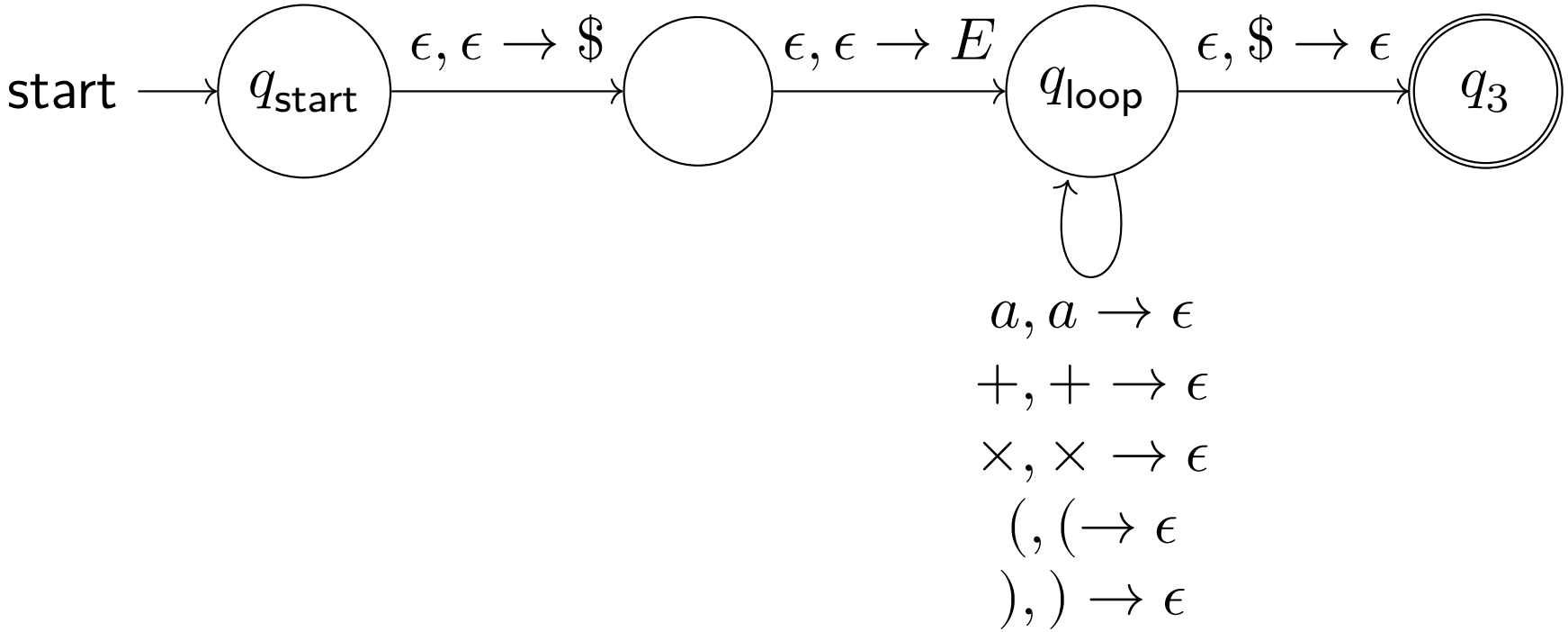
Hw6 Problem2

第二小題 $\{w \mid w \text{ is a palindrome}\}$
分成兩步驟：吃字並把字塞入 stack，以及吃字並把 stack 的字 pop 掉
兩步驟之間的轉移是 ϵ（回文長度為偶數）或吃某一個字（回文長度為奇數）
要用 $ 符號來標記 stack 是否已經空了



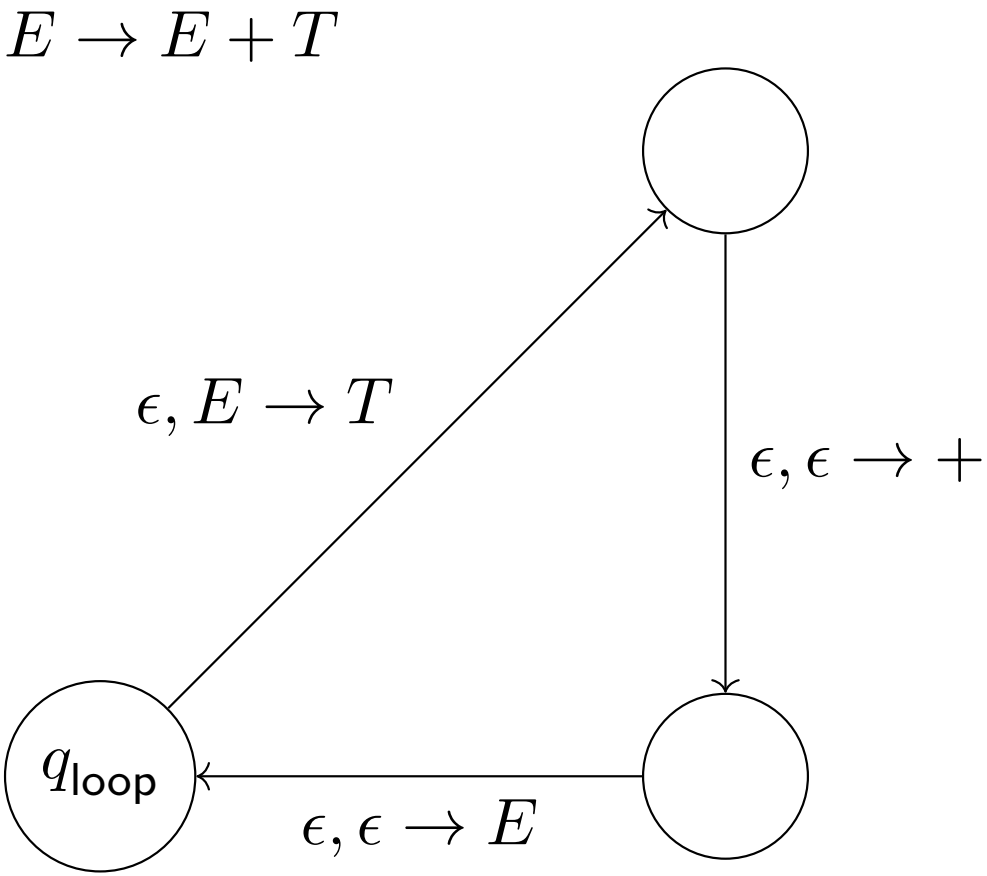
Hw6 Problem3

按照程序搭建出 PDA
首先在開頭塞入 $ 與 E，並在結尾 pop 掉 $
並且對所有 terminal $a \in \Sigma$，都有一個自迴圈，輸入 a 並從 stack pop 掉 a



