第 5 組 系統程式-期末報告

分工

1112914 宋冠穎 (25 %)

- Regular Expression -> Dotted Item -> NFA
- NFA -> DFA
- 使用 LaTex 將 Dotted Item 手寫過程整理成數位檔案
- Demo
- 簡報製作

1112919 丁柏凱 (25 %)

- NFA -> DFA
- 使用 LaTex 將 NFA -> DFA 手寫過程整理成數位檔案
- Python Program
- Demo
- inkscape 繪圖
- 簡報製作

1112929 詹佳瑜 (10 %)

- 想測資
- 檢查輸入 & 輸出

1112946 王嘉汶 (18 %)

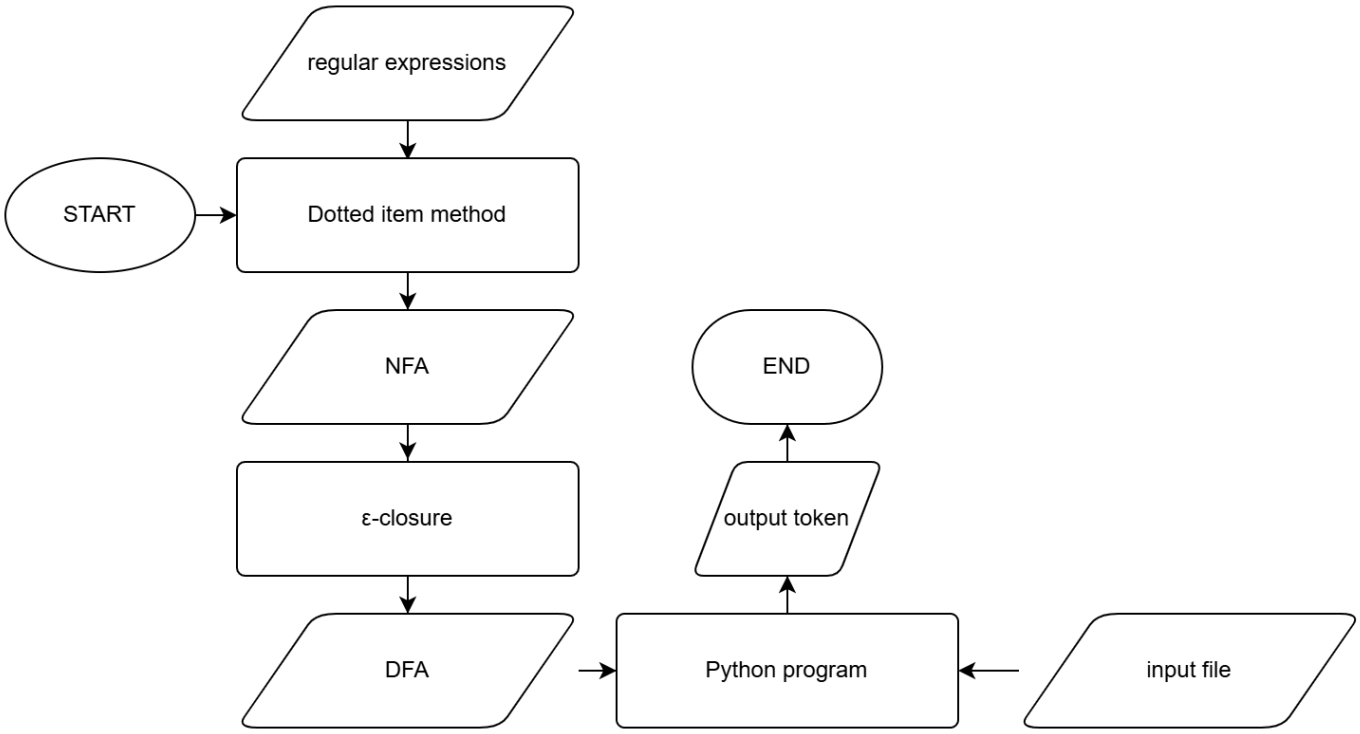
- inkscape 繪圖
- Demo
- 簡報製作

1112957 顏琦恩 (22 %)

- Regular Expression -> Dotted Item -> NFA
- NFA -> DFA

- 簡報製作
- 影片剪輯

題目 & 流程圖



1. 利用 dotted item method 把以下的 regular expressions 轉 NFA/DFA

$$\begin{aligned}
&\textbf{underscore} \rightarrow _ \\
&\textbf{letter} \rightarrow \mathrm{A} \mid \mathrm{B} \ldots \mid \mathrm{Z} \mid \mathrm{a} \mid \mathrm{b} \ldots \mid \mathrm{z} \\
&\textbf{digit} \rightarrow 0 \mid 1 \mid 2 \ldots \mid 9 \\
&\textbf{underscore_tail} \rightarrow \textbf{underscore}\ (\textbf{letter} \mid \textbf{digit})^{+} \\
&\textbf{id} \rightarrow \textbf{letter}\ (\textbf{letter} \mid \textbf{digit})^{*}\ \textbf{underscore_tail}^{*} \\
&\textbf{fraction} \rightarrow \cdot\ \textbf{digit}^{+} \\
&\textbf{optional_exponent} \rightarrow (\mathrm{E}\ (+ \mid - \mid \varepsilon)\ \textbf{digit}^{+}\ ) \mid \varepsilon \\
&\textbf{integer} \rightarrow \textbf{digit}^{+} \\
&\textbf{float} \rightarrow \textbf{digit}^{*}\ \textbf{fraction}\ \textbf{optional_exponent}
\end{aligned}$$

2. 寫一個 lexical analyzer 來分析讀到的是哪個 token (Separators and Brackets, Binary Operators, Keywords, Comments, Identity, Integer and Float)

Separators and Brackets: the language uses ';', '(', ')', '[', ']', '{', '}' as the separators or brackets.
Binary Operators: the language uses the following operators +, -, *, /, <=, >=, <>, <, >, =, ==.
Keywords: the language uses the following keywords int, float, bool, void, while, if, else, for, return.
Comments: the language supports C style comments. A comment starts with /* and end with */ with anything in between except a comment ending sequence (*/).
Layout: the language uses the following layout symbols: any number of blank, tabulation, carriage return and line feed.

注意: 不能用 lex 或 similar lexical analyzer

Regular Expression -> NFA

將 Regular Expression 利用 dotted item method 轉換成 NFA

NFA:

- 可以對同一個輸入字元有多個轉移方向
- 可以存在 ε-transition（空轉移，不需輸入符號即可移動)

dotted item method

- 定義
  - dotted item 是一個帶有「dot ．」的產生式（production），標記目前已經分析到哪個位置。點號位於產生式右側的某個位置，表示目前已經「看到」這個位置前的所有符號，下一步將會分析點號後的符號。
- 範例
  - 一個正規表達式

    `A → XYZ`
  - 會有以下 dotted items
    a. `A → · XYZ` ：表示尚未處理任何符號
    b. `A → X · YZ` ：表示已處理了 X，下一步要處理 Y
    c. `A → XY · Z` ：表示已處理了 X 和 Y
    d. `A → XYZ ·` ：表示整個產生式已處理完
- Character moves
  - 基本項目（Basic item）
    - 如果項目中的點（dot）位於基本項目前方，則在輸入字元與基本項目相符時，該項目會將點移動到下一個字元位置。
  - 非基本項目（Non-basic item）

    - 項目中的點無法直接跨過一個字元，因為它沒有定義一組可比對的字元集合（無法判斷輸入字元是否符合）。
    - 如 $*$ 、$+$ 、$?$ 、$|$

dotted item 過程為簡化狀態機有進行部分簡化

- 一般 dotted item 中 Non-basic item $\varepsilon - move$ 規則

$$T \to \alpha \bullet (R)^* \beta \quad \Rightarrow \quad T \to \alpha (R)^* \bullet \beta$$
$$T \to \alpha (\bullet R)^* \beta$$

$$T \to \alpha (R \bullet)^* \beta \quad \Rightarrow \quad T \to \alpha (R)^* \bullet \beta$$
$$T \to \alpha (\bullet R)^* \beta$$

$$T \to \alpha \bullet (R)^+ \beta \quad \Rightarrow \quad T \to \alpha (\bullet R)^+ \beta$$

$$T \to \alpha (R \bullet)^+ \beta \quad \Rightarrow \quad T \to \alpha (R)^+ \bullet \beta$$
$$T \to \alpha (\bullet R)^+ \beta$$

- 簡化範例: (2) 讀完 letter 後，移到 (4) 表示括號內容讀取完畢，與移到 (1) 表示再次讀取括號內容。

$$⓪ \Rightarrow ① \ underscore\_tail \to underscore \bullet (letter \mid digit)^+$$

$$① \Rightarrow ② \ underscore\_tail \to underscore(\bullet letter \mid digit)^+$$
$$① \Rightarrow ③ \ underscore\_tail \to underscore(letter \mid \bullet digit)^+$$

$$② \Rightarrow ④ \ underscore\_tail \to underscore(letter \mid digit)^+ \bullet <<<< recognized$$
$$② \Rightarrow ① \ underscore\_tail \to underscore \bullet (letter \mid digit)^+$$

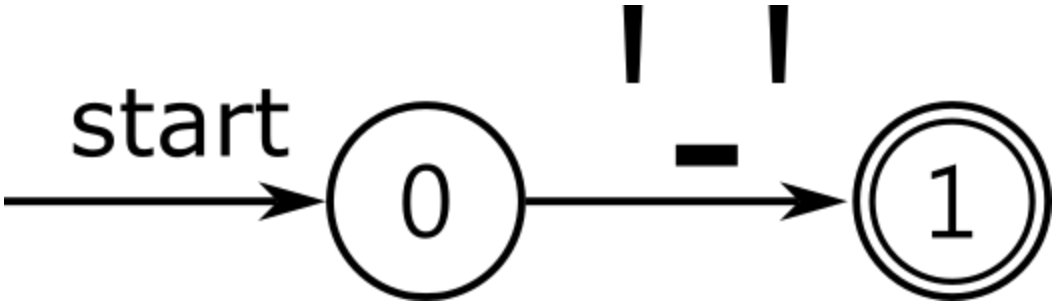
1. underscore

underscore有可能為
⓪:接下來要匹配' _ '
①:已經完成匹配' _ '

$$underscore \rightarrow \_$$

⓪ $underscore \rightarrow \bullet\_$

① $underscore \rightarrow \_\bullet$ $<<<<$ $recognized$



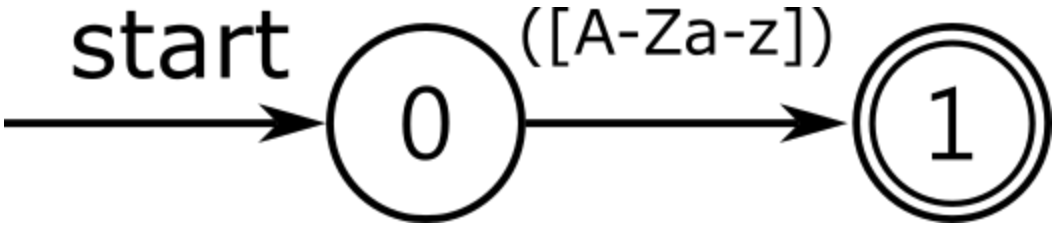
2. letter

letter有可能為
⓪接下來要匹配[A-Za-z]
①已經完成匹配[A-Za-z]

