

自動機理論

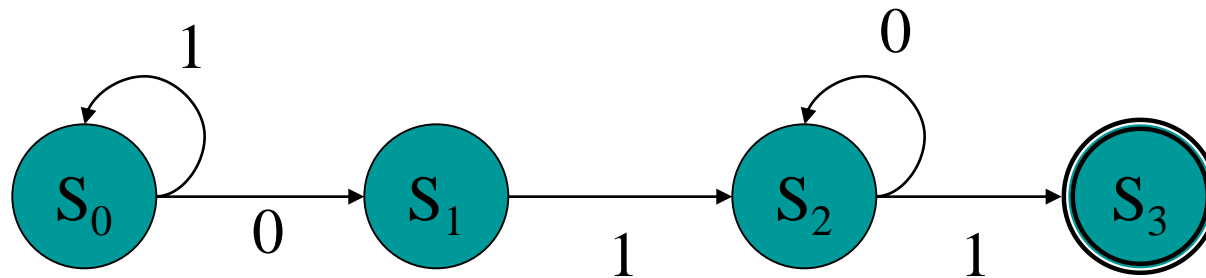
- 文法與機器
- 文法與機器之關係
- BNF 語法與正規語法
 - BNF 描述法
 - 正規語法之描述
- 正規文法與自動機之關係
- 有限自動機轉換正規文法

文法與機器

- 有限狀態機 (Finite State Machine, FSM) 或稱為有限自動機 (Finite Automata)
 - 設某狀態 S_i 為可接受的 (Acceptable) 狀態
 - 輸入字串 (String) 經一連串的轉移 (Transit) 後恰好到達為可接受的 (Acceptable) 狀態 S_i ,
 - 則此一字串為合法字串 (Legal String);
 - 否則稱之為不合法字串 (Illegal String)
 - 所有可被此 FSM 接受的字串所成的集合稱此集合為可被此 FSM 認知 (Recognized) 的語言 (Language)

文法與機器

■ 有限狀態機範例



- 101001 與 0101 皆可被此FSM所接受,但0111 則不能被此FSM 所認知(Recognize)
- 經推演後得知,此FSM所認知的語言為 1^*010^*1
- Acceptable state 稱為 Final State 以雙圈表示,而單圈表示一般狀態
- Acceptable State 可以不止一個

文法與機器

■ 文法(語法)

■ G 被稱為文法或稱語法(Grammar),

若 $G = \langle N, T, \Sigma, P \rangle$, 其中

N: Non-terminal 的集合。

T: Terminal 的集合。

Σ : Start 符號, $\Sigma \in N$ 。

P: 產生規則的集合。

如 $\alpha \rightarrow \beta$ 其中 $\alpha, \beta \in (N \cup T)^*$, $\alpha \neq \lambda$; 即 α 不可為空字串。
 $(N \cap T) = \phi$ 。

文法與機器

■ 文法範例

$G = \langle N, T, \Sigma, P \rangle$

$N = \{A, B, \Sigma\}$

$T = \{0, 1\}$

$P = \{ \begin{array}{l} \Sigma \rightarrow A1, \\ A \rightarrow B0, \\ B \rightarrow 1, \\ B \rightarrow B1 \end{array} \}$

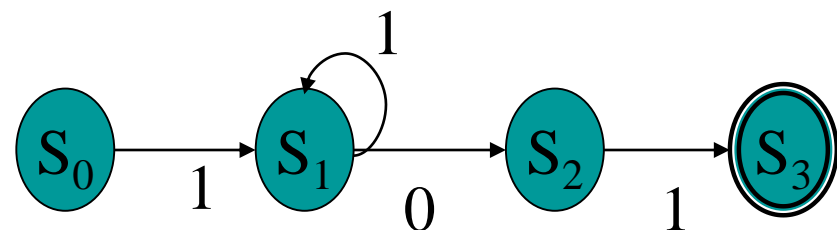
$\Sigma \rightarrow A1$

$\rightarrow B01 \rightarrow (B1)01 \rightarrow (B1)101$

$\rightarrow B1^n 01 \rightarrow 11^n 01$

$\rightarrow 1^{n+1} 01 \rightarrow 1^+ 01$

即 $\{1^+ 01\} = \{101, 1101, 11101, \dots\}$



(FSM)

文法與機器之關係

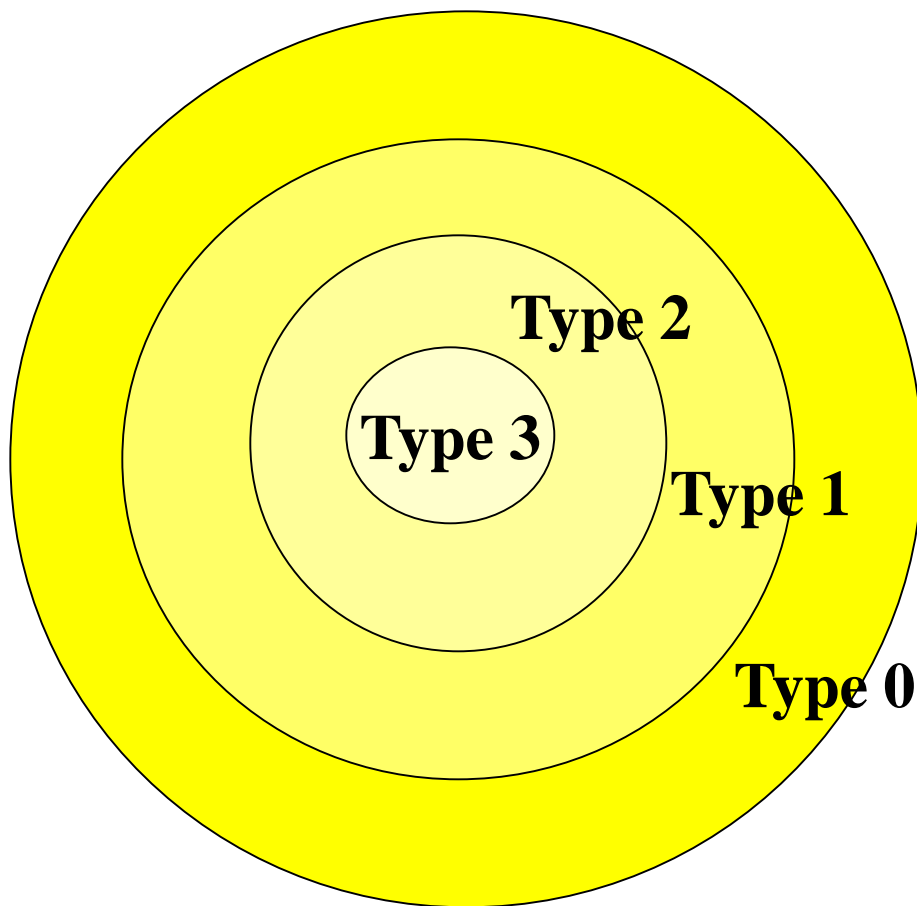
語言	文法型態	相對應之機器	產生規則	
Type 0	Unrestricted grammar	Turing machine	$\alpha \rightarrow \beta,$ $\alpha \neq \lambda$ $\alpha, \beta \in (N \cup T)^*$	
Type 1	Context sensitive Grammar	Linear bounded automata	$\gamma_1 \alpha \gamma_2 \rightarrow \gamma_1 \beta \gamma_2$ $\gamma_1, \gamma_2 \in (N \cup T)^*$ $\alpha \in N, \beta \in (N \cup T)^* - \lambda,$ $\text{length}(\alpha) \leq \text{length}(\beta)$	
Type 2	Context free grammar	Pushdown automata	$A \rightarrow \beta,$ $\beta \in (N \cup T)^* - \lambda,$ $A \in N$	
Type 3	Regular grammar	Finite automata or FSM	$A \rightarrow aB,$ $A \rightarrow a,$ $A, B \in N,$ $a \in T,$ (right-linear)	$A \rightarrow Ba,$ $A \rightarrow a,$ $A, B \in N,$ $a \in T,$ (left-linear)