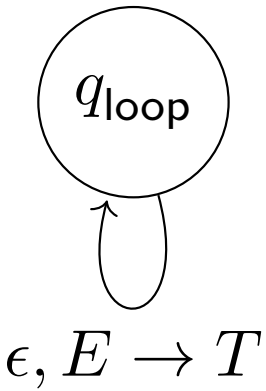
接下來把 CFG 的 production rules 放進去

Hw6 Problem3

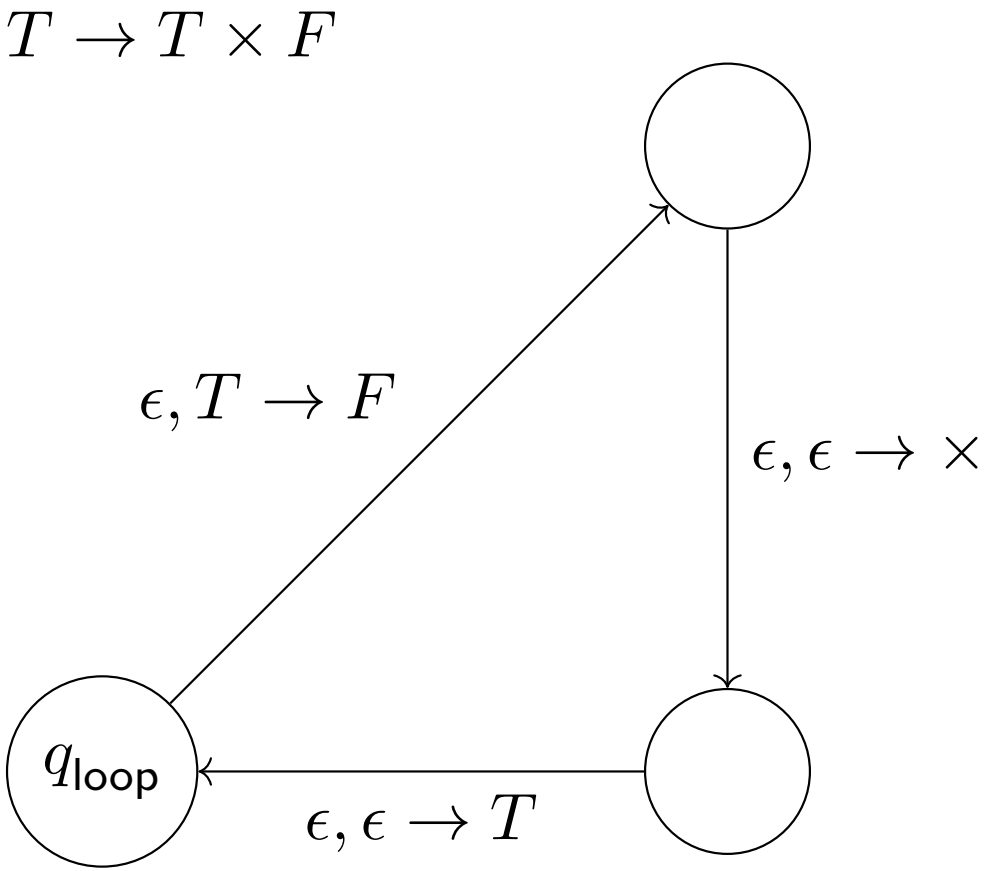


Hw6 Problem3

$E \to T$

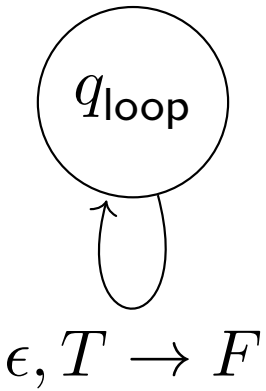


Hw6 Problem3

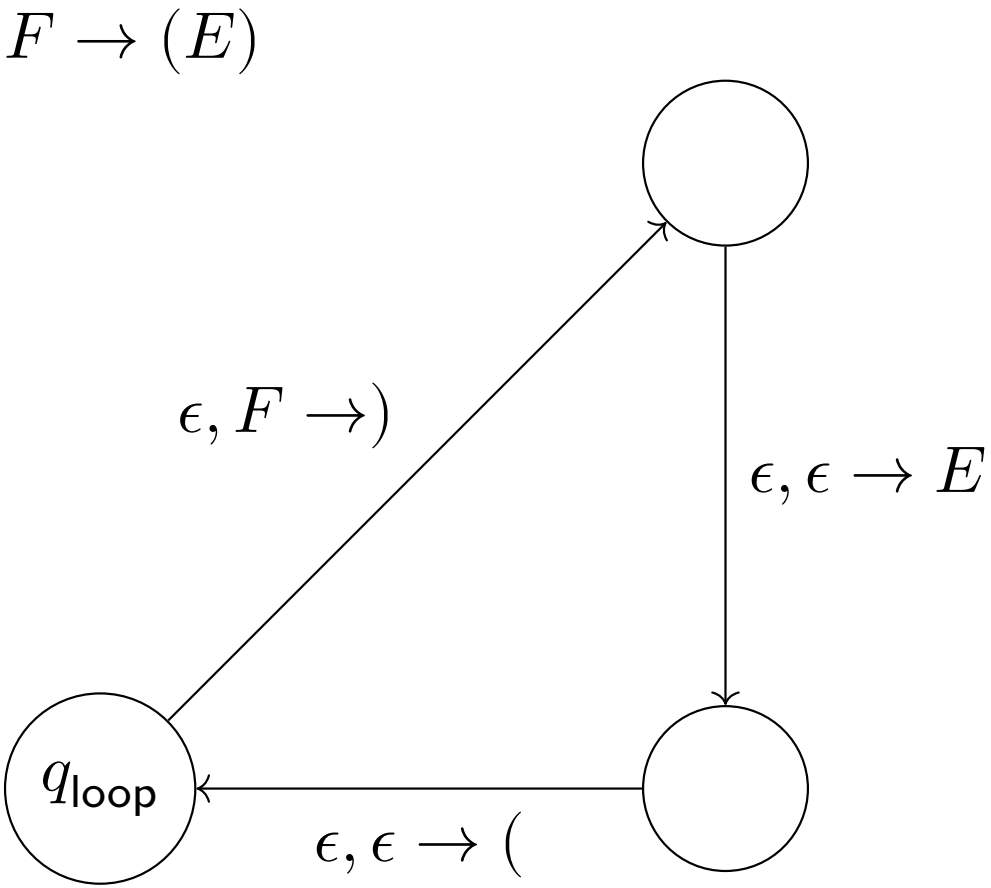


Hw6 Problem3

$T \to F$

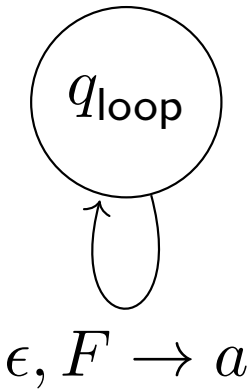


Hw6 Problem3



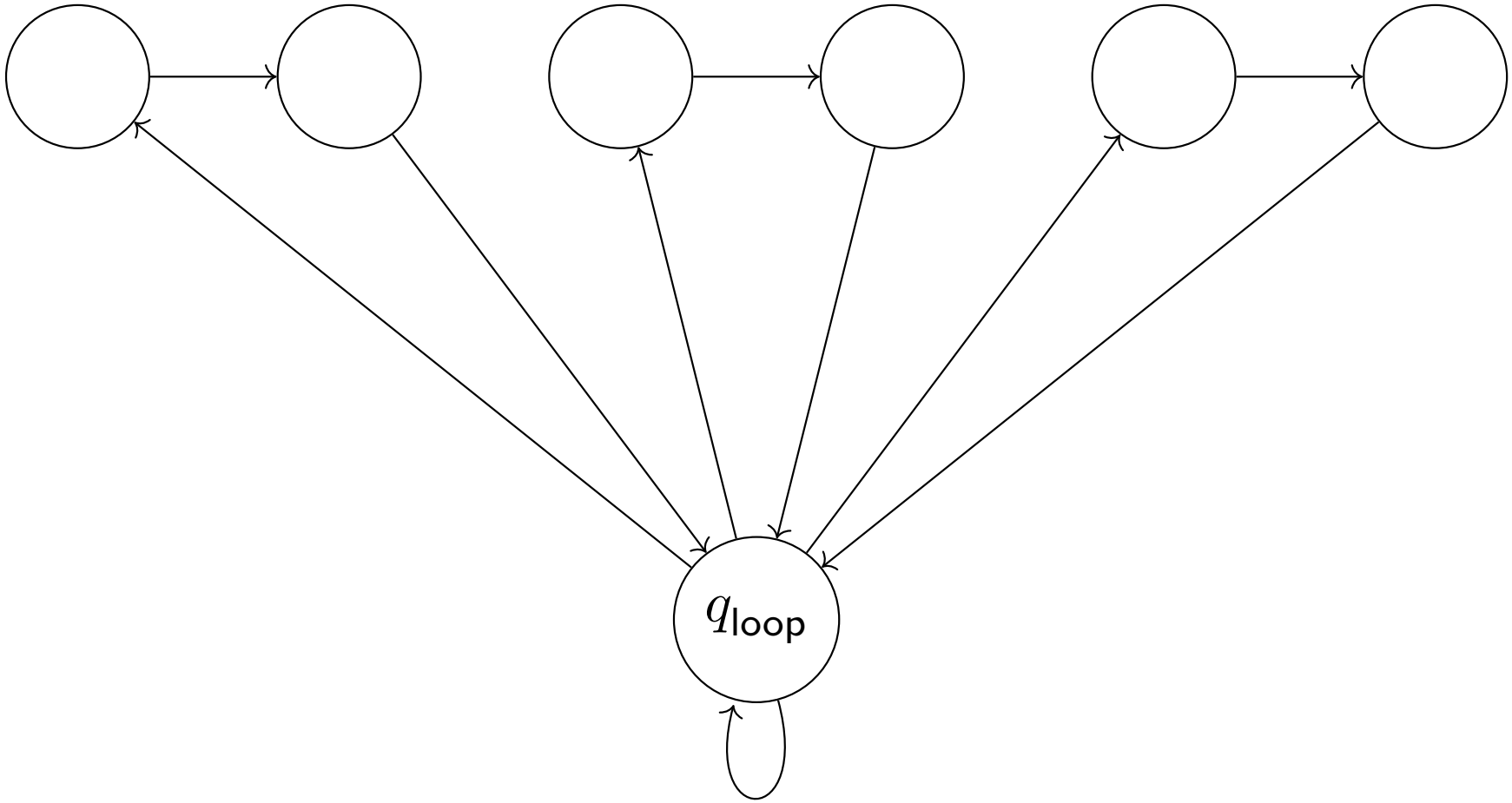
Hw6 Problem3

$F \to a$



Hw6 Problem3

q_{loop} 最終會像這樣：



$$a, a \to \epsilon \quad +, + \to \epsilon \quad \times, \times \to \epsilon$$
$$(, (\to \epsilon \quad),) \to \epsilon$$
$$\epsilon, E \to T \quad \epsilon, T \to F \quad \epsilon, F \to a$$

Hw6 Problem4

給定上下文無關語言 A 與正規語言 B
試著說明 $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$ 也是 context-free

Hw6 Problem4

OK，來建構 A/B 對應的 PDA 吧
令 PDA $M_A = (Q_A, \Sigma, \Gamma, \delta_A, q_{0A}, F_A)$
NFA $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$
建構 $M_{A/B}$ 的思路是這樣子：
第一步：先把 w 餵給 M_A
第二步：再「假裝」輸入一些字，同時跑在 M_A 與 M_B 上
可以想像，第一步並不需要理會 M_B，所以只需要記錄 M_A 的狀態
而第二步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

Hw6 Problem4

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

$$Q = Q_A \cup Q_A \times Q_B$$
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

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

Hw6 Problem4

第一步：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
$\delta(q_A, \epsilon, b) = \delta_A(q_A, \epsilon, b) \cup \{((q_A, q_{0B}), b)\}$

Hw6 Problem4

第二步：同步模擬 M_A 與 M_B
對於 $q_A \in Q_A, q_B \in Q_B, b \in \Gamma$
若是 $a \in \Sigma$
實際輸入字元的程序應該在第一步完結，第二步不能吃
$\delta((q_A, q_B), a, b) = \{\}$

若是 $a = \epsilon$
雖然實際上這機器不會再接收輸入，但我們還是得「假裝」有字打進去
要怎麼模擬？把此時輸入每個字元或 ϵ 的可能性彙總起來
$\delta((q_A, q_B), \epsilon, b) =$

$$\{((q'_A, q'_B), c) \mid \exists a \in \Sigma_\epsilon . (q'_A, c) \in \delta_A(q_A, a, b) \wedge q'_B \in \delta_B(q_B, a)\}$$