$$letter \rightarrow [A-Za-z]$$

⓪ $letter \rightarrow \bullet[A-Za-z]$

⓪ $\Rightarrow$① $letter \rightarrow [A-Za-z]\bullet$

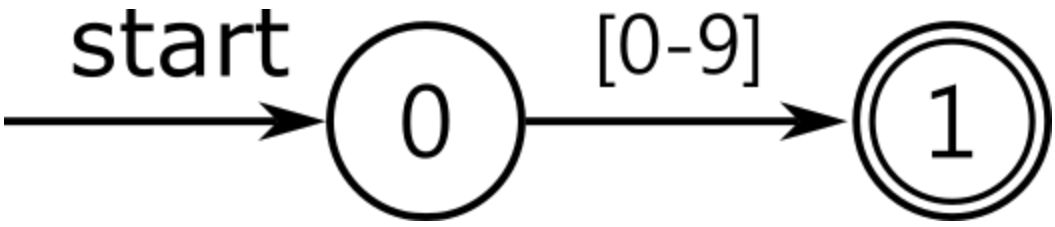


3. digit

$$digit \rightarrow [0-9]$$

⓪ $digit \rightarrow \bullet[0-9]$

⓪ ⇒① $digit \rightarrow [0-9]\bullet$ <<<< $recognized$



4. underscore_tail

$$underscore\_tail \rightarrow underscore(letter \mid digit)^{+}$$

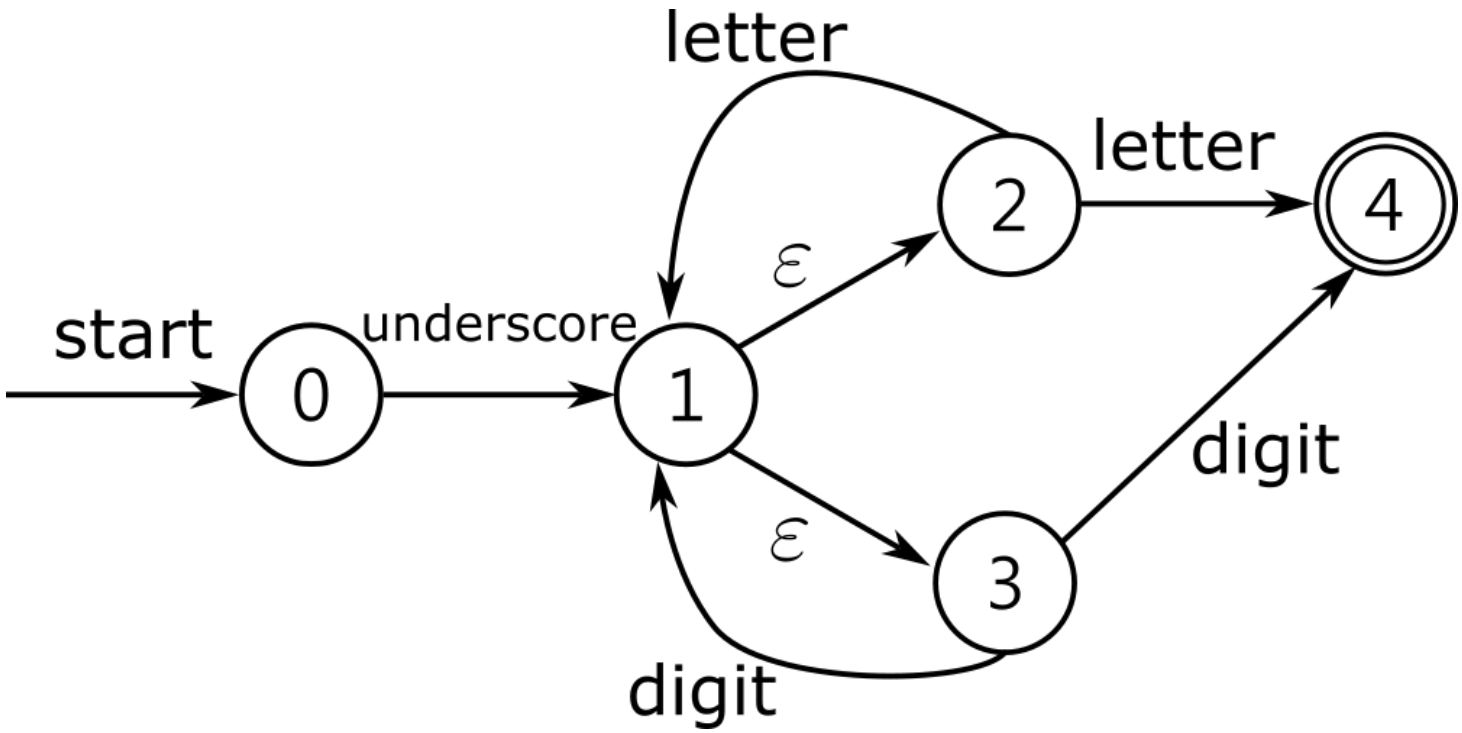
⓪ $underscore\_tail \rightarrow \bullet underscore(letter \mid digit)^{+}$

⓪ ⇒① $underscore\_tail \rightarrow underscore \bullet (letter \mid digit)^{+}$

① ⇒② $underscore\_tail \rightarrow underscore(\bullet letter \mid digit)^{+}$
① ⇒③ $underscore\_tail \rightarrow underscore(letter \mid \bullet digit)^{+}$

② ⇒④ $underscore\_tail \rightarrow underscore(letter \mid digit)^{+}\bullet$ <<<< $recognized$
② ⇒① $underscore\_tail \rightarrow underscore \bullet (letter \mid digit)^{+}$

③ ⇒④ $underscore\_tail \rightarrow underscore(letter \mid digit)^{+}\bullet$ <<<< $recognized$
③ ⇒① $underscore\_tail \rightarrow underscore \bullet (letter \mid digit)^{+}$


letter
letter
2
4
ε
start
underscore
0
1
digit
ε
3
digit

5. id

$$id \rightarrow letter(letter \mid digit)^*(underscore\_tail)$$

⓪ $id \rightarrow \bullet letter(letter \mid digit)^*(underscore\_tail)$

⓪ ⇒① $id \rightarrow letter \bullet (letter \mid digit)^*(underscore\_tail)$

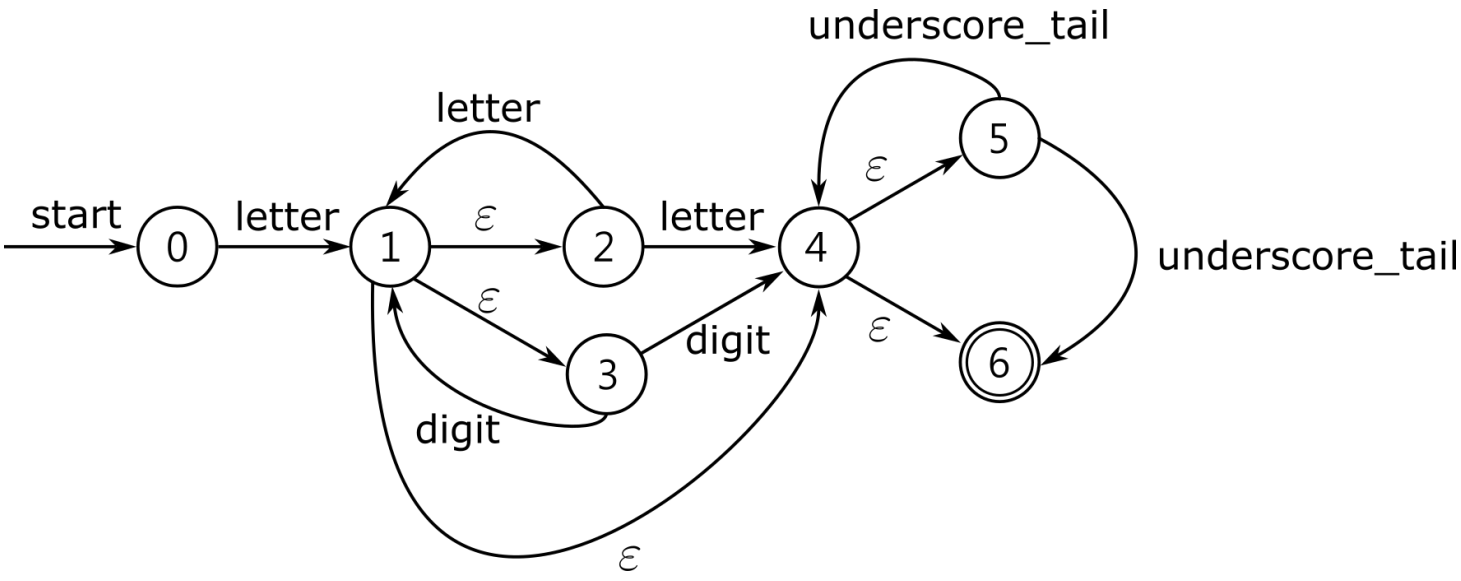
① ⇒② $id \rightarrow letter(\bullet letter \mid digit)^*(underscore\_tail)$
① ⇒③ $id \rightarrow letter(letter \mid \bullet digit)^*(underscore\_tail)$
① ⇒④ $id \rightarrow letter(letter \mid digit)^* \bullet (underscore\_tail)$

② ⇒④ $id \rightarrow letter(letter \mid digit)^* \bullet (underscore\_tail)$
② ⇒① $id \rightarrow letter \bullet (letter \mid digit)^*(underscore\_tail)$

③ ⇒④ $id \rightarrow letter(letter \mid digit)^* \bullet (underscore\_tail)$
③ ⇒① $id \rightarrow letter \bullet (letter \mid digit)^*(underscore\_tail)$

④ ⇒⑤ $id \rightarrow letter(letter \mid digit)^*(\bullet underscore\_tail)$
④ ⇒⑥ $id \rightarrow letter(letter \mid digit)^*(underscore\_tail)\bullet$ <<<< $recognized$

⑤ ⇒⑥ $id \rightarrow letter(letter \mid digit)^*(underscore\_tail)\bullet$ <<<< $recognized$
⑤ ⇒④ $id \rightarrow letter(letter \mid digit)^* \bullet (underscore\_tail)$



6. fraction

$$fraction \rightarrow .(digit)^+$$

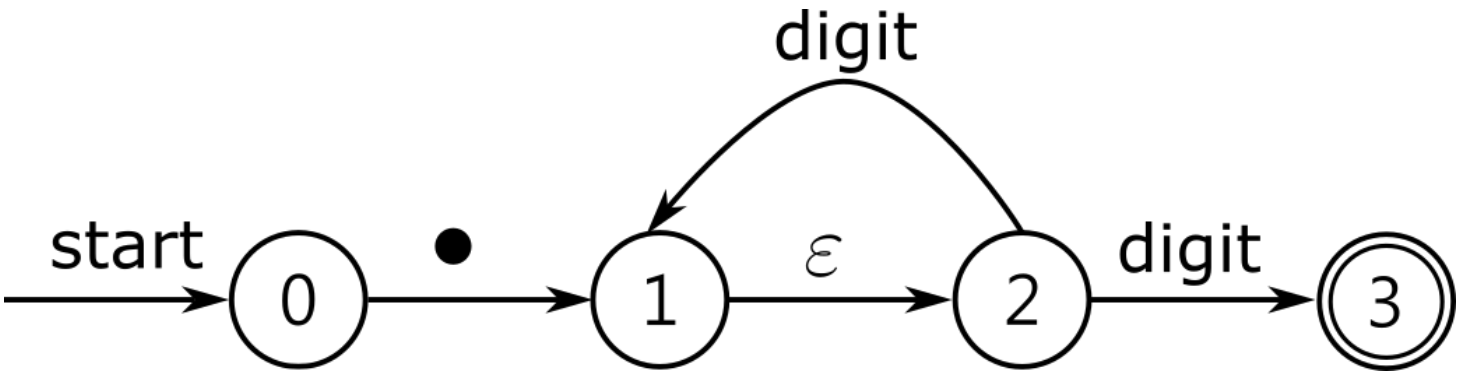
⓪ $fraction \rightarrow \bullet.(digit)^+$

⓪ ⇒① $fraction \rightarrow . \bullet (digit)^+$

① ⇒② $fraction \rightarrow .(\bullet digit)^+$

② ⇒① $fraction \rightarrow . \bullet (digit)^+$

② ⇒③ $fraction \rightarrow .(digit)^+ \bullet$ <<<< recognized



7. optional_exponent

$$optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon$$

⓪ $optional\_exponent \rightarrow \bullet(E(+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon$