文法與機器之關係

- Type 0(Unrestricted Grammar)
 - 沒有限制的文法
- Type 1(Context Sensitive Grammar)
 - 與內容有關的文法
- Type 2(Context free Grammar)
 - 與內容無關的文法,規則的左邊僅能有一 Non-terminal Symbol.
- Type 3(Regular Grammar)
 - 規則的左邊與右邊只能有一個Non-terminal symbol, 且產生規則的右邊也僅能有一個Terminal Symbol.

文法與機器之關係

■ 關係圖



BNF 描述法

- BNF (Backus Normal Form or Backus Naur Form)
 - 自 Backus and Naur 創建BNF描述法定義了 ALGOL 60 的構文(Syntax)以來許多計算機語言都採BNF描述法來描述程式語言的架構
- BNF 所使用的符號
 - ” | ”表示 ” 或(or)”
 - ” ::= ” 表示 ” 定義為(Define)”
 - ”被<>所括住者” 表示 ” Non-terminal Symbol”
 - ”沒被<>所括住者” 表示” Terminal Symbol”

BNF 描述法

■ 範例

<digital> ::= 0|1|2|3|4|5|6|7|8|9

<letter> ::=

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<Identifier> ::=

<letter>|<Identifier><letter>|<Identifier><digit>

BNF 描述法

- BNF僅可描述Type 2 之文法
- 優點
 - 明確易懂
 - 較易於建構有效的Parser.
 - 較易將程式翻譯成機器碼及易於偵測出程式中語法的錯誤

正規語法之描述

- 假設 I 為輸入集合(Input Set), 則正規表示式(regular Expression) 定義為:

法則1:

- ε 為一個正規表示式, 寫成 $\{\varepsilon\}$, 即表示含空字串之語言

法則2:

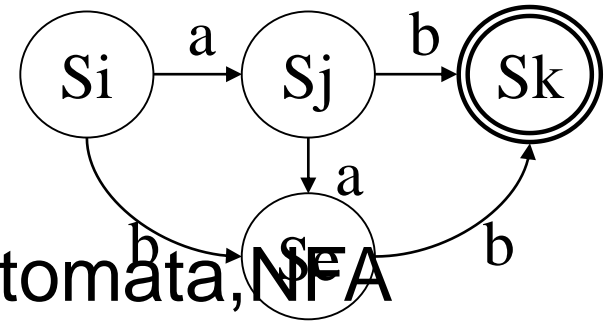
- 對於所有的 $c \in I$, 若 c 為一個正規表示式, 則寫成 $\{c\}$, 即表示含有一個文字(包括數字)之語言

法則3:

- 若 S 與 R 為兩個正規表示式, 它們分別表示 L_R 與 L_S 兩種語言, 則
- $(R)|(S)$ 的正規表示式表示 $(L_R \cup L_S)$
- $(R) \cdot (S)$ 的正規表示式表示 $(L_R \cdot L_S)$
- $(R)^*$ 的正規表示式表示 L_R^*