也就是，模擬某個字 a 同時輸入給 M_A 與 M_B 時的反應

Hw6 Problem4

於是我們得出了 A/B 是 context-free language 的結論

Hw6 Problem5

G 是 ambiguous 的

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

右邊的 else 可以與第一個 if 配對，也可以與第二個 if 配對

一開始 STMT 走 IF-THEN-ELSE 路線，第一個 STMT 再生成 IF-THEN

或者一開始走 IF-THEN 路線，再生成 IF-THEN-ELSE

Hw6 Problem5

要怎麼修改 CFG 使其產生同樣的語言，而又不會 ambiguous
程式語言當中常見的作法是，讓 else 匹配前面最近尚未匹配的 if
也就是說

`if blabla else ......`

blabla 裡頭不應該出現還沒有被配對的 if
否則 else 就會去和 blabla 裡頭的 if 配對，而不是跟左邊的 if

Hw6 Problem5

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

被我加上 ALTER 的 variable 不會有沒被捕捉到的 if

Hw6 Problem6

用 pumping lemma 證明 $\{1^n0^n1^n0^n\}$ 不是 context-free
令 pumping length 為 p
挑出 $s = 1^p0^p1^p0^p$
把 s 分割成 $uvxyz$，且 $|vxy| \leq p$
vxy 有可能是全都是 0、全都是 1、前面 1 後面 0，與前面 0 後面 1 共四種可能

Hw6 Problem6

vxy 全都是 0：uxz 的 0 會變少
vxy 全都是 1：uxz 的 1 會變少
vxy 前面 1 後面 0：
可能是左邊的 1^p0^p 或右邊的 1^p0^p
不管是哪個，v 與 y 的數量變化都會使某一邊的 0 1 數量與另一邊 (p 個) 不一致
vxy 前面 0 後面 1：
一定是中間的 0^p1^p
總之 v 與 y 的數量變化一定會使中間的 0 或 1 數量不等於 p

得證 $\{1^n0^n1^n0^n\}$ 不是 context-free

Hw7 Problem1

以下程序完全參照課堂上的圖靈機
首先是檢查 0
$q_1 01\#01$
$xq_2 1\#01$
$x1q_2\#01$
$x1\#q_4 01$
$x1q_6\#x1$
$xq_7 1\#x1$
$q_7 x1\#x1$
$xq_1 1\#x1$

Hw7 Problem1

接下來檢查 1

$xq_1 1 \# x1$ （接續上頁）

$xxq_3 \# x1$

$xx \# q_5 x1$

$xx \# x q_5 1$

$xx \# q_6 xx$

$xx q_6 \# xx$

$x q_7 x \# xx$

$xx q_1 \# xx$

Hw7 Problem1

再來是收尾

$xxq_1\#xx$ （接續上頁）

$xx\#q_8xx$

$xx\#xq_8x$

$xx\#xxq_8\sqcup$

$xx\#xx\sqcup q_{\text{accept}}\sqcup$

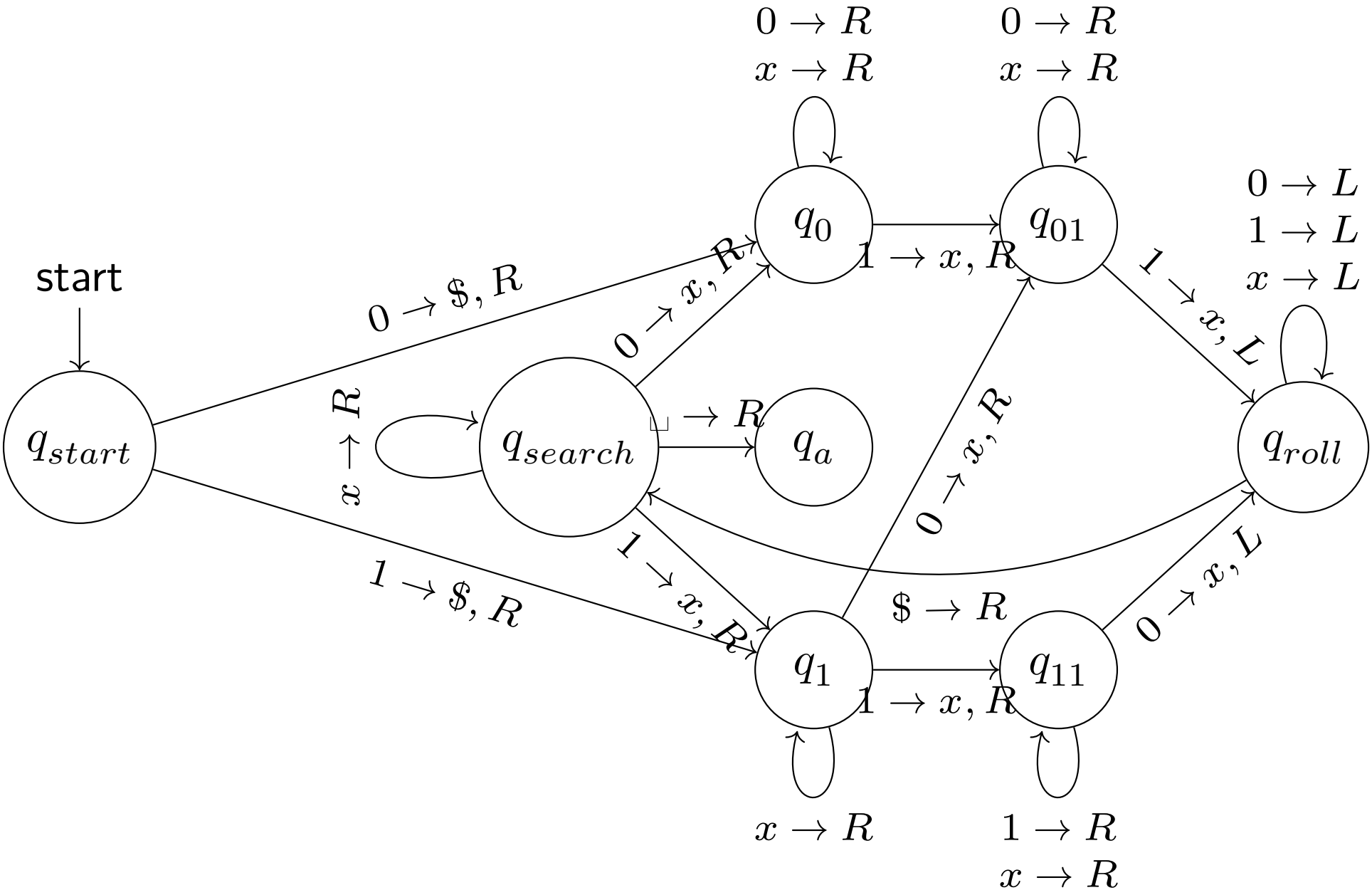
Hw7 Problem2

非空字串，且 1 的數量是 0 的數量的兩倍
吃到 0 就找後面有沒有兩個 1
吃到 1 就找後面有沒有 0 與 1 各一個
用 x 來標註已經找過的格子
每找完一次就回頭
需要注意的是，我們要有東西標註起點在哪
所以第一次吃字會有特別處理，順便偵測輸入是否為空

Hw7 Problem2

$M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_a, q_{reject})$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, \sqcup, x, \$\}$

Hw7 Problem2



Hw7 Problem2

可以看到初始狀態 q_{start} 與之後往右搜尋的狀態 q_{search} 最大的不同是，q_{start} 要處理紙帶的開頭
放入特殊的識別符號 $ 使後續作業得以進行
並順便確認輸入是否為空

q_0 代表目前吃到一個 0，需要在後面找兩個 1
q_1 代表目前吃到一個 1，需要在後面找 0 與 1
q_{01} 代表吃到 0 與 1，需要找剩下的 1
q_{11} 代表吃到兩個 1，需要找剩下的 0
q_{roll} 則將指針往左推到開頭為止
若 q_{serach} 一路吃 x 而後吃到空格，則計算完畢，輸入字串符合條件

Hw7 Problem3

為何這台機器不是合法的圖靈機？
這台機器會試圖嘗試所有可能性
但「所有」是多少？
要將係數是任何整數的可能性都考慮進去
所以會有無窮多種可能性
如果係數只有一個，那就是 0 1 2 ... 嘗試下去
但是如果有兩個呢？
00 10 20...？還是 00 10 01 20 11 02...？
這台機器並沒有說明它的執行順序

Hw7 Problem3

另外一個毛病是 “otherwise, reject” 是多餘的
因為可能性有無限多種，所以永遠不可能觸發這句

不過影響這台機器合法性的重點還是上一頁，如何遍歷這無限多種設定

以下是補充說明，這台機器能不能改成合法的圖靈機，去 recognize 同一個 language？
由於多項式的係數數量是有限的，有限個整數組合數量是**可數**無限多的
所以整個遍歷過程是可以在圖靈機上跑的
所以結論是可行的，決定好遍歷順序是關鍵

Hw7 Problem4

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
2. Input x to M_A and y to M_B
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內停機，M decides the concatenation of A and B

Hw7 Problem5

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

Hw7 Problem5

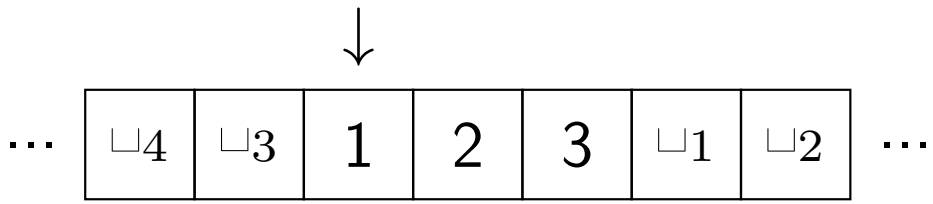
接下來要說明一般圖靈機有辦法模擬兩端無限長的圖靈機
依照課堂上講過的定理，對兩條紙帶的圖靈機都有一個一般圖靈機與之等價
所以這邊我們利用一台有兩條紙帶的圖靈機去模擬
效力等同於拿一個一般圖靈機去模擬它