⓪ ⇒① $optional\_exponent \rightarrow (\bullet E(+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon$
⓪ ⇒② $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon)(digit)^+) \mid \bullet\varepsilon$

② ⇒③ $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon\bullet <<<< recognized$

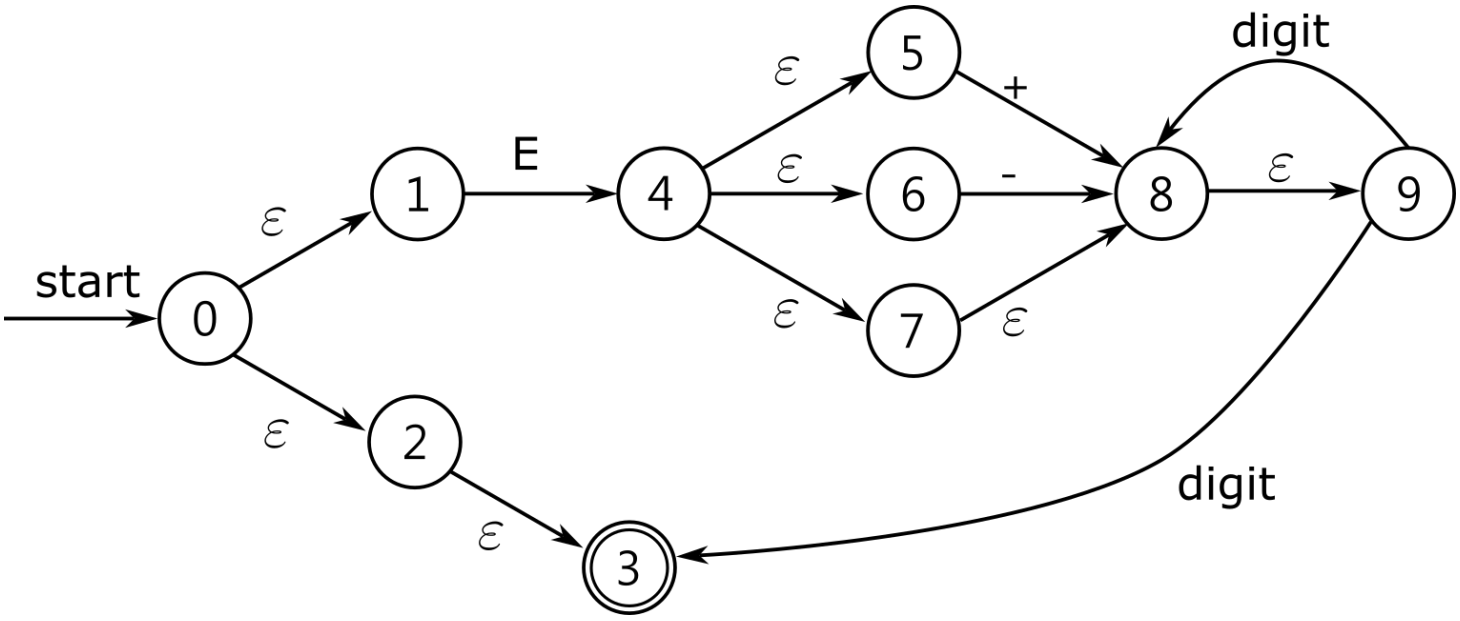
① ⇒④ $optional\_exponent \rightarrow (E \bullet (+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon$

④ ⇒⑤ $optional\_exponent \rightarrow (E(\bullet+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon$
④ ⇒⑥ $optional\_exponent \rightarrow (E(+ \mid \bullet- \mid \varepsilon)(digit)^+) \mid \varepsilon$
④ ⇒⑦ $optional\_exponent \rightarrow (E(+ \mid - \mid \bullet\varepsilon)(digit)^+) \mid \varepsilon$

⑤⑥⑦ ⇒⑧ $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon) \bullet (digit)^+) \mid \varepsilon$

⑧ ⇒⑨ $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon)(\bullet digit)^+) \mid \varepsilon$

⑨ ⇒⑧ $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon) \bullet (digit)^+) \mid \varepsilon$
⑨ ⇒③ $optional\_exponent \rightarrow (E(+ \mid - \mid \varepsilon)(digit)^+) \mid \varepsilon\bullet <<<< recognized$



8. integer

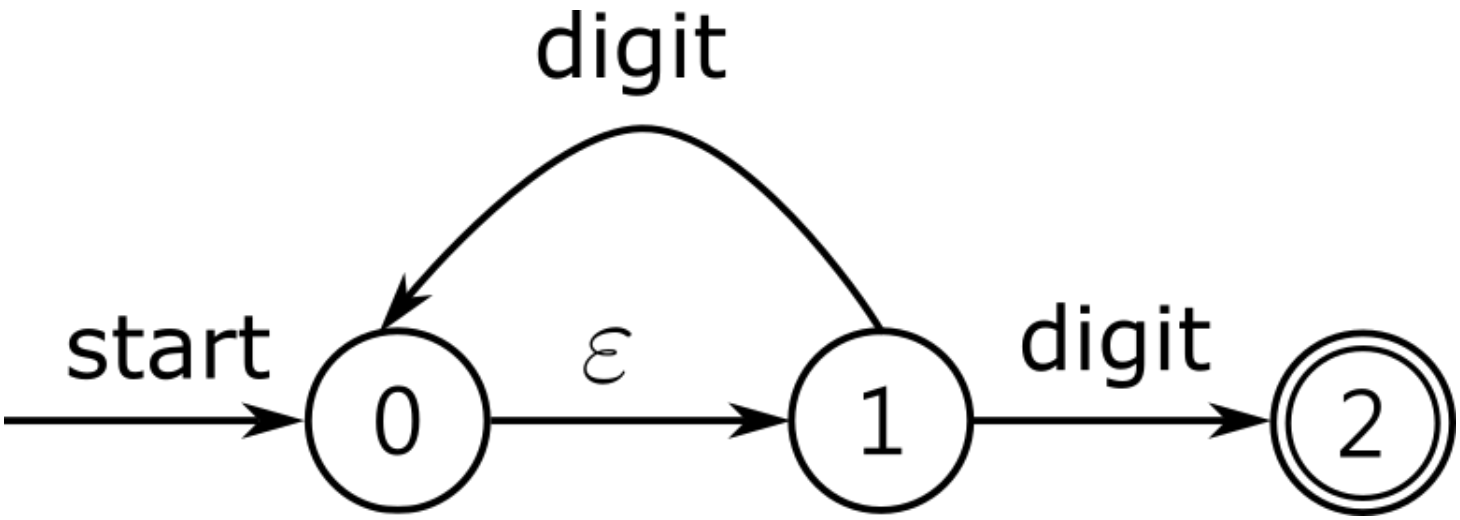
$$integer \rightarrow (digit)^+$$

⓪ $integer \rightarrow \bullet(digit)^+$

⓪ ⇒① $integer \rightarrow (\bullet digit)^+$

① ⇒⓪ $integer \rightarrow \bullet(digit)^+$

① ⇒② $integer \rightarrow (digit)^+\bullet$ $<<<<$ $recognized$



9. float

$$float \rightarrow (digit)^* \ fraction \ optional\_exponent$$

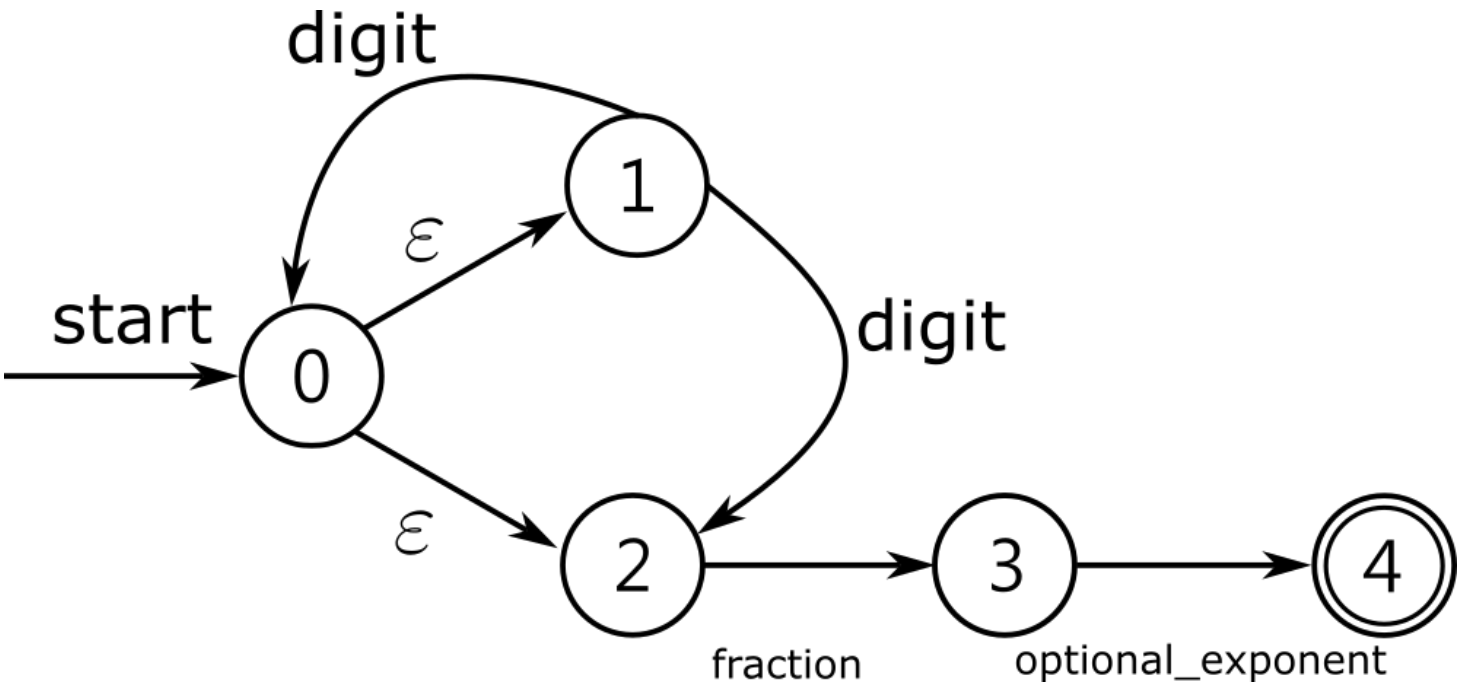
⓪ $float \rightarrow \bullet(digit)^* \ fraction \ optional\_exponent$

⓪ ⇒① $float \rightarrow (\bullet digit)^* \ fraction \ optional\_exponent$
⓪ ⇒② $float \rightarrow (digit)^* \bullet \ fraction \ optional\_exponent$

① ⇒② $float \rightarrow (digit)^* \bullet \ fraction \ optional\_exponent$
① ⇒⓪ $float \rightarrow \bullet(digit)^* \ fraction \ optional\_exponent$

② ⇒③ $float \rightarrow (digit)^* \ fraction \ \bullet optional\_exponent$

③ ⇒④ $float \rightarrow (digit)^* \ fraction \ optional\_exponent \bullet <<<< recognized$



NFA -> DFA

DFA

- 在任一狀態下，對於一個輸入字元，最多只能有一條對應的邊
- 不能對同一個輸入字元有兩種選擇

ε-closure

- ε-closure(S)： 從S這個狀態出發，能沿著ε邊一直走下去的所有狀態就是ε-closure(S)
- ε-closure(T)： T可以定義多個狀態來出發，就是可以不只closure一個狀態
- move(S, 字元)： 從T中的所有狀態出發，根據輸入字元，所能轉移到的所有狀態

步驟

- 首先Initializations： 一開始只放一個初始狀態，move後並設為 A

$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

- 從初始狀態開始move(S, 字元)，表示從S中的所有狀態出發，根據輸入字元，所能轉移到的所有狀態
- 再ε-closure(S)，也就是將ε消掉，如果有新集合，設定為新狀態(B、C、D...)，重複下去直到沒有新狀態。