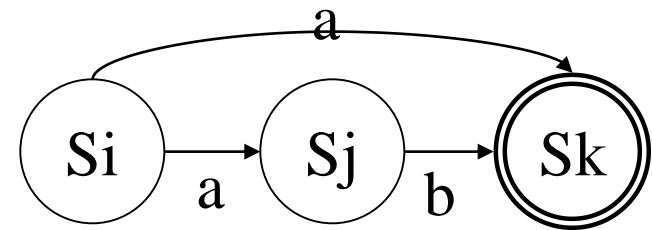
正規文法與自動機之關係

■ Deterministic Finite Automata, DFA



■ Non-deterministic Finite Automata, NFA

P.S.NFA一定可以化簡成一個DFA



正規文法與自動機之關係

- 正規表示式轉換成NFA
- NFA 轉換成DFA
- DFA之最小化

正規表示式轉換成NFA

■ 方法如下

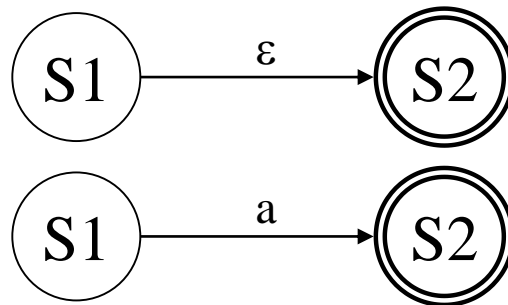
輸入: 文字集($N \cup T$)上所定義之正規表示式 R

輸出: 一個可以接受正規表示式 R 所定義之語言的NFA

方法: 將正規表示式分解為Primitive Components, 對於每一個基本成份建立一個對應的NFA

正規表示式轉換成NFA

- 對於基本成分建立對應的NFA 的規則如下
- Basic Regular Expression

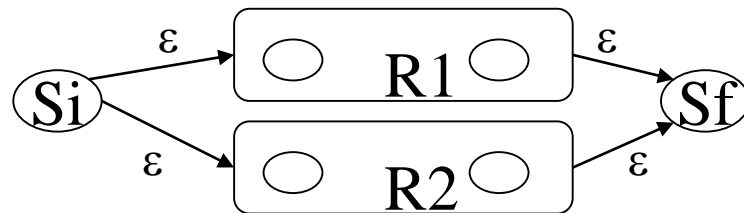


正規表示式轉換成NFA

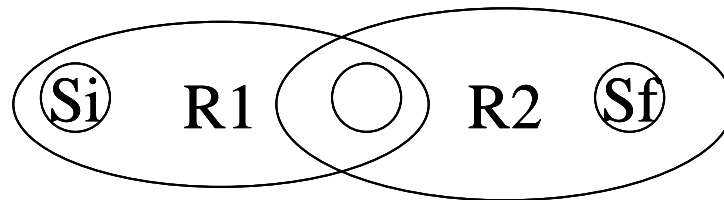
■ Compound Regular Expression

a. 先建立 Si(Initial State) 及 Sf(Final State)