Hw7 Problem5

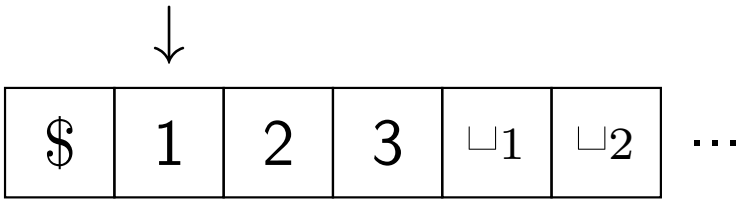
兩條紙帶的第一條代表兩端無限長圖靈機的右半邊紙帶，第二條代表左半邊
輸入的字串都是放在第一條紙帶上
第一條用來模擬右半邊的運算，第二條則模擬左半邊

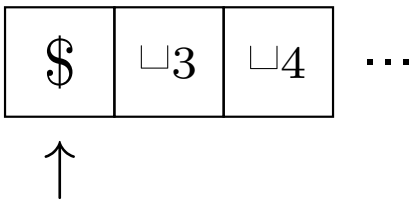
Hw7 Problem5

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



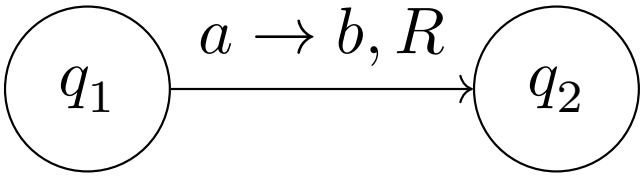


Hw7 Problem5

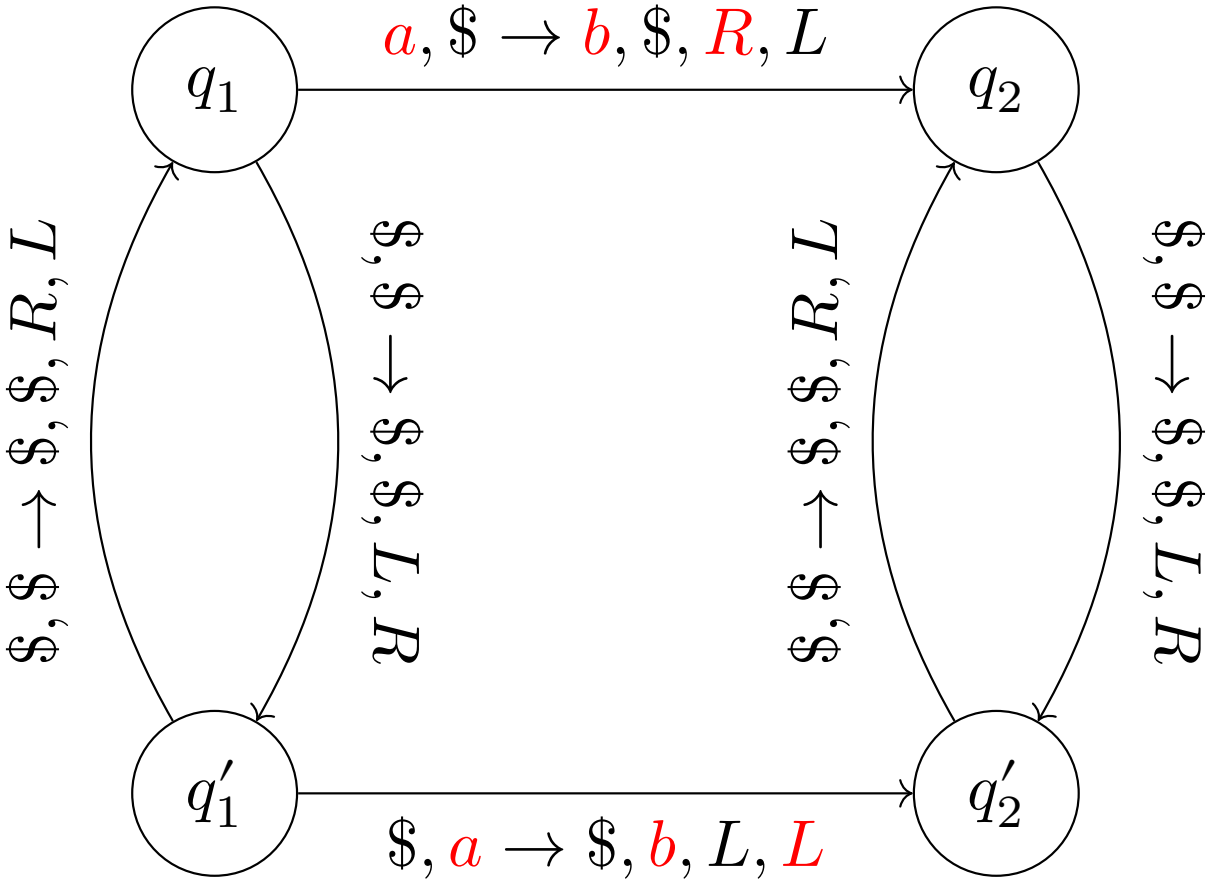
將 state diagram 複製出一份並將 LR 顛倒
原本的 diagram 給第一條紙帶用，以模擬右半邊紙帶，另一個以此類推
在左半右半切換時 (對應到碰上 $ 時) 移動到另一個 state diagram

Hw7 Problem5

也就是說原本這樣的 transition



會變成



Hw7 Problem6

一台只能往右與停在原地的圖靈機，它的辨識能力究竟為何？

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？

Hw7 Problem6

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

Hw7 Problem6

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
因為它永遠不回頭
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

Hw7 Problem6

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，因為永遠不回頭，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 狀態遇到空格記為一個 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

Hw7 Problem6

那麼 NFA 裡頭包含 q_{accept} 的狀態要怎麼處理？
因為圖靈機是碰到 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 為 q'_{accept}
那麼 NFA 的 accepting states 就是 $F \cup \{q'_{accept}\}$

Hw7 Problem6

假設原本圖靈機的 transition function 為 δ，而 NFA 的 transition relation 是 δ'
若是圖靈機在 q 狀態接收到一個 $a \in \Sigma$，而下述的 $X \in \Gamma$
$\delta(q, a) = (q', X, R)$，因為往右走所以不需要理會 X，所以
$(q, a, q') \in \delta'$
$\delta(q, a) = (q', X, S)$，因為停下來了，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 $Y \in \Gamma$
$\delta(q, X) = (q', Y, R)$，因為往右走所以不需要理會 Y，所以
$((q, X), \epsilon, q') \in \delta'$
$\delta(q, X) = (q', Y, S)$，因為停下來了，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

Hw7 Problem6

$(q_{accept}, \epsilon, q'_{accept}) \in \delta'$
$((q_{accept}, X), \epsilon, q'_{accept}) \in \delta'$ for all $X \in \Gamma$
$(q'_{accept}, a, q'_{accept}) \in \delta'$ for all $a \in \Sigma$

Hw7 Problem6

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
因為 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

Hw7 Problem7

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

Hw7 Problem7

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

Hw7 Problem7

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

Hw7 Problem7

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

Hw7 Problem7

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 為空
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)

Hw7 Problem7

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 (暫時稱為 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

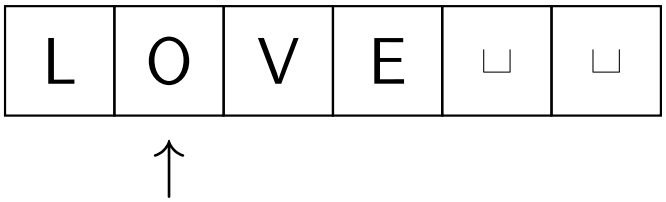
Hw7 Problem7

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
因為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

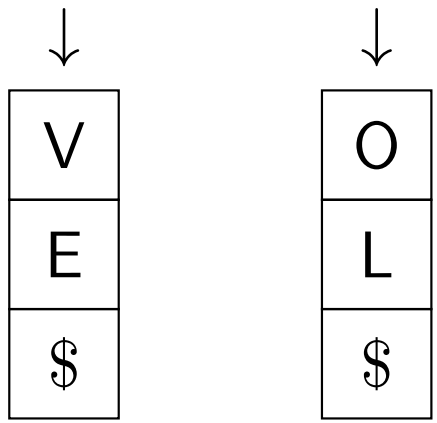
圖靈機指針向左，就是先對 1 號 stack 做 pop，塞入 TM 所寫入的字，再從 2 號 stack pop 字元塞入 1 號 stack
若是 2 號 stack 已經到底了，代表圖靈機跑到左邊的端點，上述的 pop 動作就不會執行