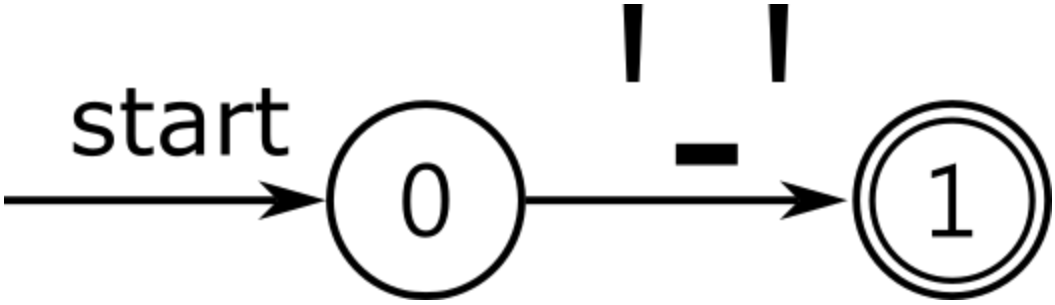
$$A = \{0, 1, 2, 4, 7\} \qquad D = \{1, 2, 4, 5, 6, 7, 9\}$$
$$B = \{1, 2, 3, 4, 6, 7, 8\} \qquad E = \{1, 2, 4, 5, 6, 7, 10\}$$
$$C = \{1, 2, 4, 5, 6, 7\}$$

- 再將每個 states 化成 transition diagram

STATE	INPUT SYMBOL	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

我們針對剛剛產生的 9 個 NFA 圖形，來產生 DFA：

1. underscore



首先他的初始狀態是0，對她closure後因為沒有ε，所以一樣是1的集合，然後把它設為A狀態
再來要對所有可能輸入的字元做move，那這邊所有可能讀到的只有底線，所以move(A,_)開始找可能轉移到的所有狀態，那他會到的只有1，
接下來對1，ε-closure，那他一樣因為沒有ε的邊，所以一樣只有1，且為新狀態所以設成B
再從狀態B重複以上步驟直到沒有新狀態，畫出轉移表

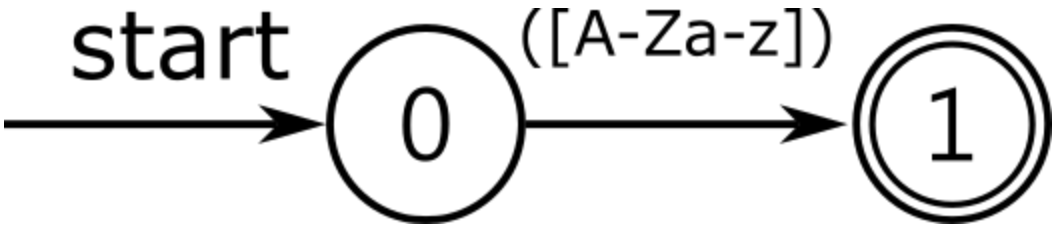
$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, \_) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1\} \Longrightarrow B$$

$$move(B, \_) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

No	State	_
0	A	B
1	B	-

2. letter



以此類推underscore

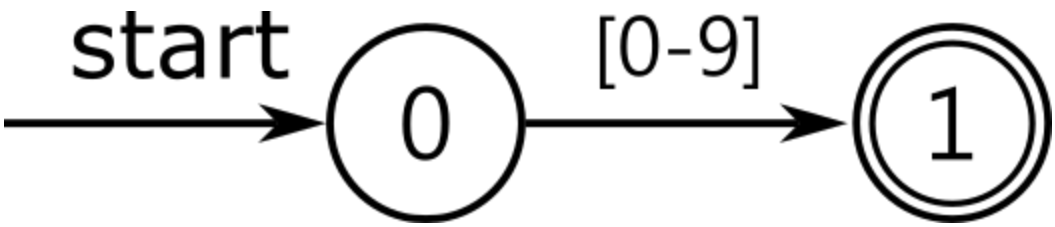
$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, [A - Za - z]) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1\} \Longrightarrow B$$

$$move(B, [A - Za - z]) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

No	State	[A-Za-z]
0	A	B
1	B	-

3. digit



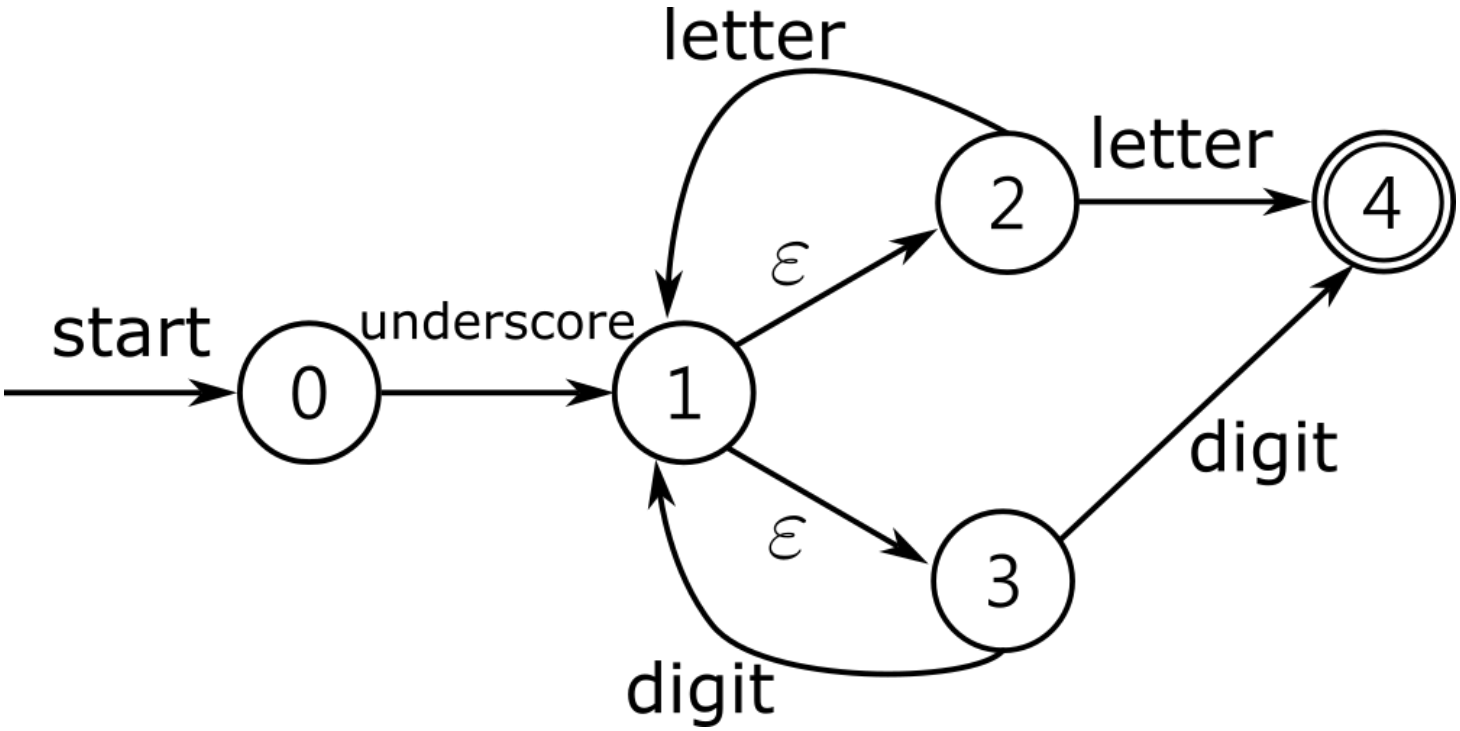
$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, [0-9]) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1\} \Longrightarrow B$$

$$move(B, [0-9]) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

No	State	[0-9]
0	A	B
1	B	-

4. underscore_tail



$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, underscore) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1, 2, 3\} \Longrightarrow B$$

$$move(A, letter) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, underscore) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, letter) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4\} \Longrightarrow C$$

$$move(B, digit) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4\} \Longrightarrow C$$

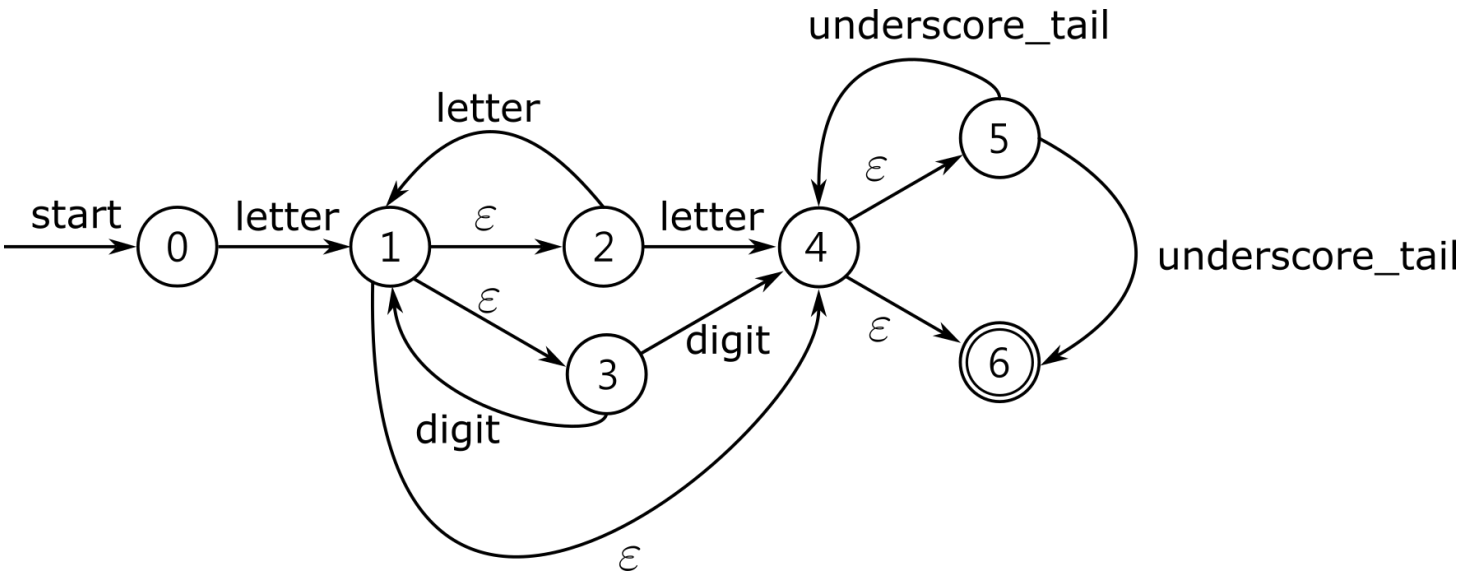
$$move(C, underscore) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, letter) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4\} \Longrightarrow C$$

$$move(C, digit) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4\} \Longrightarrow C$$

No	State	underscore	letter	digit
0	A	B	-	-
1	B	-	C	C
2	C	-	C	C

5. id



$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, letter) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1, 2, 3, 4, 5, 6\} \Longrightarrow B$$

$$move(A, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, underscore\_tail) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, letter) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4, 5, 6\} \Longrightarrow B$$

$$move(B, digit) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 3, 4, 5, 6\} \Longrightarrow B$$

$$move(B, underscore\_tail) = \{4, 6\}$$
$$\varepsilon - closure(\{4, 6\}) = \{4, 5, 6\} \Longrightarrow C$$