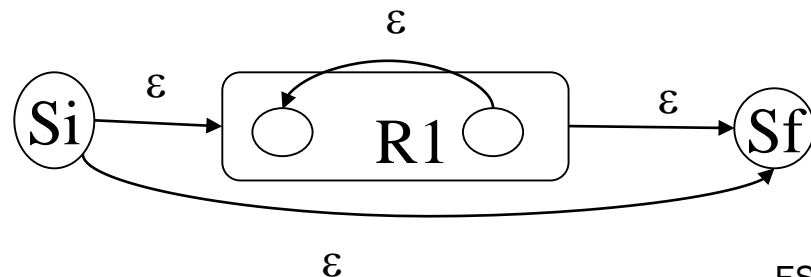
b. $R1|R2$



c. $R1.R2$

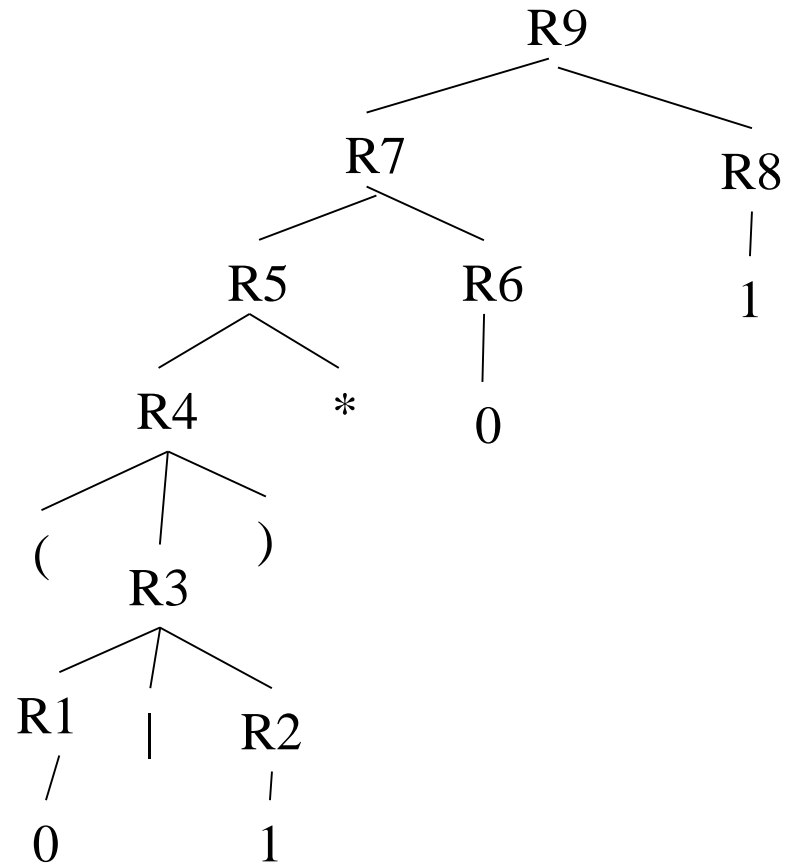


d. $R1^*$



範例：將正規表示式 $R=(0|1)^*01$ 轉成 NFA

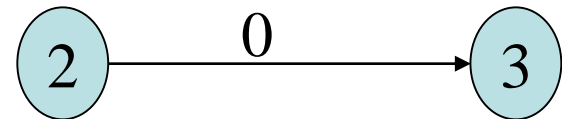
Step 1: 分解成基本成份



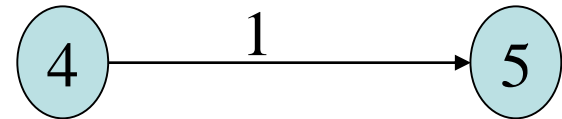
範例：將正規表示式 $R=(0|1)^*01$ 轉成NFA

Step2: 求其Primitive NFA 與Compound NFA

1.(Primitive Component)R1



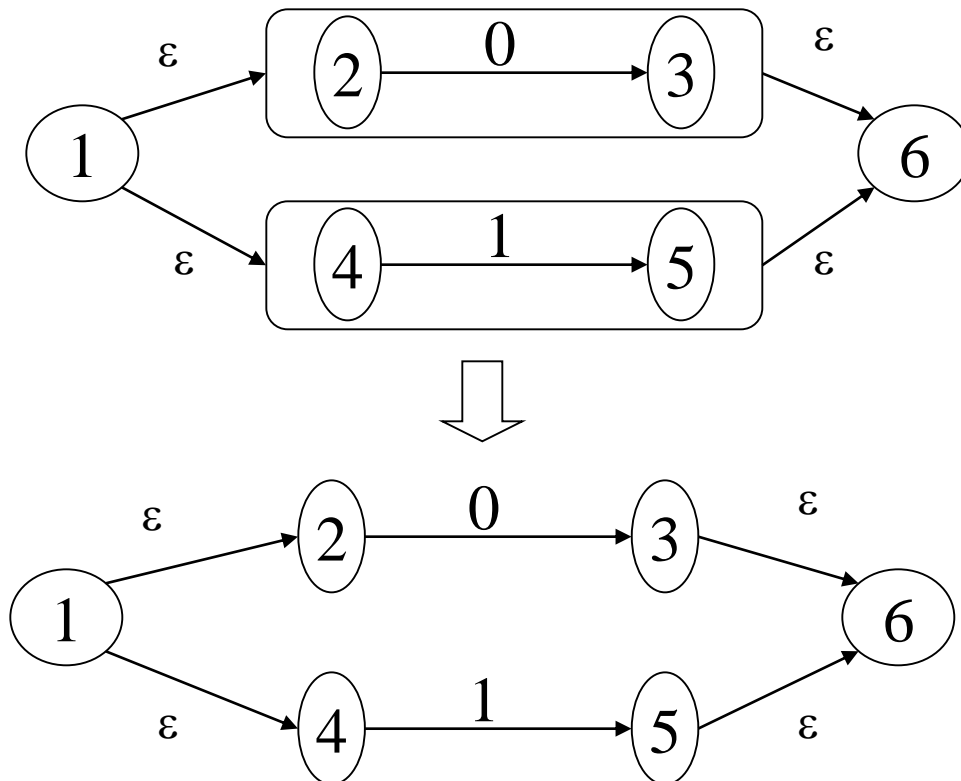
2.(Primitive Component)R2



範例：將正規表示式 $R=(0|1)^*01$ 轉成 NFA

Step2: 求其 Primitive NFA 與 Compound NFA

3.(Compound Component) $R3=R1|R2$

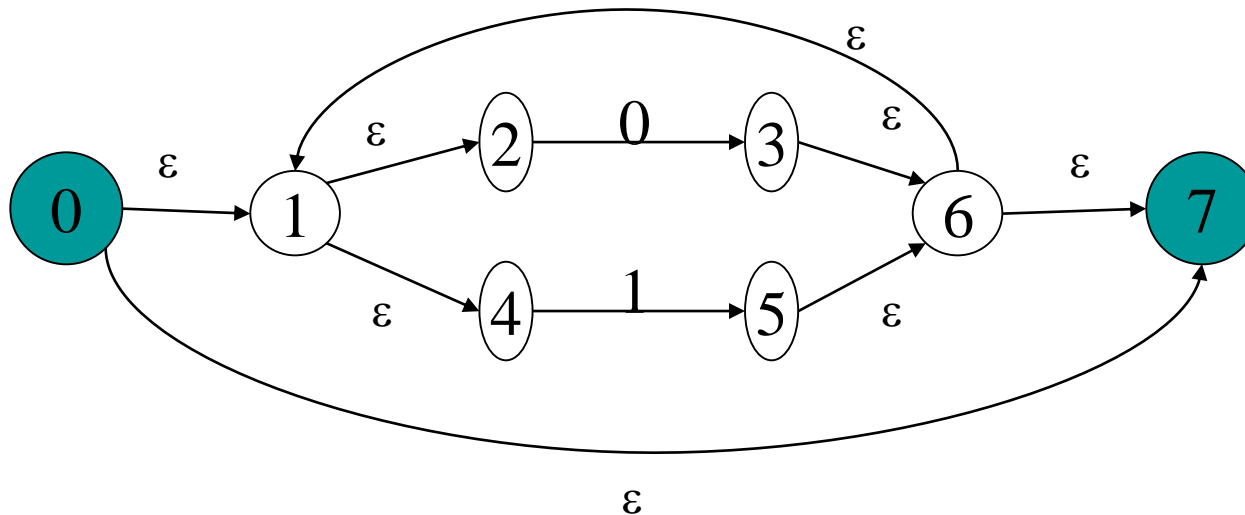


範例：將正規表示式 $R=(0|1)^*01$ 轉成 NFA

Step2: 求其 Primitive NFA 與 Compound NFA

4. (Compound Component) $R4=R3$

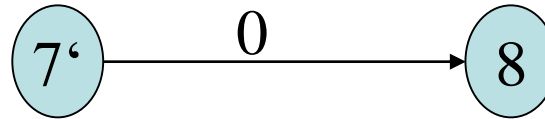
5. .(Compound Component) $R5=R4^*$




範例：將正規表示式 $R=(0|1)^*01$ 轉成 NFA

Step2: 求其 Primitive NFA 與 Compound NFA

6. (Compound Component) $R_6=0$

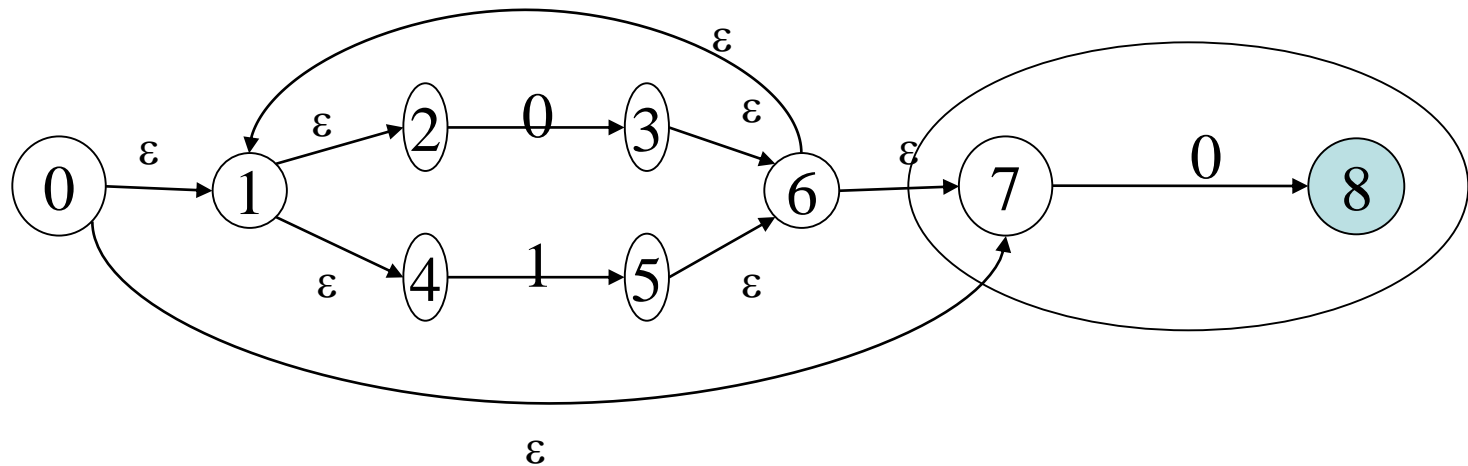


- 取用  是便於稍後合併時無需再修改其它狀態之值

範例：將正規表示式 $R=(0|1)^*01$ 轉成NFA