Hw7 Problem7

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

Hw7 Problem7

因為 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

Hw8 Problem1

設計一台圖靈機輸入字元往右移一格
並在原本的開頭填上空格

Hw8 Problem1

大家的作法主要分成兩種
從開頭往右，以及從結尾往左

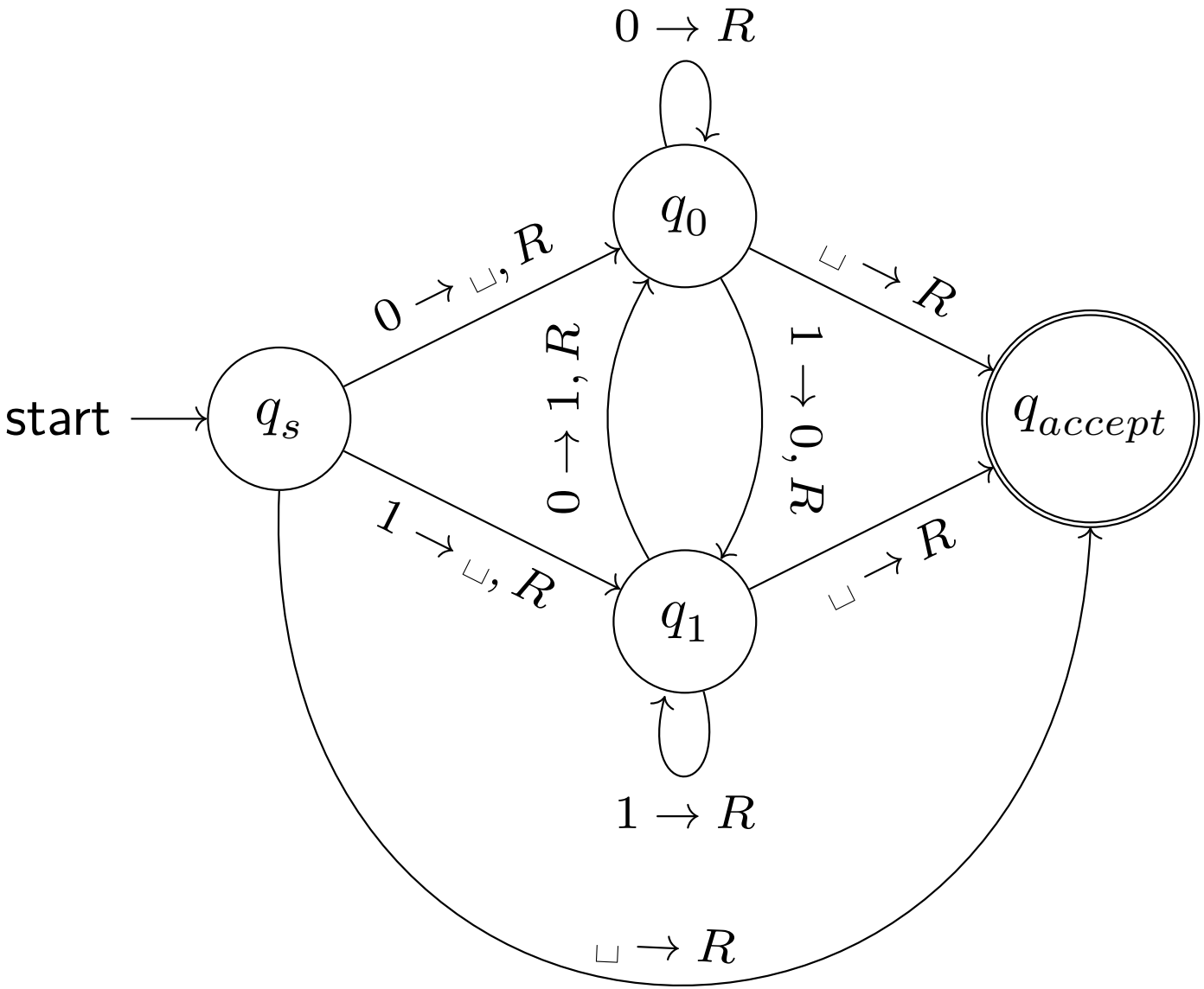
前者做起來會比較簡單
所以我在此使用從左到右的作法

Hw8 Problem1

$$M = (Q, \Sigma, \Gamma, \delta, q_s, q_{accept}, q_{reject})$$
$$Q = \{q_s, q_0, q_1, q_{accept}, q_{reject}\}$$
$$\Sigma = \{0, 1\}$$
$$\Gamma = \{0, 1, \sqcup\}$$

Hw8 Problem1

存取這一步吃到的字，並把上一步記下的字寫下去



Hw8 Problem2

一台 Enumerator 會有一個工作區域負責計算，一個印表機負責印出字串
所以可以把它當成兩條紙帶的圖靈機
一條運算，一條輸出 (只寫不讀)
我們讓輸出字串以某個不屬於輸出字母的符號為分隔，假設為 #

Hw8 Problem2

$E = (Q, \Sigma, \Gamma, \delta, q_0, q_h)$
Q is a finite set of states
Σ is the output alphabet without $\#$
Γ is the working tape alphabet
q_0 is the initial state
q_h is the halting state, the eumerator halts when meeting q_h

Hw8 Problem2

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \times (\Sigma \cup \{\epsilon, \#\})$
機器會像一般圖靈機那樣，依照現在的狀態與工作區指針指向的位置決定在工作區的指針要寫入什麼、要往左還是往右
同時會決定要在輸出區輸出什麼
可以輸出的字元是 $\sigma \in \Sigma \cup \{\#\}$，同時輸出區的指針往右
而當 δ 給出的最後一項是 ϵ，代表輸出區指針不做任何事

Hw8 Problem2

當機器執行過程中，輸出區紙帶呈現
$S_1\#S_2\#...S_n\#...$
被紙帶開頭與 $\#$，或被兩個 $\#$ 包圍的 $S \in \Sigma^*$
此 S 屬於 enumerated language

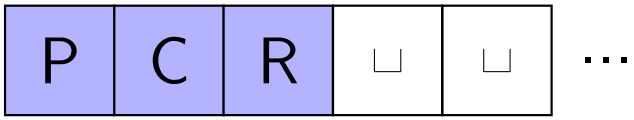
Hw8 Problem3

無法在輸入區寫入紙帶的圖靈機，辨識的範圍是否止於正規語言

說明這種圖靈機沒辦法記憶字串出現過的所有可能性，我會給對
不過這題要更精確說明的話...

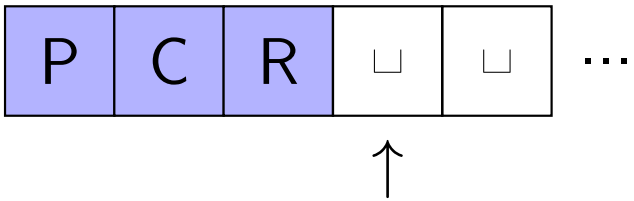
Hw8 Problem3

假設輸入一個字串



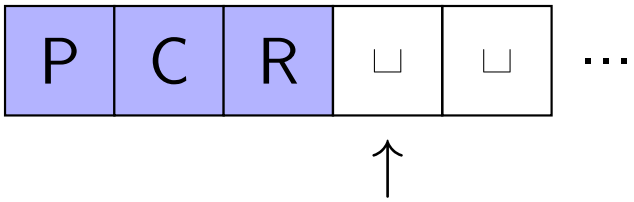
因為圖靈機無法修改輸入區的紙帶內容，只要指針卡在這區域內 $|Q| \times |s|$ 步以上還沒停機，就可以肯定不會停機

Hw8 Problem3



如果指針會跑出去，或者說發生 “out event”
第一次跑出去的時候，指針面對的是右邊一片白的紙帶
而此時圖靈機的狀態為 q

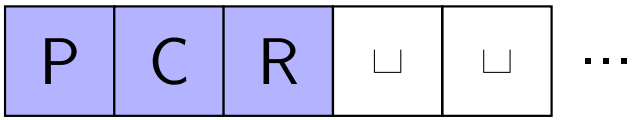
Hw8 Problem3



於是我們令，輸入一個字串 s 後，第一次發生 “out event” 的狀態為 first(s)
完整的定義如下：

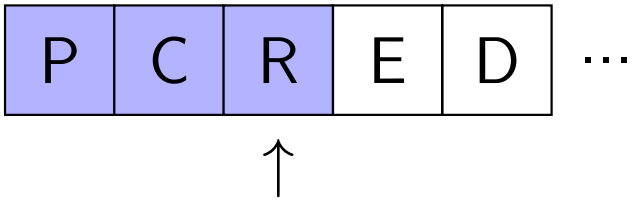
$$\text{first}(s) = \begin{cases} q & \text{輸入 } s \text{ 第一次 “out event” 時的狀態為 } q \\ q_{accept} & \text{若不會發生 “out event” 且停機於 } q_{accept} \\ q_{reject} & \text{otherwise (包含無法停機)} \end{cases}$$

Hw8 Problem3



我們無法完全掌握右邊會發生什麼事，因為取回了圖靈機原有的能力，所以停機問題無法判定
不過左邊我們能掌握
所以如果指針能再次回到左邊，或者說發生 “in event”
我們可以掌握指針會停在左邊、待在左邊無法停機，或者再次跑到右邊

Hw8 Problem3



我們可以令，對一個字串 s，發生 “in event” 的狀態為 q 時，再次發生 “out event” 時的狀態為 $\mathrm{f}(s, q)$

完整定義如下：

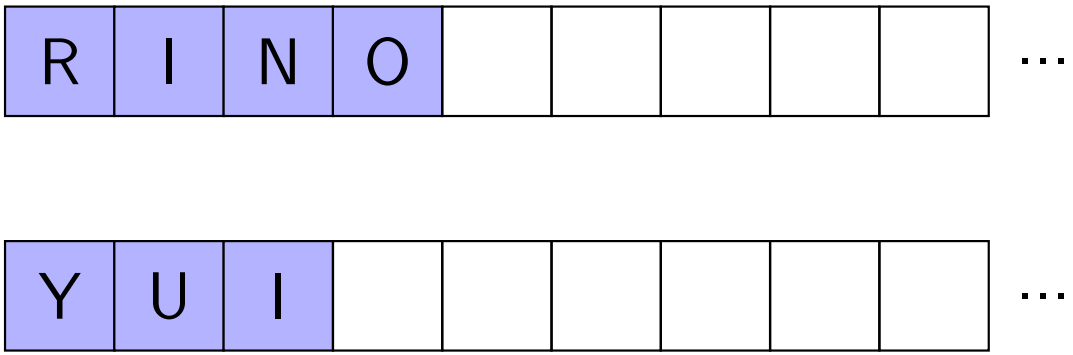
$$\mathrm{f}(s,q) = \begin{cases} q' & \text{再發生 “out event” 時的狀態為 } q' \\ q_{accept} & \text{若不會再發生 “out event” 且停機於 } q_{accept} \\ q_{reject} & \text{otherwise (包含無法停機)} \end{cases}$$