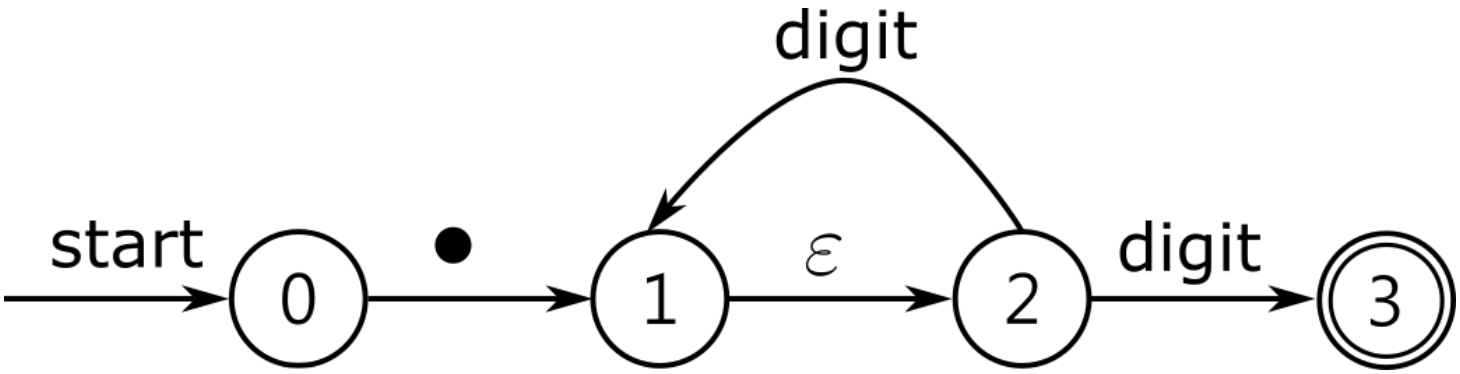
$$move(C, letter) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, underscore\_tail) = \{4, 6\}$$
$$\varepsilon - closure(\{4, 6\}) = \{4, 5, 6\} \Longrightarrow C$$

No	State	letter	digit	underscore_tail
0	A	B	-	-
1	B	B	B	C
2	C	-	-	C

6. fraction



$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, \cdot) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1, 2\} \Longrightarrow B$$

$$move(A, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

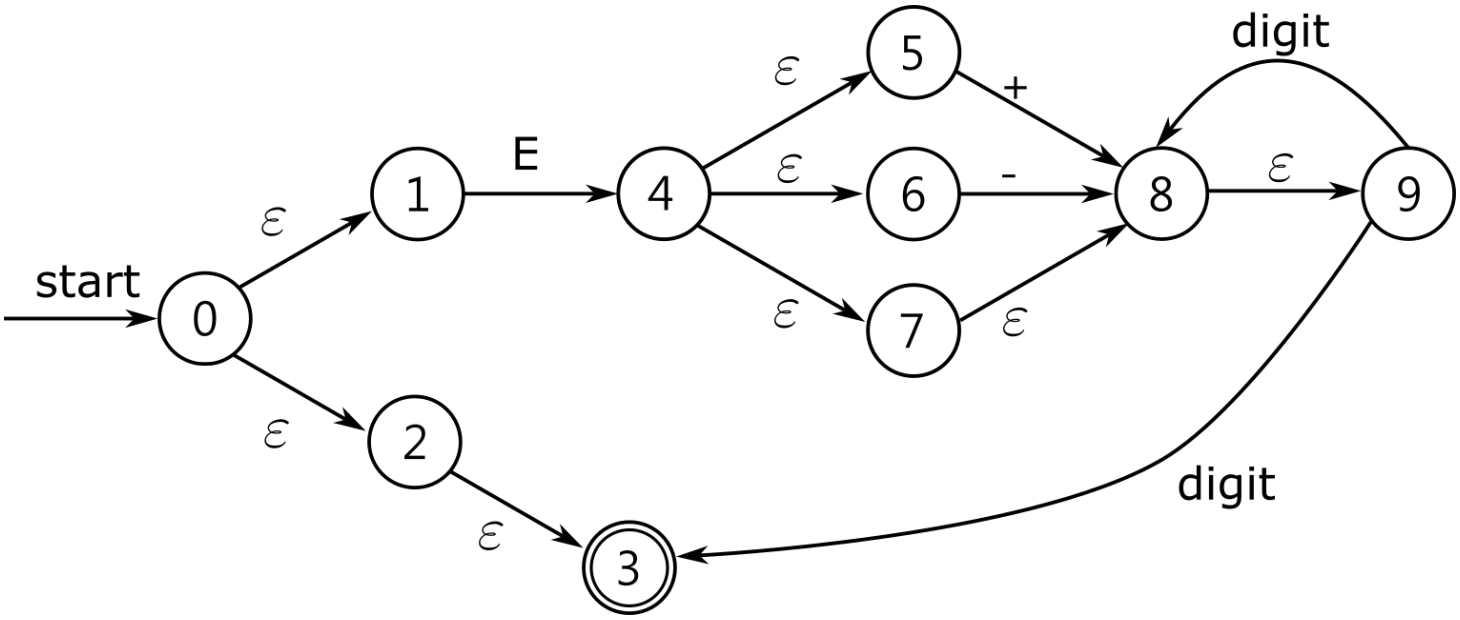
$$move(B, digit) = \{1, 3\}$$
$$\varepsilon - closure(\{1, 3\}) = \{1, 2, 3\} \Longrightarrow C$$

$$move(C, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, digit) = \{1, 3\}$$
$$\varepsilon - closure(\{1, 3\}) = \{1, 2, 3\} \Longrightarrow C$$

No	State	·	digit
0	A	B	-
1	B	-	C
2	C	-	C

7. optional_exponent



$$\varepsilon - closure(\{0\}) = \{0, 1, 2, 3\} \Longrightarrow A$$

$$move(A, E) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{4, 5, 6, 7, 8, 9\} \Longrightarrow B$$

$$move(A, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, +) = \{8\}$$
$$\varepsilon - closure(\{8\}) = \{8, 9\} \Longrightarrow C$$

$$move(B, -) = \{8\}$$
$$\varepsilon - closure(\{8\}) = \{8, 9\} \Longrightarrow C$$

$$move(B, digit)\{3, 8\}$$
$$\varepsilon - closure(\{3, 8\}) = \{3, 8, 9\} \Longrightarrow D$$

$$move(C, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, digit)\{3, 8\}$$
$$\varepsilon - closure(\{3, 8\}) = \{3, 8, 9\} \Longrightarrow D$$

$$move(D, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

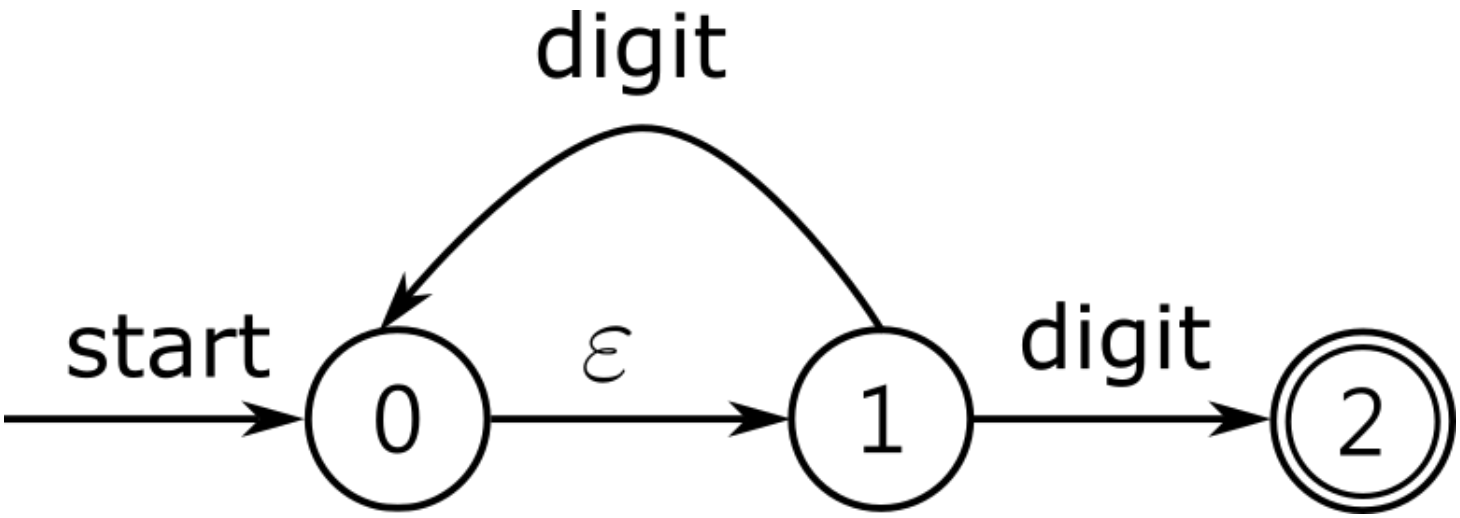
$$move(D, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, digit)\{3, 8\}$$
$$\varepsilon - closure(\{3, 8\}) = \{3, 8, 9\} \Longrightarrow D$$

No	State	E	+	-	digit
0	A	B	-	-	-
1	B	-	C	C	D
2	C	-	-	-	D
3	D	-	-	-	D

8. integer



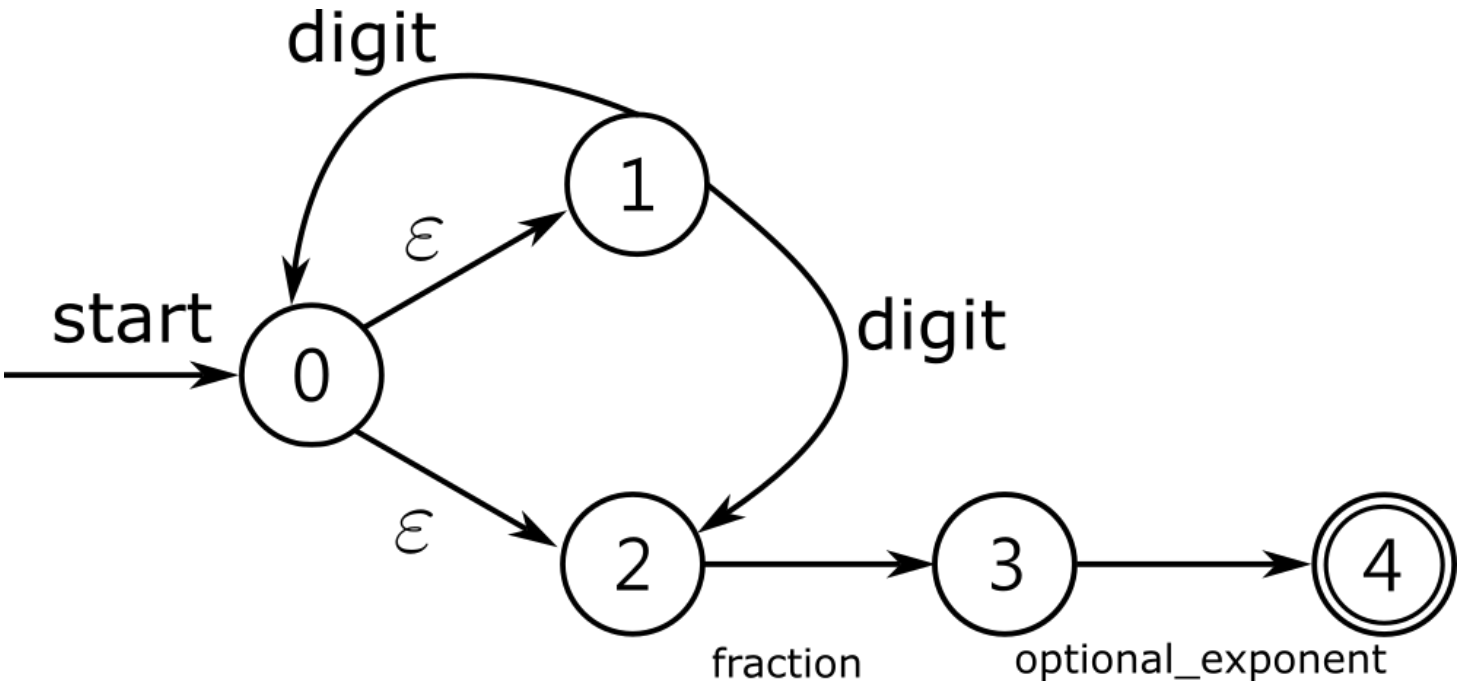
$$\varepsilon - closure(\{0\}) = \{0, 1\} \Longrightarrow A$$

$$move(A, digit) = \{0, 2\}$$
$$\varepsilon - closure(\{0, 2\}) = \{0, 1, 2\} \Longrightarrow B$$

$$move(B, digit) = \{0, 2\}$$
$$\varepsilon - closure(\{0, 2\}) = \{0, 1, 2\} \Longrightarrow B$$