Step2: 求其Primitive NFA 與Compound NFA

7. (Compound Component) $R7=R5R6$

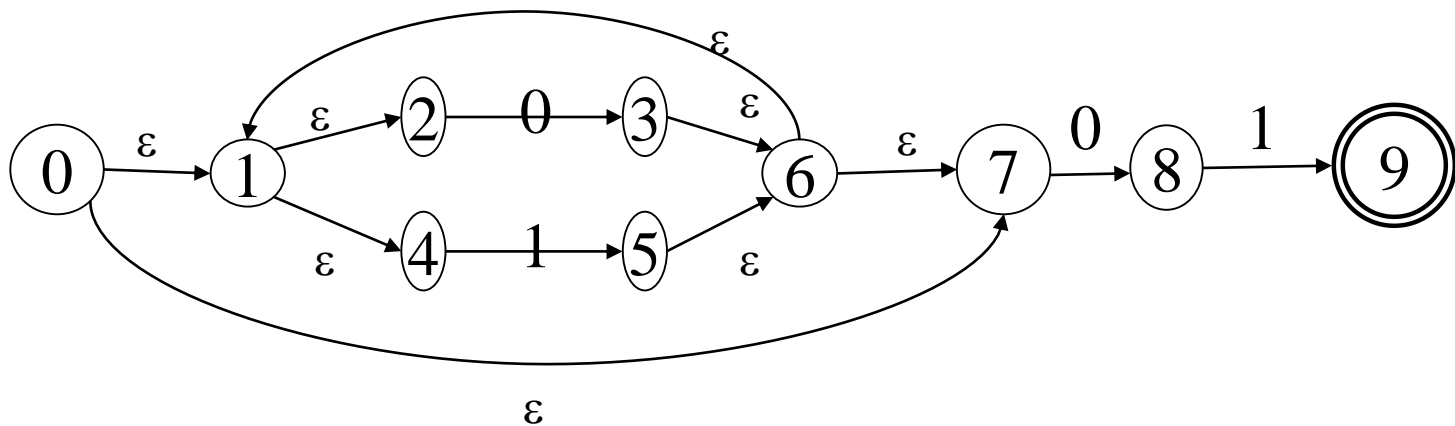


• 合併 $7'$ 與 7

範例：將正規表示式 $R=(0|1)^*01$ 轉成 NFA

Step2: 求其 Primitive NFA 與 Compound NFA

8. (Compound Component) $R_9=(0|1)^*01$



NFA 轉換成 DFA

- 每一個NFA皆可再被簡化成DFA,但依NFA 的特性細分成
 - NFA 中具有 ε 符號者
 - NFA 中無 ε 符號者

NFA 轉換成 DFA

- NFA 中具有 ε 符號者
 - 處理方式主要由 ε -封閉處理(ε -CLOSURE)與子集合建構兩個演算法來負責.

演算法:NFA=>DFA

(NFA 中具有 ϵ 符號者)

Step 1:

欲將N:NFA簡化為 D:DFA . D's Initial state 是包含出始狀態 S_0 的集合,故將 S_0 加入(ϵ -CLOSURE(S_0))中,即可將經由 S_0 輸入 ϵ 到達之狀態均加入 ϵ -CLOSURE(S_0)中,此集合即為DFA之初始狀態,並標記(Marked State)

Step 2:

若已標記的狀態輸入符號 $a \neq \epsilon$ 會轉移至新的未標記的小狀態,將這些小狀態集合起來,令其為 S_j , 然後求 S_j 的 ϵ -CLOSURE.

Step 3:

繼續尋找下一個未標記的狀態,重覆Step 2,直到沒有未標記狀態為止 .

演算法:NFA \Rightarrow DFA

(NFA 中具有 ε 符號者)

Step 4:

由上述步驟可求得一轉移表(Transition Table),若轉移表中之以標記的狀態內含有原來NFA之初始狀態(Initial State),則令此狀態為DFA之初始狀態.同理,若轉移表中之以標記的狀態內含有原來NFA之終止狀態,則令此狀態為DFA之終止狀態(Finial State).

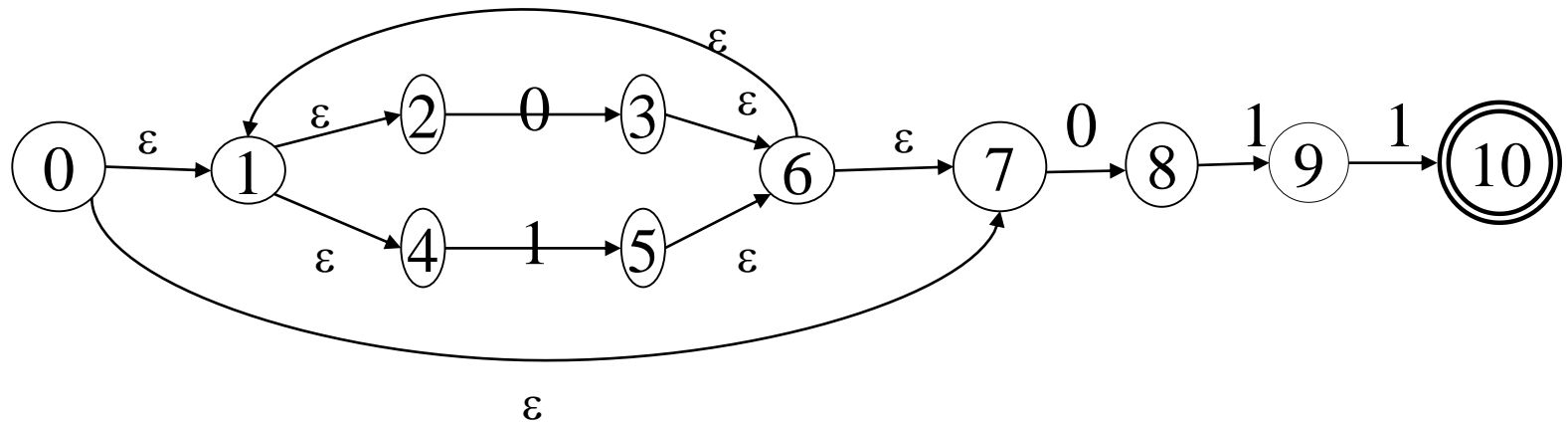
Step 5:

將Transition Table 轉換成Transition Diagram,為所求

演算法:NFA=>DFA

(NFA 中具有 ϵ 符號者)

範例: $(0|1)^*011$ 之NFA,求其DFA



Step 1: DFA 's initial State 為 ϵ -CLOSURE(0):

ϵ -CLOSURE(0)={0,1,2,4,7}

A={0,1,2,4,7} (Marked)

Step 2: State A

Input Symbol 0,

ϵ -CLOSURE({3,8})={1,2,3,4,6,7,8}

B={1,2,3,4,6,7,8}

Input Symbol 1

ϵ -CLOSURE({5})={1,2,4,5,6,7}

C={1,2,4,5,6,7}

Step 3: State B

Step 2:

input 0 $\Rightarrow \epsilon$ -CLOSURE({3,8})=B

input 1 $\Rightarrow \epsilon$ -CLOSURE({5,9})={1,2,4,5,6,7,9}=D

Step 3: State C

Step 2:

input 0 => State B

input 1 => State C

Step 3: State D

Step 2:

input 0 => STATE B

input 1 => ϵ -

CLOSURE({5,10})={1,2,4,5,6,7,10}=E

State E is DFA's Final State.

Step 3: State E

Step 2:

input 0=>State B

input 1=>State C

Step 3: 所有狀態均已標記完成

Step 4: 建立 Transition Table

State		Input Symbol	
		0	1
Initial	A	B	C
	B	B	D
	C	B	C
	D	B	E
Final	E	B	C



演算法:NFA=>DFA

(NFA 中具有 ϵ 符號者)

Step 4: 建立Transition Diagram

演算法:NFA=>DFA

(NFA 中無 ϵ 符號者)

Step 1:若某一狀態對某一輸入有兩種或兩種以上之轉移狀態,則把這數個狀態的集合視為一新狀態.

Step 2:再由新狀態針對每一輸入符號找出其輸出狀態集(Out State Set)

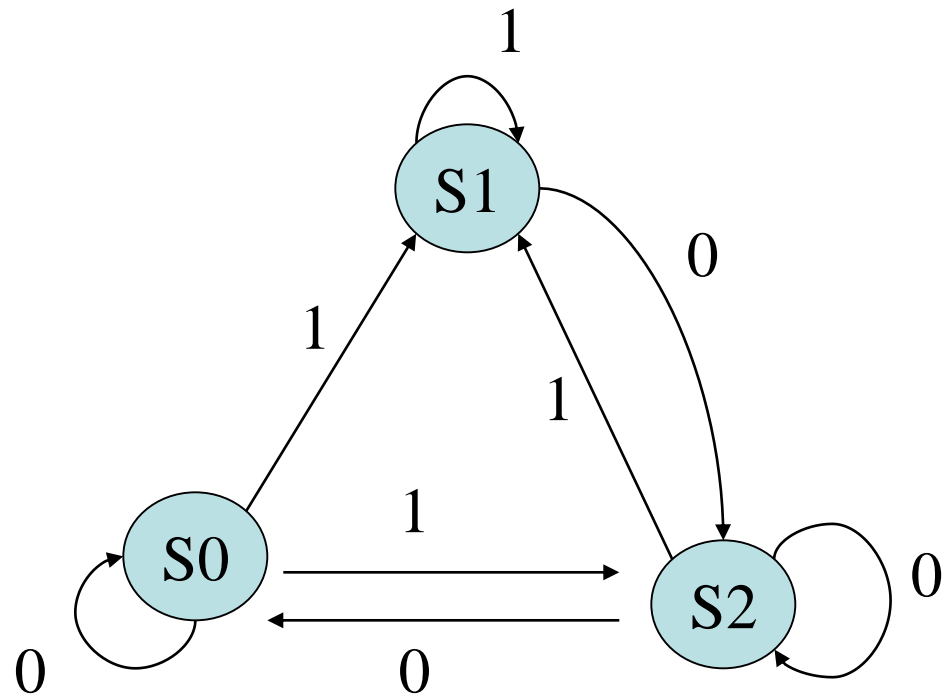
Step 3:將產生新的狀態集重新予以新的狀態編號

Step 4: 重覆step 2,直到無新的狀態產生.

演算法: $NFA \Rightarrow DFA$

(NFA 中無 ε 符號者)

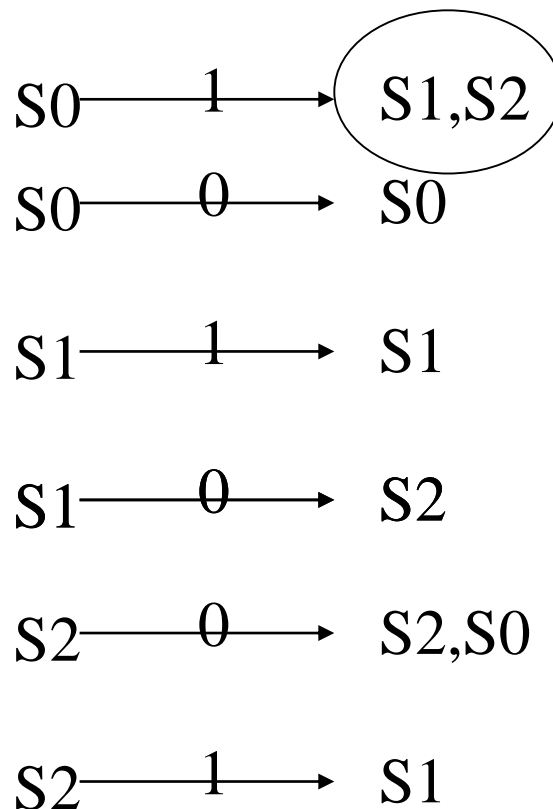
範例: 將下圖NFA 轉換成DFA



演算法:NFA=>DFA

(NFA 中無 ϵ 符號者)