這個函數的成立是基於指針無法改變左邊的內容，從而使 $\mathrm{f}(s, q)$ 有固定結果

Hw8 Problem3

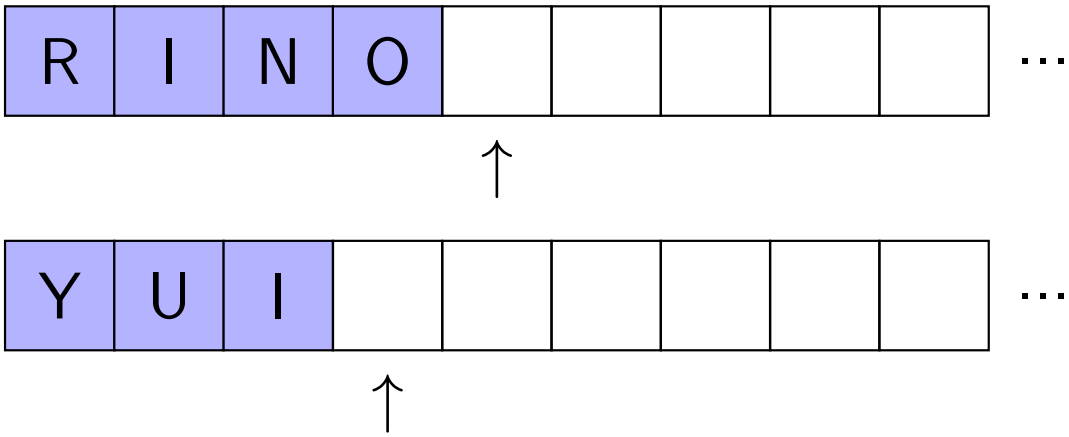
如果有兩個字串 s_1 與 s_2
符合 $\mathsf{first}(s_1) = \mathsf{first}(s_2)$
並且對所有的 $q \in Q$ 都符合 $\mathsf{f}(s_1, q) = \mathsf{f}(s_2, q)$
那這兩個字串有什麼關係呢？

Hw8 Problem3



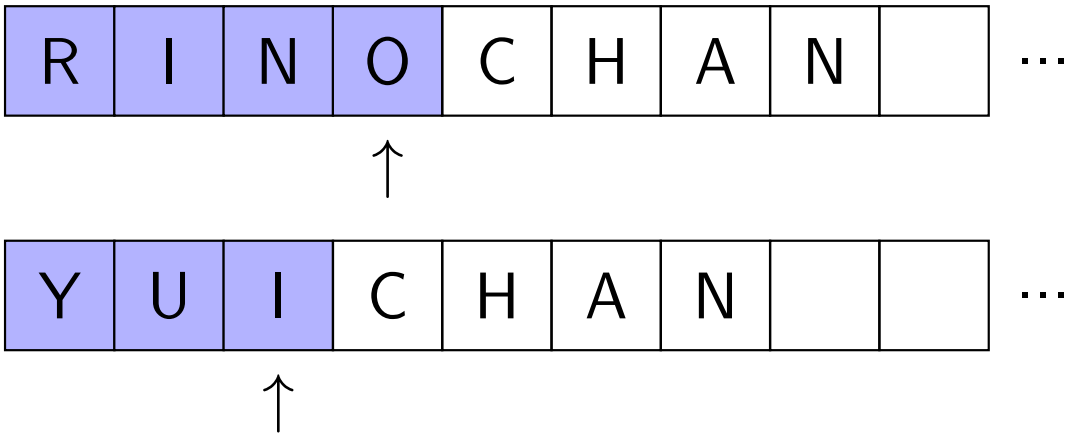
這兩個字串都有相同的 first 與 f
$\text{first}(s_1) = \text{first}(s_2) = q_{accept}/q_{reject}$
代表兩個字串都會在發生 “out event” 之前確定被接受，或是確定都不被接受

Hw8 Problem3



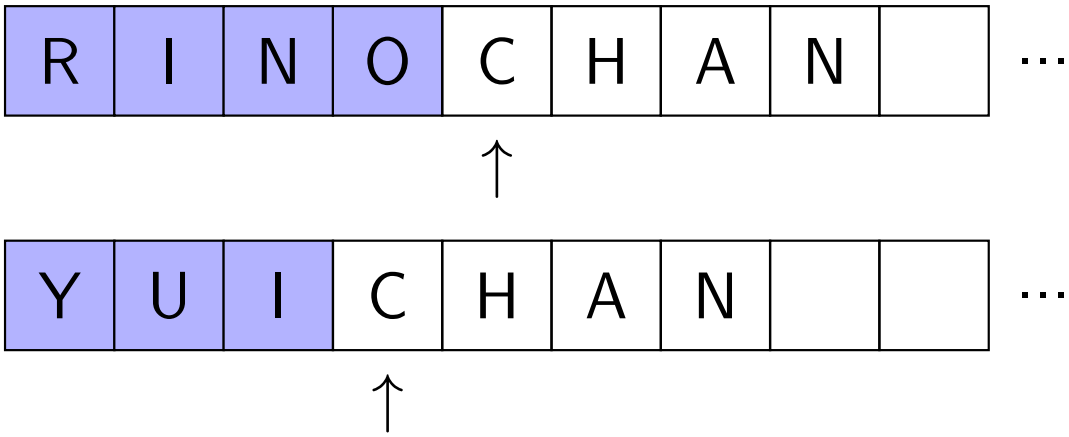
這兩個字串都有相同的 first 與 f
若是 first(s_1) = first$(s_2) \neq q_{accept}/q_{reject}$
在發生第一次 “out event” 時兩者處於相同的狀態
兩個指針的右邊都一樣是一望無際的空白
所以兩個指針在右邊的行為也一定是一模一樣的
如果在這過程中其中一邊 accept 了，另外一邊也一定會 accept

Hw8 Problem3



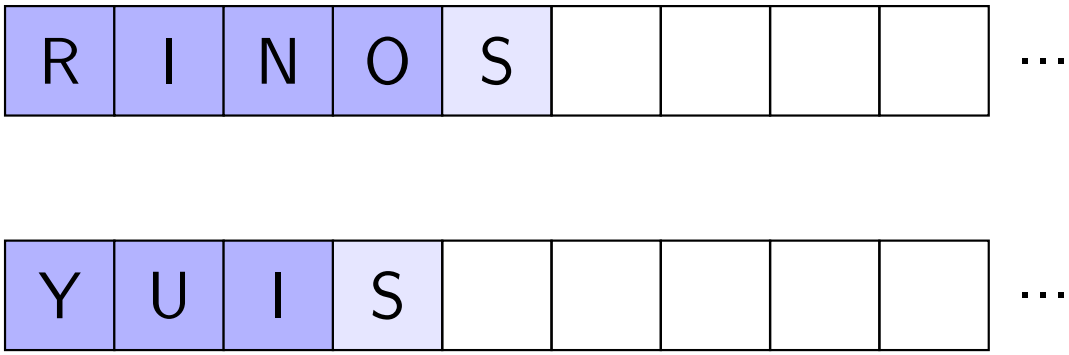
這兩個字串都有相同的 first 與 f
若是觸發了 “in event” 呢？
因為兩者在右邊的行為始終保持相同，所以觸發 “in event” 時的狀態也是一樣的，假設這狀態為 q
已知這兩個字串對任何 q 都有 $\mathsf{f}(s_1, q) = \mathsf{f}(s_2, q)$
以相同的 q 觸發 “in event” 之後，若
$\mathsf{f}(s_1, q) = \mathsf{f}(s_2, q) = q_{accept}/q_{reject}$
代表兩個字串在再一次發生 “out event” 之前就會確定都被接受，或確定都不被接受

Hw8 Problem3



這兩個字串都有相同的 first 與 f
若 $\mathrm{f}(s_1, q) = \mathrm{f}(s_2, q) \neq q_{accept}/q_{reject}$
代表會以同樣的狀態觸發 “out event”
這時兩個指針看到的右邊紙帶也會是相同內容 (儘管不一定是全白的)
所以兩個指針在右邊的行為也一定是一模一樣的
如果在這過程中其中一邊 accept 了，另外一邊也一定會 accept
若又觸發 “in event” 則以此類推

Hw8 Problem3



這兩個字串都有相同的 first 與 f
若一開始在輸入字串後面加上同樣的字串呢
以原本的位置作為 “in/out event” 的界線來看
第一次觸發 “out event” 時，兩個指針右邊依然是一樣的內容
(雖然多添加上去的格子無法編輯)
然後回到左邊的區域，會以一樣的狀態再次出去，右邊依然是一樣的內容.....
可以看出無論在後面加上任何字串 x，s_1x 與 s_2x 都會同時被接受或不被接受
所以這兩個字串是 “indistinguishable”

Hw8 Problem3

將 Σ^* 分成若干個集合，集合內的所有字串彼此之間 indistinguishable
可以拿「first 與 f 相同」來做區隔
對於一個字串 s
first(s) 的可能性有 |Q| 種
f(s, q_1) 的可能性有 |Q| 種
f(s, q_2) 的可能性有 |Q| 種
.....
f$(s, q_{|Q|})$ 的可能性有 |Q| 種
全部組合起來，總共有 $|Q|^{|Q|+1}$ 種可能性
所以可以分成有限多個集合
依照 Myhill-Nerode Theorem，證明這種圖靈機的效力只到正規語言

Hw8 Problem4

Decidable 與可以照順序 enumerate 的關係

首先，若有一個 decidable language，試著建構一台 Enumerator E 來照順序 enumerate 這個 language
因為 Σ^* 是可數集合，所以可以訂出每個字串被造訪的順序
因為給定的 language 是 decidable，所以會有一台對應的 Decider
構建一台 E，它會照順序拿 Σ^* 裡頭的字串並丟到這台 Decider
如果 accept 則將字串輸出，否則不做事
然後再拿下一個字串，反覆進行

Hw8 Problem4

若有一個照順序輸出的 Enumerator E，試著建構一個 Decider decides the enumerated language
構建一台 D，輸入字串 w
它會執行 E，並且查看 E 現在做到哪邊
當 E 輪到 w 時 (會在有限次內輪到)，如果 E 輸出 w 則 accept，否則 reject

得證

Hw8 Problem5

ALL_{DFA} 是 decidable

DFA 不會有 ϵ-transition，從每個狀態輸入任何字元也一定都有一條 transition
所以這個問題可以簡化為，會不會從起點走到一個不是 accepting state 的點
M = "On input $\langle A \rangle$, A is a DFA:
1. Mark the initial state
2. Mark the states that can be arrived from any marked states
3. Repeat step 2 until no state can be marked
4. If there is any non-accepting state marked, reject; otherwise, accept"
由於 DFA 的狀態數量是有限的，所以這個圖靈機一定會停機
於是可得到 ALL_{DFA} 是 decidable 的結論
另一種思路是直接用已知的結論，將 ALL_{DFA}reduce 成 E_{DFA} 或 EQ_{DFA}

Hw8 Problem6

$A = \{\langle R, S\rangle \mid R \text{ and } S \text{ are REs and } L(R) \subseteq L(S)\}$
M = "On input $\langle R, S\rangle$, R, S are REs:
1. Construct DFA F, G recognizing $L(R)$, $L(S)$ accordingly
2. Construct DFA G' recognizing $\overline{L(S)}$
3. Construct DFA H recognizing $L(R) \cap \overline{L(S)}$
4. Run $M_{E_{\text{DFA}}}$ on input $\langle H\rangle$
5. If $M_{E_{\text{DFA}}}$ accepts, accept; otherwise, reject."