No	State	digit
0	A	B
1	B	B

9. float



$$\varepsilon - closure(\{0\}) = \{0, 1, 2\} \Longrightarrow A$$

$$move(A, digit) = \{0, 2\}$$
$$\varepsilon - closure(\{0, 2\}) = \{0, 1, 2\} \Longrightarrow A$$

$$move(A, fraction) = \{3\}$$
$$\varepsilon - closure(\{3\}) = \{3\} \Longrightarrow B$$

$$move(A, optional\_exponent) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, fraction) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, optional\_exponent) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{4\} \Longrightarrow C$$

$$move(C, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, fraction) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, optional\_exponent) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

No	State	digit	fraction	optional_exponent
0	A	A	B	-
1	B	-	-	C
2	C	-	-	-

DFA + DFA -> NFA -> DFA

接下來我們要處理一些複合型的 token

1. id

第一個是 id

- id

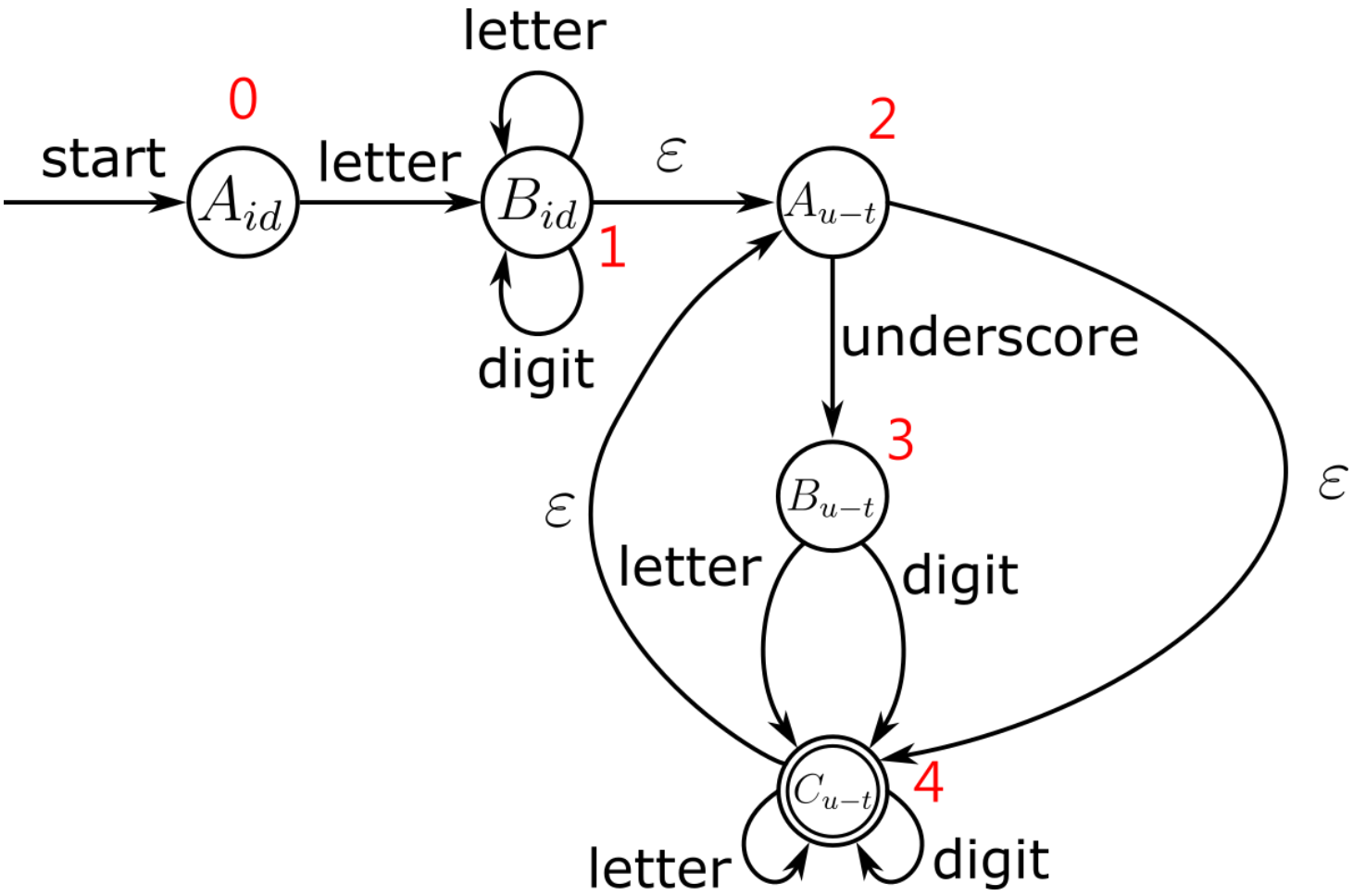
No	State	letter	digit	underscore_tail
0	A	B	-	-
1	B	B	B	C
2	C	-	-	C

可以看到，在 id 的 DFA transition table 裡面有一個叫做 underscore_tail 的東西，它還可以再拆解，我們前面的 DFA 也有提到

- underscore_tail

No	State	underscore	letter	digit
0	A	B	-	-
1	B	-	C	C
2	C	-	C	C

於是我們可以通過 id 和 underscore_tail 的 DFA Transition table，來重構出一張新的 NFA 圖形:



接著將這個 NFA 再轉換成 DFA，就像我們前面有做過的方式，這邊就不再重複。

$$\varepsilon - closure(\{0\}) = \{0\} \Longrightarrow A$$

$$move(A, letter) = \{1\}$$
$$\varepsilon - closure(\{1\}) = \{1, 2, 4\} \Longrightarrow B$$

$$move(A, digit) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, underscore) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, letter) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 4\} \Longrightarrow B$$

$$move(B, digit) = \{1, 4\}$$
$$\varepsilon - closure(\{1, 4\}) = \{1, 2, 4\} \Longrightarrow B$$

$$move(B, underscore) = \{3\}$$
$$\varepsilon - closure(\{3\}) = \{3\} \Longrightarrow C$$

$$move(C, letter) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{2, 4\} \Longrightarrow D$$

$$move(C, digit) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{2, 4\} \Longrightarrow D$$

$$move(C, underscore) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, letter) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{2, 4\} \Longrightarrow D$$

$$move(D, digit) = \{4\}$$
$$\varepsilon - closure(\{4\}) = \{2, 4\} \Longrightarrow D$$

$$move(D, underscore) = \{3\}$$
$$\varepsilon - closure(\{3\}) = \{3\} \Longrightarrow C$$

完成後，就可以得到新的 id 的 DFA Transition table:

No	State	letter	digit	underscore
0	A	B	-	-
1	B	B	B	C
2	C	D	D	-
3	D	D	D	C

2. float

接下來是 float 的部分，在 float 的 DFA transition table 中，可以看到也有一個 optional_exponent，也是可以再拆解

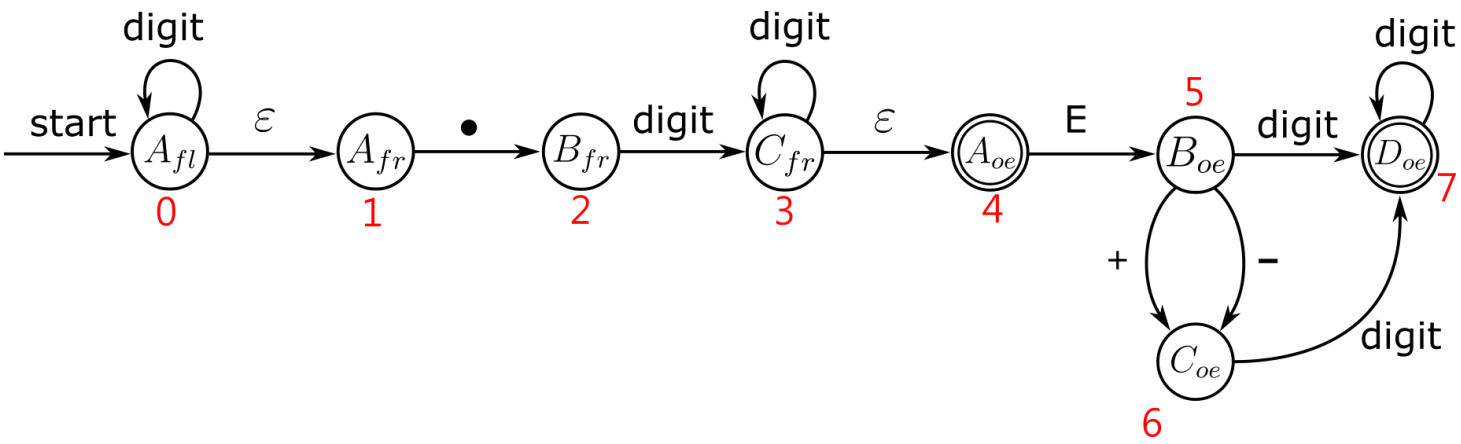
- float

No	State	digit	fraction	optional_exponent
0	A	A	B	-
1	B	-	-	C
2	C	-	-	-

- optional_exponent

No	State	E	+	-	digit
0	A	B	-	-	-
1	B	-	C	C	D
2	C	-	-	-	D
3	D	-	-	-	D

我們就一樣結合 float 和 optional_exponent 的 DFA transition table，產生新的 float NFA



再由這個新的 NFA 來產生 DFA

$$\varepsilon - closure(\{0\}) = \{0, 1\} \Longrightarrow A$$

$$move(A, digit) = \{0\}$$
$$\varepsilon - closure(\{0\}) = \{0, 1\} \Longrightarrow A$$

$$move(A, \cdot) = \{2\}$$
$$\varepsilon - closure(\{2\}) = \{2\} \Longrightarrow B$$

$$move(A, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(A, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, digit) = \{3\}$$
$$\varepsilon - closure(\{3\}) = \{3, 4\} \Longrightarrow C$$

$$move(B, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(B, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, digit) = \{3\}$$
$$\varepsilon - closure(\{3\}) = \{3, 4\} \Longrightarrow C$$

$$move(C, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, E) = \{5\}$$
$$\varepsilon - closure(\{5\}) = \{5\} \Longrightarrow D$$

$$move(C, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(C, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, digit) = \{7\}$$
$$\varepsilon - closure(\{7\}) = \{7\} \Longrightarrow E$$

$$move(D, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(D, +) = \{6\}$$
$$\varepsilon - closure(\{6\}) = \{6\} \Longrightarrow F$$

$$move(D, -) = \{6\}$$
$$\varepsilon - closure(\{6\}) = \{6\} \Longrightarrow F$$

$$move(E, digit) = \{7\}$$
$$\varepsilon - closure(\{7\}) = \{7\} \Longrightarrow E$$

$$move(E, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(E, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(E, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(E, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(F, digit) = \{7\}$$
$$\varepsilon - closure(\{7\}) = \{7\} \Longrightarrow E$$

$$move(F, \cdot) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(F, E) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(F, +) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

$$move(F, -) = \phi$$
$$\varepsilon - closure(\phi) = \phi$$

No	State	·	digit	E	+	-
0	A	B	A	-	-	-
1	B	-	C	-	-	-
2	C	-	C	D	-	-
3	D	-	E	-	F	F
4	E	-	E	-	-	-
5	F	-	E	-	-	-