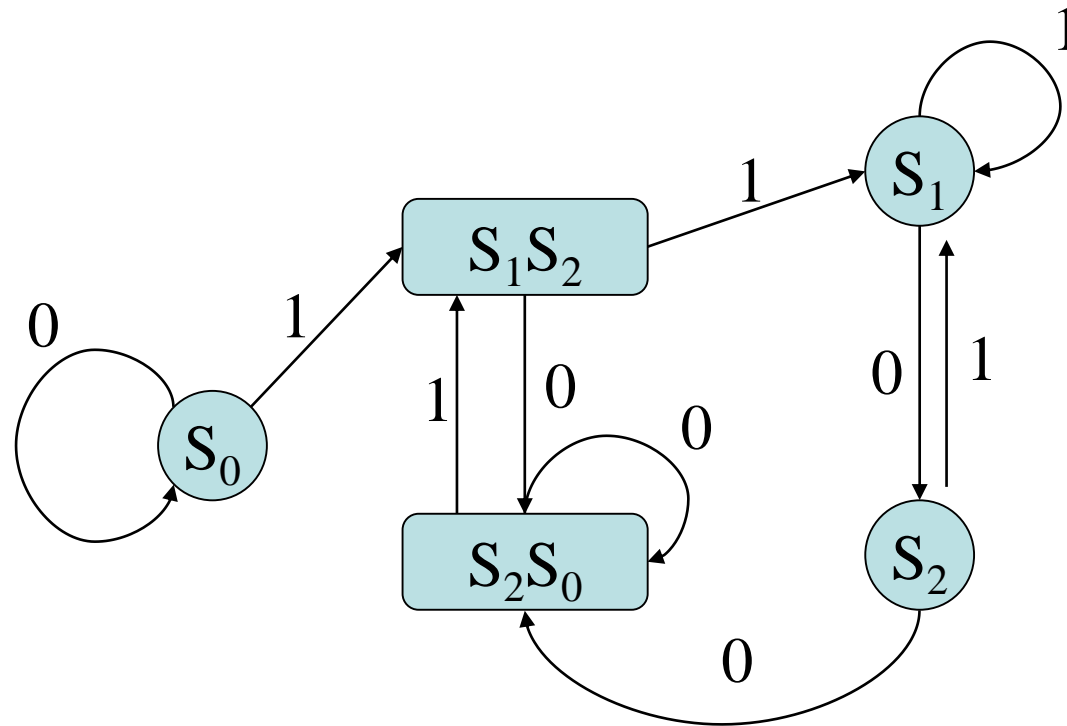
■ Step 1:



演算法:NFA=>DFA

(NFA 中無 ϵ 符號者)

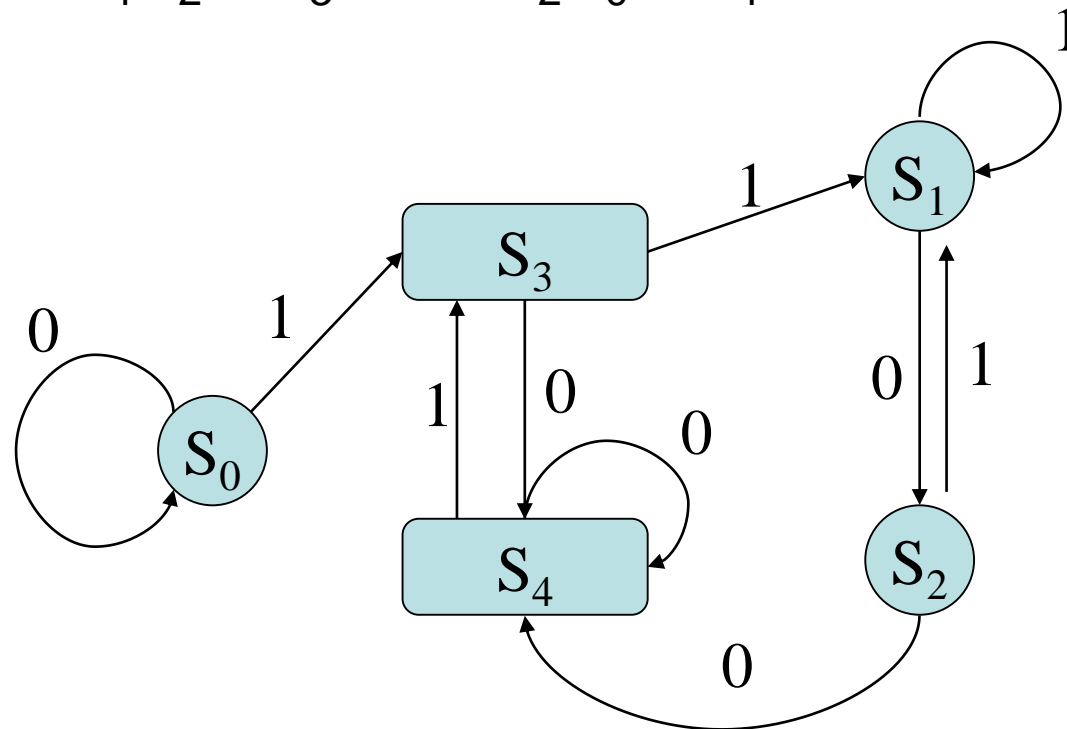
■ Step 2:



演算法:NFA=>DFA

(NFA 中無 ϵ 符號者)

- Step 3 : 將 S_1S_2 以 S_3 代替, S_2S_0 以 S_4 代替



DFA之最小化

演算法:DFA之最小化

Step 1: 將DFA中所有狀態所形成之狀態群(Group)分割成終止狀態群(F)及非終止狀態群(G-F).

Step2: 藉由 **Procedure SPLIT** 對所有的狀態群做分裂.直到所有狀態群皆無法再分裂.

Step3: 令每一狀態群為一個新的狀態(State)

Step4: 刪除所有死亡狀態(Dead State).

所謂死亡狀態即該狀態非初始狀態,且又無法自其他狀態到達之狀態

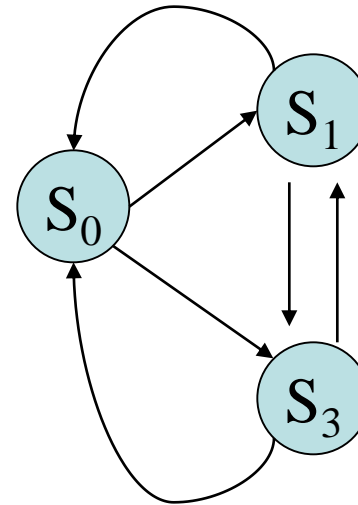
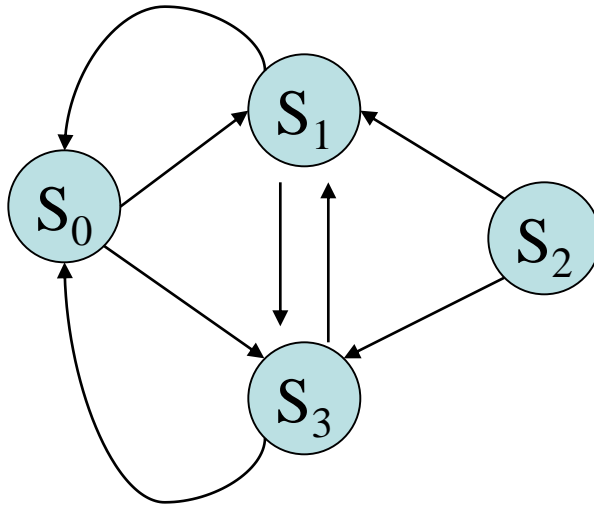
演算法:Procedure SPLIT