這利用了一個性質，$S \subseteq S' \iff S \cap \overline{S'} = \emptyset$
上面所有的 construction 都是能在有限時間內完成的
可得出 A 是 decidable 的結論

Hw8 Problem7

$A_{\epsilon_{\text{CFG}}}$
如果想起 Chromsky Normal Form 就能讓這題變得簡單
M = "On input $\langle G \rangle$, G is a CFG:
1. Converse G into CNF G'
2. If $S_0 \to \epsilon$ in G', accept; otherwise, reject."

利用 CNF 的 CFG 只有起始變數可以有機會生成空字串這個性質
CNF 的轉換是可以在有限時間內完成的
所以 $A_{\epsilon_{\text{CFG}}}$ 是 decidable
當然也可以 reduce 成 A_{CFG} 問題，丟入 $\langle G, \epsilon \rangle$

Hw9 Problem1

A 是 Turing-recognizable language，包含了某些 Deciders
說明必然存在一個 decidable language D，它不能被 A 裡頭的任何 Decider 給 decide

Hw9 Problem1

學到對角論證法之後，當然要拿來用一用
至於怎麼用呢
題目提示告訴我們，既然 A 是 Turing-recognizable，就表示有一個 Enumerator E 可以生成 A
將 E 生成的第 i 個 TM 標記為 M_i
而因為 Σ^* 是可數集，存在一種排序法使對於任一個字串 $s \in \Sigma^*$ 而言，都能標記它出現的順序
於是可以做出一張表

Hw9 Problem1

	s_1	s_2	...	s_i
M_1	accept	accept	...	reject
M_2	accept	reject	...	accept
⋮	...	...	...	...
M_i	reject	accept	...	reject
⋮	...	...	...	...

依照這張表，建構一個 TM M_D recognize D

$M_D =$ "On input s:

1. 計算出 s 在 Σ^* 當中的順位 i
2. 將 s 丟入 M_i 當中計算
3. If M_i accepts, reject; otherwise, accept."

這樣就能建構出一台與 A 當中的任何圖靈機都不一樣的機器

而且 M_i 本身是 Decider，這台機器一定會停機，所以 M_D 是 Decider，D 是 decidable language

得證，存在一個 D 不能被 A 當中的任何 Decider 給判定

Hw9 Problem1

這題能告訴我們什麼
一個存著「所有」Deciders 的語言
$D_{ALL} = \{\langle D \rangle \mid \text{D decides a language over } \Sigma^*\}$ 不可能是 Turing-recognizable

Hw9 Problem2

存在一個 decider，可以判斷 CFG G 是否會生成某個字串 y 使得 x 是它的子字串

那麼就是要把 L(G) 與 $\Sigma^* x \Sigma^*$ 這兩個 language 取交集
一個 CFL 與 RL 的交集也是 CFL (將 PDA 與 DFA 的狀態合在一起做成新的 PDA)
再把交集出來的語言丟到 E_{CFG} 的 Decider 裡頭即可

Hw9 Problem2

M = "On input $\langle G, x \rangle$ where G is a CFG:
1. Construct a CFG G' s.t. $L(G') = L(G) \cap \Sigma^* x \Sigma^*$
2. Run $M_{E_{CFG}}$ on input $\langle G' \rangle$
3. If $M_{E_{CFG}}$ accept, reject; otherwise, accept."

上面每個步驟都能在有限時間完成，所以得 C 是 decidable

Hw9 Problem3

如何偵測 PDA 裡頭的 useless state
將所有狀態都變成 nonaccepting，然後再單獨把一個 state 畫成 accepting
如果此時 PDA recognize 的語言為空，則代表這個 state 完全不會被抵達

Hw9 Problem3

M = "On input $\langle P \rangle$, P is a PDA:
1. 將 PDA 所有狀態都變成 nonaccepting
2. Choose one state to be accepting
3. Convert this PDA into CFG G
4. Run $M_{E_{CFG}}$ on input $\langle G \rangle$
5. Repeat step 2 to 4
6. If $M_{E_{CFG}}$ has ever accepted, accept; otherwise, reject."

Hw9 Problem4

判定 PDA 辨識的字串是否有無限多個
由 pumping lemma 可以知道，只要 CFL 內有個字串 s 長度有 pumping length p 以上，就可以生成無限多個字串也在 CFL 內
而且此時一定會有一個長度介於 p 與 2p 之間的字串：
如果 $p \leq |s| \leq 2p$ 那就有了
如果 $|s| > 2p$，將 s 分成 $wvxyz$，因為 $0 < |vxy| \leq p$，wxz 的長度一定比 s 短，且不會短到低於 p

Hw9 Problem4

M = "On input $\langle M \rangle$ where M is a PDA:
1. Convert M into CFG G
2. Take p = $b^{|V|+1}$, where b is the maximum length of productions in G, and |V| is the number of variables of G
3. Run $M_{A_{CFG}}$ on input $\langle G, s \rangle$ for some s with $p \leq |s| \leq 2p$
4. Repeat step 3 with all s satisfying $p \leq |s| \leq 2p$, if $M_{A_{CFG}}$ accepts for some s, accept; otherwise, reject."

Hw9 Problem5

EQ_{CFG} is undecidable
這一題在已知 ALL_{CFG} 是 undecidable 的前提下就很好解

如果 EQ_{CFG} 是 decidable
那麼只要做出一個生成 Σ^* 的 CFG G
檢查其他 CFG 與 G 在不在 EQ_{CFG} 裡頭
就能判定這個 CFG 是否能生成 Σ^*
但已知 ALL_{CFG} 是不可判定語言，矛盾

Hw9 Problem6

若 A 能 reduce 成一個正規語言 B，那 A 是不是正規的呢？

假設 A 是一個 CFL，而 B $= \{1\}$
試著找到 f 使得 $w \in A \iff f(w) \in B$
假設 A 對應的 CFG 為 G
F $=$ "On input w:
1. Run $M_{A_{CFG}}$ on input $\langle G, w \rangle$
2. If $M_{A_{CFG}}$ accepts, output 1; otherwise, output 0"
因為 A_{CFG} 是 decidable，所以 f 是 computable
如此可知，雖然 A 可以 reduce 成正規語言 B，但 A 並不見得是正規的

Hw10 Problem1

試著說明 $AMBIG_{\text{CFG}}$ 是 undecidable
題目提示得很清楚了，只需要照著寫，建構出假想中可以判定 PCP 問題的圖靈機即可證明
M = "On input P:
1. Construct CFG G with the rules:

$$S \to T \mid B$$

$$T \to t_1 T a_1 \mid ... \mid t_k T a_k \mid t_1 a_1 \mid ... \mid t_k a_k$$

$$B \to b_1 T a_1 \mid ... \mid b_k T a_k \mid b_1 a_1 \mid ... \mid b_k a_k$$

2. Run $M_{AMBIG_{\text{CFG}}}$ on input G
3. If $M_{AMBIG_{\text{CFG}}}$ accepts, accept; otherwise, reject."

如果 $AMBIG_{\text{CFG}}$ 是 decidable，那這台自動機就可以 decide PCP 問題，但已知 PCP 問題不可判定，所以 $AMBIG_{\text{CFG}}$ 是 undecidable

Hw10 Problem1

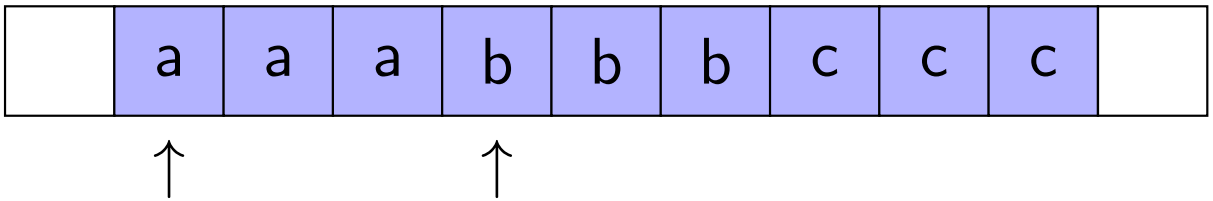
至於為什麼這個 CFG 可以拿來輔助 PCP 問題
一開始 S 就分成兩條路走，T 與 B 兩條路
兩個變數每分裂一次都會產生若干字元 t_i 或 b_i，並且在右邊加上 a_i
a_i 是獨立的標示符號，用途就是標註生成的是 t 幾或 b 幾
如果 ambiguous 發生了，就一定是存在一個 a 序列 $a_i a_{i-1}...a_1$，使得 T 路與 B 路產生了 $t_1...t_i a_i...a_1 = b_1...b_i a_i...a_1$，也就是 $t_1...t_i = b_1...b_i$
如果沒有 ambiguity，就代表對所有的 a 序列，a 序列左邊的 $t_1...t_i \neq b_1...b_i$