完成後就可以得到新的 DFA transition table

程式

我們的程式是用 python 來實作

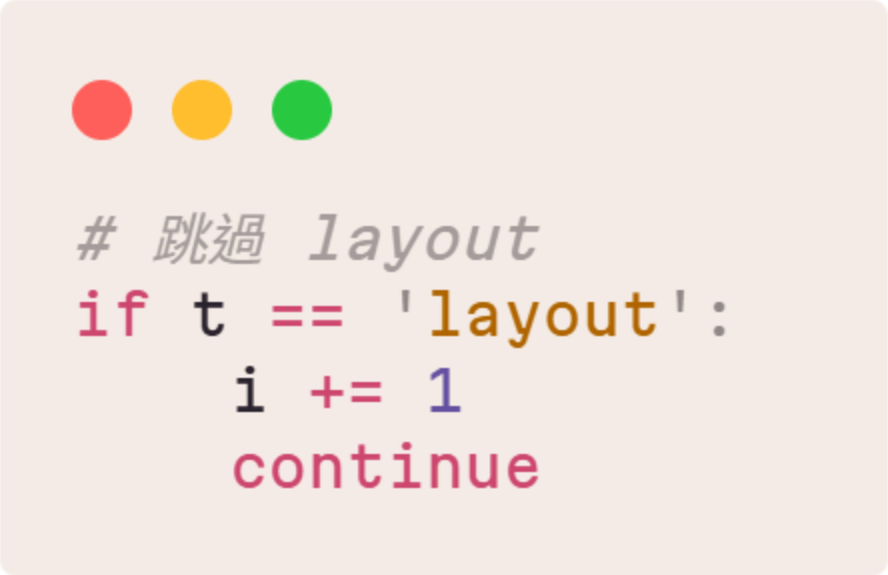
讀取字元

我們首先要逐個字元來讀取，並判斷讀取到什麼類型的字元:

```python
def char_type(ch):
    ascii_val = ord(ch)  # 取得 ASCII 編碼

    # 判斷是否為字母 (A-Z, a-z)
    if (65 <= ascii_val <= 90) or (97 <= ascii_val <= 122):
        return 'letter'

    # 判斷是否為數字 (0-9)
    elif 48 <= ascii_val <= 57:
        return 'digit'

    # 判斷是否為底線 (_)
    elif ascii_val == 95:
        return 'underscore'

    # 判斷是否為空白字元 (空格、tab、換行、Enter)
    elif ascii_val in (9, 10, 13, 32):
        return 'layout'

    else:
        return ch  # 其他字元
```

除了英文字母跟數字之外，還要判斷底線和空格等等。

如果讀取到的是空格的類別 layout，就直接跳過:



```
# 跳過 layout
if t == 'layout':
    i += 1
    continue
```

處理完空格之後，題目要求我們要可以辨識出:
(Separators and Brackets, Binary Operators,
Keywords, Comments, Identity, Integer, Float)

所以我們先定義一下每一個類別的 token

因為 Separators and Brackets, Binary Operators, Keywords 這幾種，裡面都是 or 的關係，像是 Keywords

Keywords: the language uses the following keywords int, float, bool, void, while, if, else, for, return.

就會是有列舉出來的這幾種之一，所以簡單的用 list 的方式來處理

```
# 比較簡單的定義，只會是裡面的其中一個

# Separators
separators = {';'}

# Brackets
brackets = {'(', ')', '[', ']', '{', '}'}

# Binary Operators
binary_Operators = ['<=', '>=', '<>', '==', '+', '-', '*', '/', '<', '>', '=']

# Keywords
keywords = {'int', 'float', 'bool', 'void',
            'while', 'if', 'else', 'for', 'return'}
```

那 Comments 的部分，因為需要識別開頭和結尾，所以額外寫他的規則:

```
if dfa is dfa_comment:
            if state == 0:
                if c == '/':
                    state = 1
                else:
                    break
            elif state == 1:
                if c == '*':
                    state = 2
                else:
                    break
            elif state == 2:
                if c == '*':
                    state = 3
                else:
                    # 只要不是 '*', 留在狀態 2
                    state = 2
            elif state == 3:
                if c == '/':
                    state = 4
                elif c == '*':
                    state = 3
                else:
                    state = 2
            else:
                break

            if state in accept_states:
                last_accept = state
                last_accept_pos = i
            i += 1
            continue
```

讀取了開頭的 /* 後，判斷接下來是不是讀到 *，如果不是的話，就會一直停在狀態 2，這樣就可以判斷出，是不是完整的註解 /* */ 配對，也就是看註解符號有沒有閉合

id

接下來是 ID 的部分，這邊看到的是重構後的 transition table，我們需要把 DFA 的狀態轉移，放入到程式中

- DFA(重構後)

No	State	letter	digit	underscore
0	A	B	-	-
1	B	B	B	C
2	C	D	D	-
3	D	D	D	C

將原本的狀態 A~D 重新編號成 0~3 狀態，在 python 程式中，使用 dictionary 的方式，來進行對應

```python
# id DFA
dfa_id = {
    0: {'letter': 1},
    1: {'letter': 1, 'digit': 1, 'underscore': 2},
    2: {'letter': 3, 'digit': 3},
    3: {'letter': 3, 'digit': 3, 'underscore': 2},
}
```

像這邊

從原本初始狀態 A，也就是 0，如果讀了 letter，就會跑到狀態 B，也就是狀態 1，那程式中就寫上，狀態 0，讀取 letter，就切換到狀態 1。

那在狀態 B，也就是 1，如果讀取了 letter 或是 digit，還是會留在狀態 B，讀取 underscore 會轉換到狀態 C，那程式中就寫上，狀態 1，讀取 letter，切換到狀態 1，讀取 digit，切換到狀態 1，讀取 underscore，切換到狀態 2。

後面依此類推，最後我們需要決定一下，在哪些狀態，是可以最為合法的結束狀態。

狀態 A 是初始狀態，當然不能當作是結束狀態。

而狀態 B，是從 A 讀了英文字母之後，或是 B 自己讀了字元或數字，也就是這串輸入的最後是英文字母或是數字，因此狀態 B 是可以作為結束狀態的。

狀態 C，是狀態 B 或 D 讀取了底線而來，也就是這串輸入的最後是底線，那根據規則:

$$underscore \rightarrow \_$$
$$letter \rightarrow A|B...|Z|a|b...|z$$
$$digit \rightarrow 0|1|2...|9$$
$$underscore\_tail \rightarrow underscore\ (letter|digit)^+$$
$$id \rightarrow letter\ (letter|digit)^*\ underscore\_tail^*$$
$$fraction \rightarrow .\ digit^+$$
$$optional\_exponent \rightarrow (E\ (+|-|\varepsilon)\ digit^+)|\varepsilon$$
$$integer \rightarrow\ digit^+$$
$$float \rightarrow digit^*\ fraction\ optional\_exponent$$

id 的結尾不可能是底線，所以狀態 C 不能當作結束狀態。

狀態 D，是由狀態 C 讀了英文字母或數字而來，也就是底線後面跟了英文字母或數字，或是從狀態 D 再讀英文字母或數字，因此這串輸入的最後面是英文字母或是數字，可以當作結束狀態。

id 這個 DFA，可以做為結束狀態的有 B 和 D，也就是狀態 1 和 3，因此我們把合法的結束狀態記錄下來:

```
id_accept_states = {1, 3}
```

另外的 float 和 integer 也是一樣的做法，把 DFA 的結果寫入程式中。

- float

```
# float DFA
dfa_float = {
    0: {'digit': 0, '.': 1},
    1: {'digit': 2},
    2: {'digit': 2, 'E': 3},
    3: {'digit': 5, '+': 4, '-': 4},
    4: {'digit': 5},
    5: {'digit': 5},
}
float_accept_states = {2, 5}
```

- integer

```
# integer DFA
dfa_int = {
    0: {'digit': 1},
    1: {'digit': 1}
}
int_accept_states = {1}
```

那程式中一些迭代的過程，我們就不去提細節。

完整程式碼

```
# 在同個目錄下，建立 input.txt 檔案
# 裡面放入要測試的輸入

def char_type(ch):
    ascii_val = ord(ch)  # 取得 ASCII 編碼

    # 判斷是否為字母 (A-Z, a-z)
    if (65 <= ascii_val <= 90) or (97 <= ascii_val <= 122):
        return 'letter'

    # 判斷是否為數字 (0-9)
    elif 48 <= ascii_val <= 57:
        return 'digit'

    # 判斷是否為底線 (_)
    elif ascii_val == 95:
        return 'underscore'

    # 判斷是否為空白字元 (空格、tab、換行、Enter)
    elif ascii_val in (9, 10, 13, 32):
        return 'layout'

    else:
        return ch  # 其他字元


# 比較簡單的定義，只會是裡面的其中一個

# Separators
separators = {';'}

# Brackets
brackets = {'(', ')', '[', ']', '{', '}'}

# Binary Operators
binary_Operators = ['<=', '>=', '<>', '==', '+', '-', '*', '/', '<', '>', '=']

# Keywords
keywords = {'int', 'float', 'bool', 'void',
            'while', 'if', 'else', 'for', 'return'}

```

```
# Comments DFA
dfa_comments = {
    0: {'/': 1},
    1: {'*': 2},
    2: {'*': 3, 'other': 2},
    3: {'/': 4, '*': 3, 'other': 2}
}
comments_accept_states = {4}

# integer DFA
dfa_int = {
    0: {'digit': 1},
    1: {'digit': 1}
}
int_accept_states = {1}

# id DFA
dfa_id = {
    0: {'letter': 1},
    1: {'letter': 1, 'digit': 1, 'underscore': 2},
    2: {'letter': 3, 'digit': 3},
    3: {'letter': 3, 'digit': 3, 'underscore': 2},
}

id_accept_states = {1, 3}

# float DFA
dfa_float = {
    0: {'digit': 0, '.': 1},
    1: {'digit': 2},
    2: {'digit': 2, 'E': 3},
    3: {'digit': 5, '+': 4, '-': 4},
    4: {'digit': 5},
    5: {'digit': 5},
}
float_accept_states = {2, 5}


def run_dfa(dfa, accept_states, text, start):
    state = 0
    i = start
    last_accept = -1
    last_accept_pos = start - 1

```

```
    length = len(text)

    while i < length:
        c = text[i]
        t = char_type(c)

        # Comments DFA 特殊狀態轉移
        if dfa is dfa_comments:
            if state == 0:
                if c == '/':
                    state = 1
                else:
                    break
            elif state == 1:
                if c == '*':
                    state = 2
                else:
                    break
            elif state == 2:
                if c == '*':
                    state = 3
                else:
                    # 只要不是 '*', 留在狀態 2
                    state = 2
            elif state == 3:
                if c == '/':
                    state = 4
                elif c == '*':
                    state = 3
                else:
                    state = 2
            else:
                break

            if state in accept_states:
                last_accept = state
                last_accept_pos = i
            i += 1
            continue

        # 一般 DFA 狀態轉移
        transitions = dfa.get(state, {})

```

```
        if t in transitions:
            state = transitions[t]
        elif c in transitions:
            state = transitions[c]
        else:
            break

        if state in accept_states:
            last_accept = state
            last_accept_pos = i
        i += 1

    if last_accept == -1:
        return -1
    else:
        return last_accept_pos + 1


def lexer(text):
    i = 0
    length = len(text)
    tokens = []

    while i < length:
        c = text[i]
        t = char_type(c)

        # 跳過 layout
        if t == 'layout':
            i += 1
            continue

        try:
            # Comments DFA
            if text.startswith('/*', i):
                end_pos = run_dfa(
                    dfa_comments, comments_accept_states, text, i)
                if end_pos == -1:
                    tokens.append(('Binary Operators', '/'))
                    i += 1
                    continue
                else:
                    tokens.append(('Comments', text[i:end_pos]))