■ 處理方式:

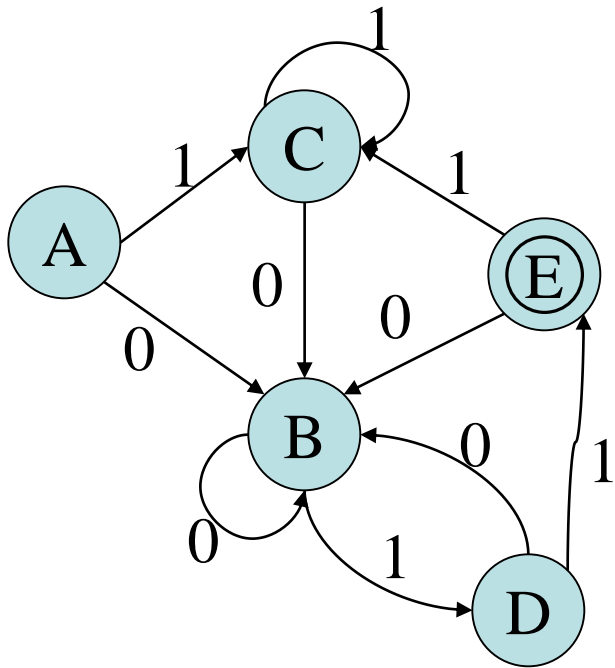
將DFA中之所有狀態對所有輸入之符號做狀態轉移的動作,若任兩個或兩個以上之狀態對於同一輸入符號會轉移至相同之狀態族群中,則將這些狀態合併成另一個新狀態.

範例:死亡狀態之刪除

- 如果 S_0 是 Initial State, 則 S_2 可以被刪除



範例:DFA 之最小化



State		Input Symbol	
		0	1
Initial	A	B	C
	B	B	D
	C	B	C
	D	B	E
Final		B	C

Step 1: 將state A,B,C,D,E分割成非終止狀態群{A,B,C,D}與終止狀態群{E}

Step 2: State {A,B,C,D} 經分裂

input 0 皆轉移至State B

input 1

1. 僅State D 會轉移至{E},

2. State {A,B,C}會轉移至{C,D},

故{A,B,C,D}分裂成{A,B,C}及{D}

再state {A,B,C}

1. input 0皆轉移至State B

2. input 1 State {B}會轉移至{D},

故{A,B,C}分裂成{A,C}及{B}

無法再分裂了

範例:DFA 之最小化

Step 3: 令每一狀態群為一新的狀態

令 $\{A, C\} = S1$

$\{B\} = S2$

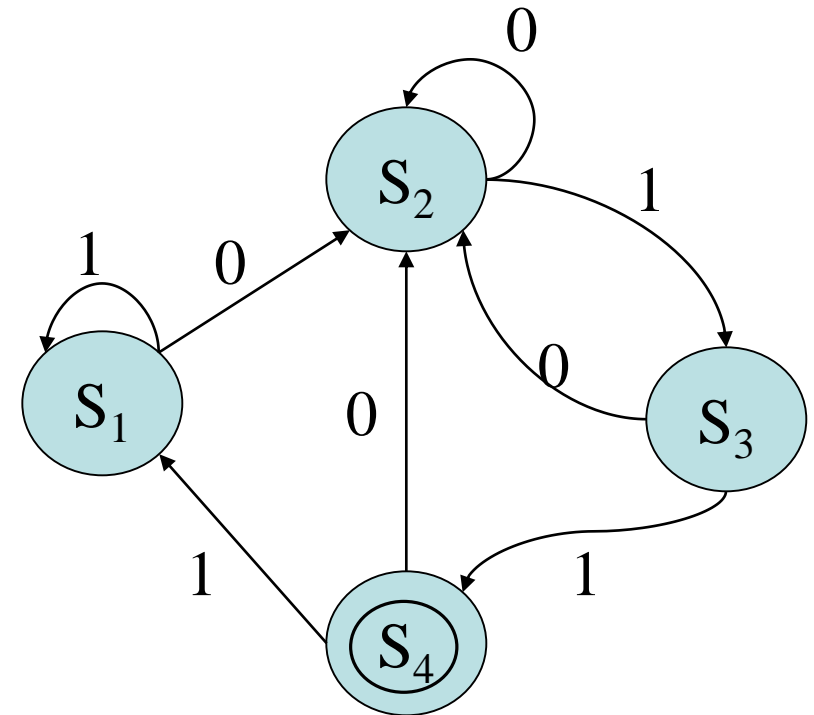
$\{D\} = S3$

$\{E\} = S4$

Step 4: 無任何死亡狀態並由上可得

Transition table & Transition Diagram

State		Input Symbol	
		0	1
Initial	S1	S2	S1
	S2	S2	S3
	S3	S2	S4
Final	S4	S2	S1



有限自動機轉換正規文法

規則有三：

1. State A, input a terminal symbol 'a' and is transited to State B, not final state.

Regular Grammar $A \rightarrow aB$

2. State A, input a terminal symbol 'a' and is transited to State B, Final state.

Regular Grammar $A \rightarrow a, A \rightarrow aB$

3. State A is both an initial State and a Final State.

Regular Grammar $A \rightarrow \lambda$

有限自動機轉換正規文法

Example

$G = \langle N, T, S, P \rangle$,

$N = \{S, A, B, C\}$,

$T = \{0, 1\}$,

$P = \{$

$S \rightarrow \lambda, S \rightarrow 0, S \rightarrow 0A, S \rightarrow 1, S \rightarrow 1B,$

$A \rightarrow 0C, A \rightarrow 1C,$

$B \rightarrow 0C, B \rightarrow 1B, B \rightarrow 1,$

$C \rightarrow 0C, C \rightarrow 1C$

$\}$