Hw10 Problem2

證明有兩個可雙向移動但無法寫入的指針，紙帶空間局限於輸入區與左右各一個空格的機器 2DFA，它的 emptiness problem E_{2DFA} 是不可判定的

首先先理解一下為什麼它可以 recognize $\{a^n b^n c^n \mid n \geq 0\}$

Hw10 Problem2



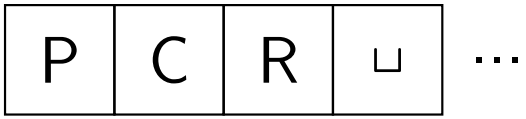
將指針向右跑就能判斷 a 的數量是否與 b 的數量相等
然後再偵測 b 的數量是否與 c 相等
最後左邊的指針跑到 c 區且右邊的指針跑到空格就 accept

Hw10 Problem2

那麼回到正題，這種機器的 emptiness 問題
我們試著將 A_{TM} 問題 reduce 成 E_{2DFA}
利用圖靈機的 configuration：
建構一台 2DFA，接受某個 TM 輸入一個字串的合法 configuration history，而且是 accept 的 history

Hw10 Problem2

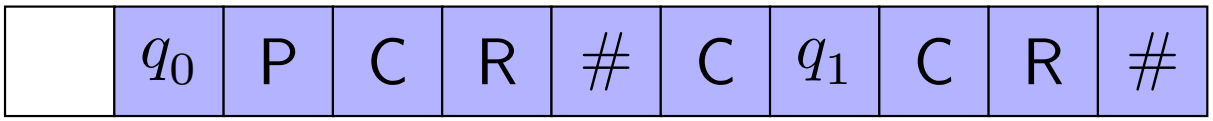
考慮一台圖靈機與其輸入字串



如果這台圖靈機會接受這個字串，那就會存在一個合法的 accepting computation history；如果不接受，那就不存在
所以我們的目標就是要建構一個接受這台機器與這個輸入字串的 accepting computation history 的 2DFA
如果這個 2DFA 的語言不是空的，就代表存在一個合法的 accepting computation history，也就是圖靈機接受這個字串；反之就是找不到這樣的 history，也就是圖靈機不接受它

Hw10 Problem2

為了讓大家更明白這台 2DFA 的行為，所以下面稍微簡短解釋

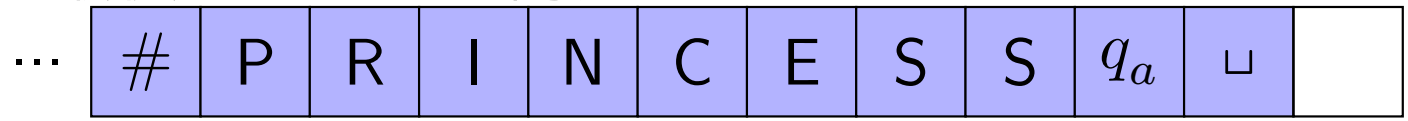


首先檢查第一個 configuration 是不是合法的，這台圖靈機輸入這個字串的 configuration
接下來用左右兩個指針檢查相鄰的 configuration 是否是這台圖靈機的正確行為
比如在這個例子當中，左指針先遇到 state，代表發生一個 R transition，這時檢查 $\delta(q_0, P) = q_1, C, R$ (註：這個 R 是方向的 R) 是否成立即可

為求簡化，可以規定兩個相鄰的 configuration 長度只有當指針走到尾端且向右的時候會往右擴充出一個空格，其餘時候相鄰的 configuration 需要有相同長度.....

Hw10 Problem2

到最後碰上 2DFA 的尾巴時



檢查這個 configuration 是否停在 accepting state

Hw10 Problem2

以上說明是為了幫助大家更了解細節
而理解了細節之後就可以寫下答案了
N = "On input $\langle M, w \rangle$ where M is a TM and w is a word:
1. Construct 2DFA D from $\langle M, w \rangle$ as described in previous slides
2. Run $M_{E_{\text{2DFA}}}$ on input $\langle D \rangle$
3. If $M_{E_{\text{2DFA}}}$ accepts, accept; otherwise, reject."
如果 E_{2DFA} 是 decidable，那我們就做出一台 A_{TM} 的 Decider 了
但 A_{TM} 是 undecidable，所以 E_{2DFA} 也是 undecidable

Hw10 Problem3

要運用 Rice's Theorem 來解這題之前，要先確認好前提
首先 $\{\langle M \rangle \mid M \text{ is a TM and } 101 \in L(M)\}$ 是要判斷一個圖靈機它辨識的語言是否包含 "101"
討論的範圍是圖靈機，而且我們只在乎圖靈機辨識的語言包不包含 101，而不在乎圖靈機的實現本身
所以「包不包含 101」是一個圖靈機辨識語言的 property
其次，「包不包含 101」並不是 trivial property，因為一定存在一些圖靈機辨識的語言包含 101，另一些不包含 101
所以直接利用 Rice's Theorem，得證
$\{\langle M \rangle \mid M \text{ is a TM and } 101 \in L(M)\}$ 是 undecidable

Hw10 Problem4

給定一個不能更動輸入區的圖靈機，以及一個輸入字串
這台圖靈機會不會接受這個字串？這個問題是否是 decidable
在作業 8-3 曾經探討過這種圖靈機的辨識能力只限於正規語言
所以看起來只需要把這台圖靈機所辨識的正規語言寫出來，轉為 DFA 就可以囉？

問題在於輸入區右方的地帶實際上擁有正常圖靈機的機能
當初證明時並沒有假設指針一定會從右方回到左方，我們是用「假如指針會回到左方則...」的概念去操作的
實際上對於某些輸入，指針可能一去不回頭，我們無法得知這個字到底是不是在它所辨識的 language 裡頭

Hw10 Problem4

更嚴謹地，我們要試著把某個 undecidable 問題 reduce 成 X
假設辨識 X 的圖靈機叫做 M_X
試著把 A_{TM} reduce 到 X
M = "On input $\langle M, w \rangle$, M is a TM and w is a string:
1. Construct M': "On input u:
   1. Move to the right of u and put $
   2. Write w after $
   3. Simulate M on the w part
   4. If M accepts, accept; otherwise, reject."
2. Run M_X on input $\langle M', u \rangle$ for arbitrary u
3. If M_X accepts, accept; otherwise, reject."

Hw10 Problem4

建構一台無論輸入什麼都只會去右邊模擬 M 輸入 w 的圖靈機 M'，M' 不會更改輸入區的內容，因為它都在輸入區右方的 $ 右方作事
所以 M' 是一台符合條件的圖靈機
也可以觀察到這台 M' 可能辨識的語言只有 Σ^* 與 $\emptyset$ 兩種可能，這兩個語言都是正規語言，與之前的結論是對得上的
不過這題的結論是，X 並不是 decidable
這其實就是說，儘管知道它辨識正規語言，但在有些狀況我們無法從某台圖靈機做出對應的 DFA，因為這台圖靈機可能不會停機

Hw10 Problem5

$USELESS_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM having any useless state}\}$
試著證明這個語言是 undecidable

與 $USELESS_{\text{PDA}}$ 不一樣，在 PDA 的場合我們可以檢查辨識的語言是否為空，也就是 E_{CFG} 是可判定的
但 E_{TM} 不可判定，所以這招不管用

Hw10 Problem5

和上題類似，建構一台無論輸入什麼都只會模擬 M 輸入 w 的圖靈機，將它與 undecidable 問題 (A_{TM}) 連結在一起

M = "On input $\langle M, w \rangle$, M is a TM and w is a string:

1. Construct M': "On input u:
   1. Run M on input w. There is a state never triggered in this step
   2. If M accepts w, traverse all states except q_{accept} and q_{reject}
   3. If $u = \epsilon$, reject; otherwise, accept."
2. Run $M_{USELESS_{\text{TM}}}$ on input $\langle M' \rangle$
3. If $M_{USELESS_{\text{TM}}}$ accepts, reject; otherwise, accept."

Hw10 Problem5

細節說明，建構 M' 的過程中，在建構好模擬 M 的部分之後，加上一個獨立的 state，再加上僅限第二步使用的 transition
在 M accepts w 的當下在紙帶寫入特殊符號，指針待在原地，並從第一個狀態開始
從第一個狀態遍歷所有狀態 (除了停機狀態)，過程都是讀取特殊符號、指針不動
最後會分成兩種停機狀況，使得 q_{accept} 與 q_{reject} 都會被用到
於是，M 若接受 w，M' 的所有 state 都有機會被拜訪到；若 M 不接受 w，則會有至少一個我們故意放的 state 永遠不會被拜訪

得證，$USELESS_{\mathsf{TM}}$ 是 undecidable

Hw10 Problem6

試圖找尋一個 undecidable 語言，是 $\{\epsilon, 1, 11, ...\}$ 的子集

這個集合內的元素之間最大的差別是 1 的數量
那麼就從 1 的數量著手

Hw10 Problem6

隨便找一個利用某種 encoding 存放圖靈機，有關圖靈機的 undecidable language
比如說 A_{TM} 或 E_{TM}
這些語言都是透過某種 encoding 存放物件的
這些 encoding 的 alphabet Σ 都是有限的
也就是說這些物件的編碼 $e \in \Sigma^*$
已知 Σ^* 是可數無窮集合，我們可以找到 e 在 Σ^* 的順序 i
將這個順序號碼 i 解釋為 1 的數量就行了

Hw10 Problem6

F = "On input $\langle M \rangle$:
1. Compute the index i of $\langle M \rangle$ in Σ^*
2. Output 1^i"
這樣就做出了一個 computable function，將一台圖靈機 map 到 1^i
更詳細地說，下列語言：

$$I = \{1^i \mid \langle M \rangle \in E_{\mathsf{TM}} \wedge F(\langle M \rangle) = i\}$$

滿足 $I \subseteq \{1\}^*$ 以及 $E_{\mathsf{TM}} \leq_m I$
已知 E_{TM} 是 undecidable，I 也會是 undecidable