```

```
            i = end_pos
            continue

        # id DFA + keyword 判斷
        end_pos = run_dfa(dfa_id, id_accept_states, text, i)
        if end_pos != -1:
            word = text[i:end_pos]
            if word in keywords:
                tokens.append(('Keyword', word))
            else:
                tokens.append(('id', word))
            i = end_pos
            continue

        # float DFA
        end_pos = run_dfa(dfa_float, float_accept_states, text, i)
        if end_pos != -1:
            tokens.append(('float', text[i:end_pos]))
            i = end_pos
            continue

        # integer DFA
        end_pos = run_dfa(dfa_int, int_accept_states, text, i)
        if end_pos != -1:
            tokens.append(('integer', text[i:end_pos]))
            i = end_pos
            continue

        # 運算子優先匹配（長字串優先）
        matched_op = None
        for op in sorted(binary_Operators, key=lambda x: -len(x)):
            if text.startswith(op, i):
                matched_op = op
                break
        if matched_op:
            tokens.append(('Binary Operators', matched_op))
            i += len(matched_op)
            continue

        if c in brackets:
            tokens.append(('Brackets', c))
            i += 1
            continue
```

```
            if c in separators:
                tokens.append(('Separators', c))
                i += 1
                continue

            # 錯誤處理
            raise ValueError(f"Unknown token starting at position {i}: {c}")

        except ValueError as e:
            # 中止 lexer、印出目前 tokens
            print_results(tokens)

            raise e  # 讓主程式捕捉錯誤

    return tokens


def print_results(tokens):
    for ttype, val in tokens:
        print(f"{ttype:20}: {val}")


if __name__ == "__main__":
    try:
        with open("input.txt", "r", encoding="utf-8") as file:
            code = file.read()

        tokens = lexer(code)

        print_results(tokens)

    except ValueError as e:
        # 有錯誤的話就輸出錯誤訊息
        print(f"\nLexer Error: {e}")
```

Demo

我們用一個 input.txt 的檔案作為輸入

```python
with open("input.txt", "r", encoding="utf-8") as file:
    code = file.read()
```

正常情況

- 輸入

```
float x = 3.14E-2;
if (x <= 10) {
/* This is a comment */
return x + 5;
}
```

- 輸出

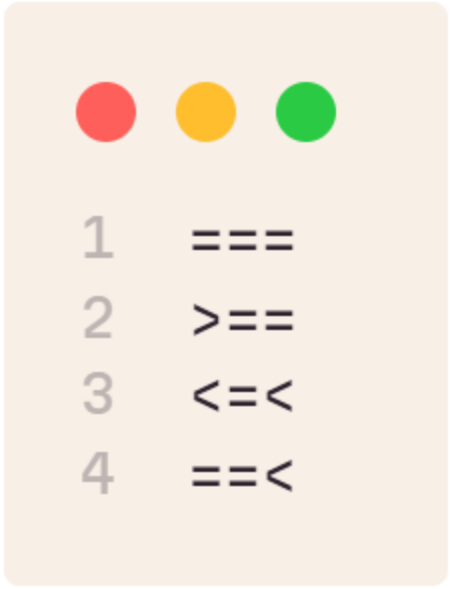
```
Final  base
❯ python index.py
Keyword                 : float
id                      : x
Binary Operators        : =
float                   : 3.14E-2
Separators              : ;
Keyword                 : if
Brackets                : (
id                      : x
Binary Operators        : <=
integer                 : 10
Brackets                : )
Brackets                : {
Comment                 : /* This is a comment */
Keyword                 : return
id                      : x
Binary Operators        : +
integer                 : 5
Separators              : ;
Brackets                : }
```

特殊情況

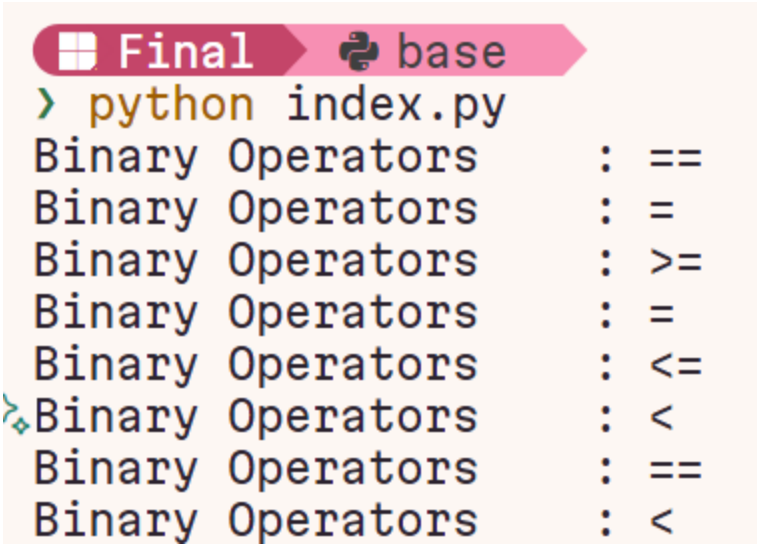
最長字串匹配

輸入會從前往後，從最常匹配的字串來判斷，像是輸入三個等於，會變成是前面兩個等號一組，後面單獨一個等號一組

- 輸入



- 輸出



這是因為他是用最常字串匹配的方式，他會先匹配最常符合的token類別
再來看第二個例子，他上會先匹配大於等於後再判斷另一個等於

錯誤輸出

像是我在原本正確的測試，多加了一個小老鼠 @ 符號，他就會顯示 Unknown token starting at position，但原本前面正確的 token 都還是會正確的輸出

- 輸入

```
1  if (x <= 10) {
2  /* This is a comment */
3  return x + 5; @
4  }
```

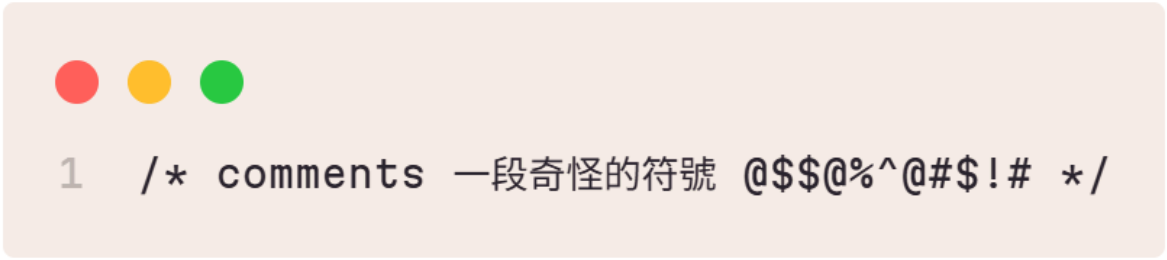
- 輸出

```
Final  base
❯ python index.py
Keyword                 : if
Brackets                : (
id                      : x
Binary Operators        : <=
integer                 : 10
Brackets                : )
Brackets                : {
Comment                 : /* This is a comment */
Keyword                 : return
id                      : x
Binary Operators        : +
integer                 : 5
Separators              : ;

Lexer Error: Unknown token starting at position 61: @
Final  base
❯
```

Comments

Comments 不會去管裡面寫些什麼，只要當作是 Comments 就會完整印出內容

- 輸入



```
1  /* comments 一段奇怪的符號 @$$@%^@#$!# */
```

- 輸出

```
Final  base
❯ python index.py
Comments                : /* comments 一段奇怪的符號 @$$@%^@#$!# */
Final  base
❯
```

如果是沒有閉合的註解，那 / 和 * 就會被判斷為 Binary Operator，而 comments 也就會被判斷成 id

- 輸入

```
1  /* comments /
```

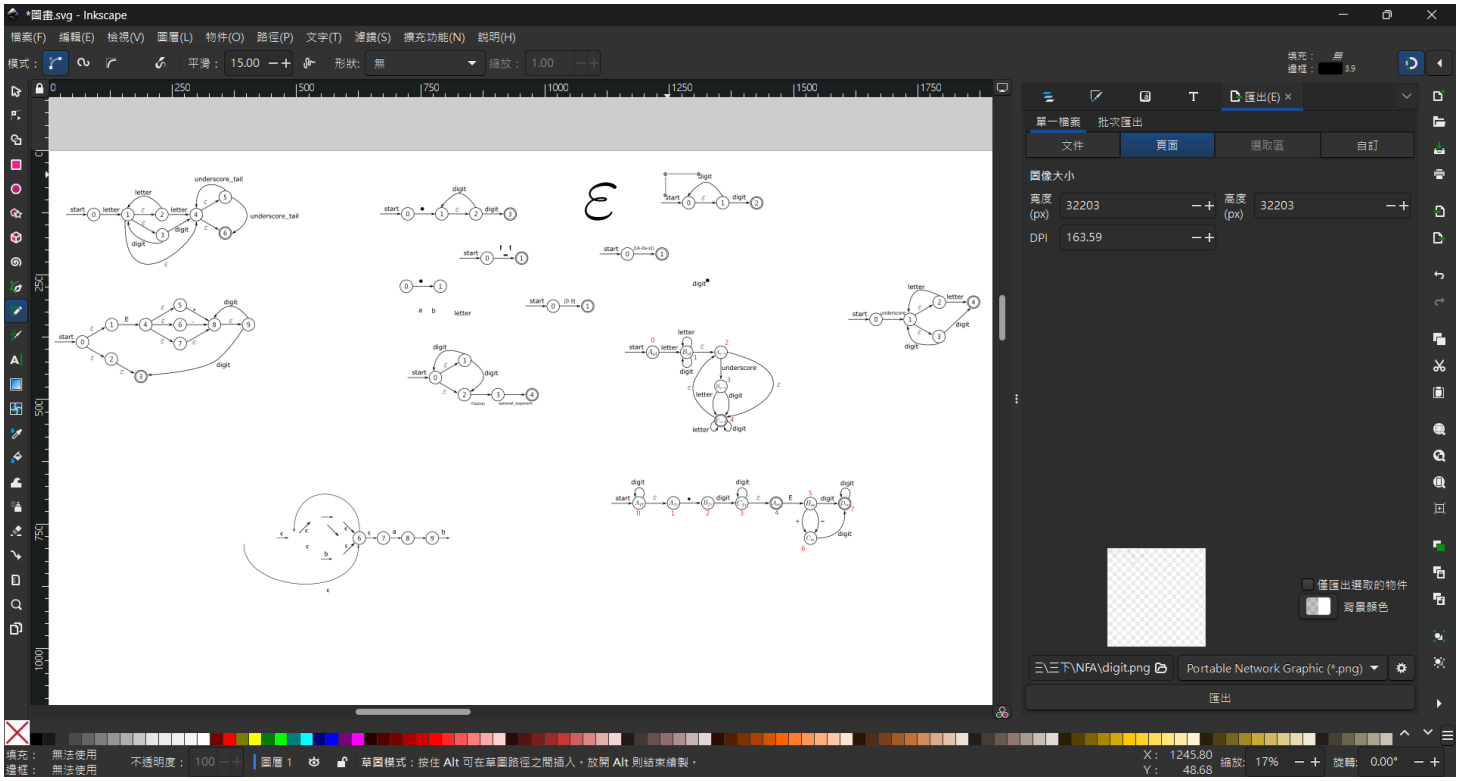
- 輸出

```
Final  base
❯ python index.py
Binary Operators    : /
Binary Operators    : *
id                  : comments
Binary Operators    : /
Final  base
❯
```

結論

我們用手繪的方式，將 Regular Expression 透過 Dotted Item 的方法轉換成 NFA，並畫出 NFA 圖形

我們使用的工具是 Inkscape 將手繪圖形繪製成電子檔案



再用 NFA 圖形，畫出 DFA 的 transition table，接著將 DFA 的過程加入程式中，再不使用 lex 或 similar lexical analyzer的情況下，完成 lexical analysis 的實作去分析讀到哪些Token

參考資料

- 老師上課 PPT