



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e
Statistica
Dipartimento di Informatica

Linguaggi e Compilatori

Autore:
Simone Lidonnici

14 marzo 2025

Indice

E	Esercizi	1
E.1	Esercizi su automi	1
E.1.1	Trasformare un'espressione regolare in NFA	1
E.1.2	Trasformare un NFA in DFA	5
E.1.3	Minimizzare un DFA	8
E.2	Esercizi su grammatiche	11
E.2.1	Eliminare la ricorsione sinistra	11
E.2.2	Fattorizzare una grammatica	13

E

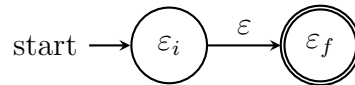
Esercizi

E.1 Esercizi su automi

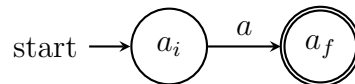
E.1.1 Trasformare un'espressione regolare in NFA

Data un'espressione regolare R e il suo Syntax Tree, possiamo trasformarlo in NFA eseguendo un visita in profondità del Syntax Tree e in base al nodo che visitiamo creiamo un NFA parziale per il suo sottoalbero. Nella spiegazione dell'algoritmo i nodi s_i, s_f indicano il primo e l'ultimo nodo dell'NFA che riconosce s , negli esercizi solitamente si numerano sequenzialmente partendo da 0 in base all'ordine di visita. Eseguendo la visita in profondità l'NFA da creare cambia in base al simbolo contenuto nel nodo visitato:

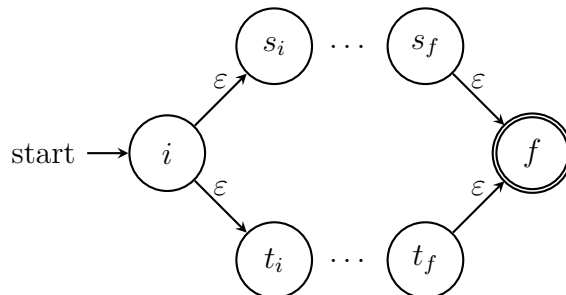
- Se stiamo visitando una foglia contenente ε , costruiremo il seguente NFA:



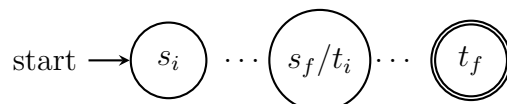
- Se stiamo visitando una foglia contenente un simbolo $a \in \Sigma$, costruiremo il seguente NFA:



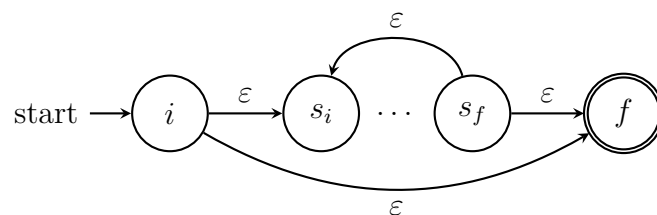
- Se stiamo visitando un nodo contenente un'unione $s \cup t$, costruiremo il seguente NFA, aggiungendo gli stati i e f :



- Se stiamo visitando un nodo contenente una concatenazione st , costruiremo il seguente NFA, unendo lo stato finale di s_f con lo stato iniziale di t_i :



- Se stiamo visitando un nodo contenente una star di Kleene s^* , costruiremo il seguente NFA, aggiungendo gli stati i e f :

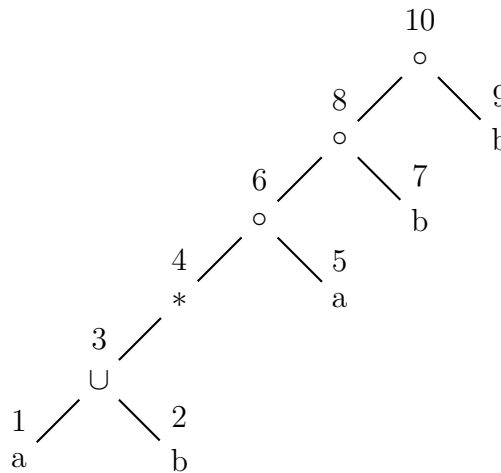


L'NFA risultante avrà un solo stato accettante e ogni stato (ad eccezione di quello accettante) avrà le transizioni uscenti in uno di questi due modi:

1. Una sola transizione con etichetta $a \in \Sigma$
2. Una o due transizioni con etichetta ε

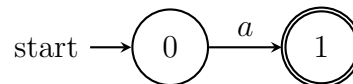
Esempio:

Data l'espressione regolare $(a \cup b)^*abb$ con il seguente Abstract Tree (o indica la concatenazione):

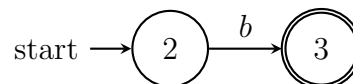


In questo esempio i nodi dell'albero sono numerati in base all'ordine in cui vanno visitati. Eseguiamo i passi dell'algoritmo, partendo dalla foglia in basso a sinistra:

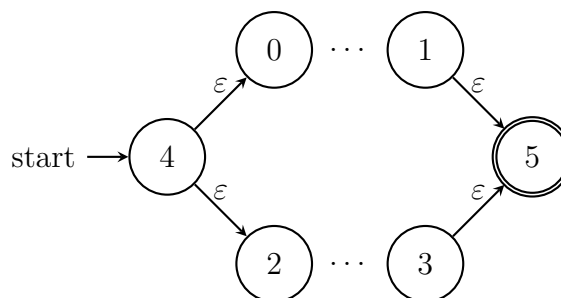
1. La foglia contiene a , quindi l'NFA corrispondente sarà:



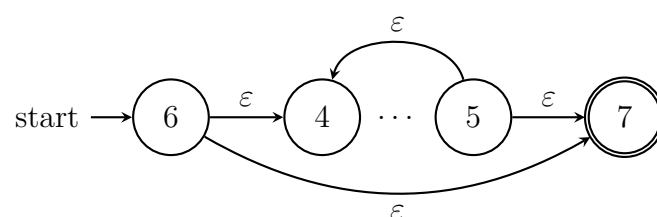
2. La foglia contiene b , quindi l'NFA corrispondente sarà:



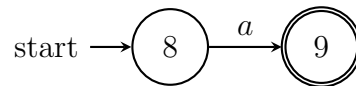
3. La foglia contiene un'unione, quindi l'NFA corrispondente a $a \cup b$ sarà:



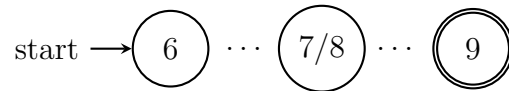
4. La foglia contiene una star di Kleene, quindi l'NFA corrispondente a $(a \cup b)^*$ sarà:



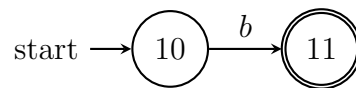
5. La foglia contiene a , quindi l'NFA corrispondente sarà:



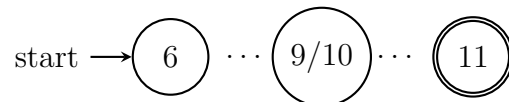
6. La foglia contiene una concatenazione, quindi l'NFA corrispondente a $(a \cup b)^*a$ sarà:



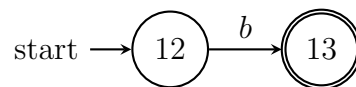
7. La foglia contiene b , quindi l'NFA corrispondente sarà:



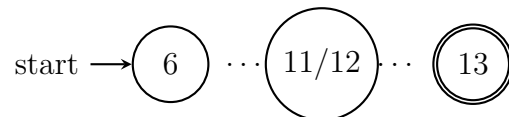
8. La foglia contiene una concatenazione, quindi l'NFA corrispondente a $(a \cup b)^*ab$ sarà:



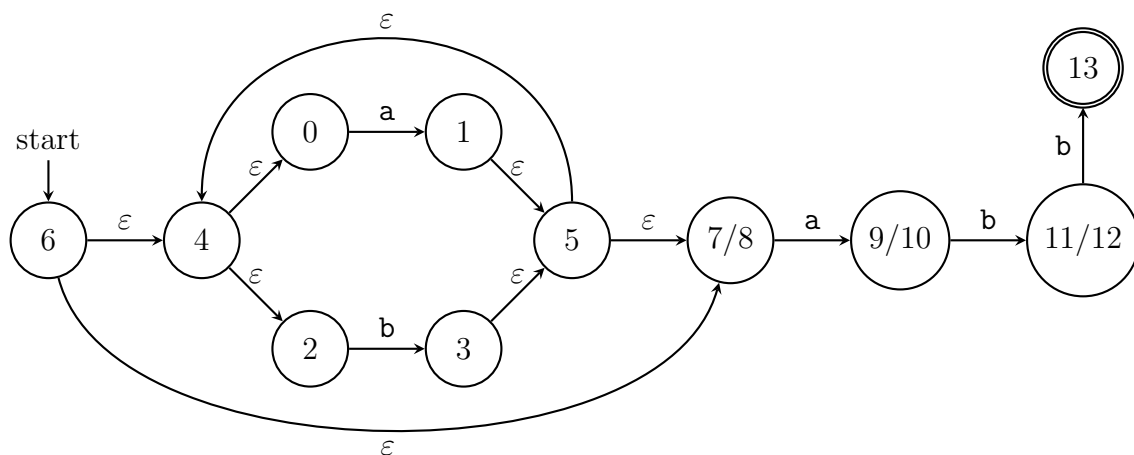
9. La foglia contiene b , quindi l'NFA corrispondente sarà:



10. La foglia contiene una concatenazione, quindi l'NFA corrispondente a $(a \cup b)^*abb$ sarà:



L'automa finale disegnato completamente sarà:



E.1.2 Trasformare un NFA in DFA

Dato un NFA $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ definiamo la ε -closure (o estensione) di uno stato $q \in Q_N$ come:

$$E(q) = \varepsilon\text{-closure}(q) = \left\{ s \in Q_N \mid \begin{array}{l} s \text{ può essere raggiunto da } q \\ \text{tramite solamente } \varepsilon\text{-archi} \end{array} \right\}$$

La ε -closure di un insieme di stati $R \subseteq Q_N$ è definita come:

$$E(R) = \varepsilon\text{-closure}(R) = \bigcup_{q \in R} \varepsilon\text{-closure}(q)$$

La ε -closure di un insieme di stati R contiene sempre almeno R .

Per creare il DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ equivalente all'NFA N , si esegue questo algoritmo:

Algoritmo: Subset Construction

```
def Subset_Construction(N):
     $Q_D = \{\varepsilon\text{-closure}(q_{0_N})\}$ 
    for  $R \in Q_D$  : // anche quelli aggiunti durante il ciclo
        for  $a \in \Sigma$  :
             $S = \varepsilon\text{-closure}(\delta_N(R, a))$ 
            if  $S \notin Q_D$  :
                | Aggiungo  $S$  a  $Q_D$  // Solitamente si numerano con A,B,...,Z
                | Creo la transizione  $\delta_D(R, a) = S$ 
    return D
```

Quando uno stato $R \in Q_D$ è un insieme di stati in Q_N , la funzione δ_N viene calcolata come:

$$\delta_N(R, a) = \bigcup_{r \in R} \delta_N(r, a)$$

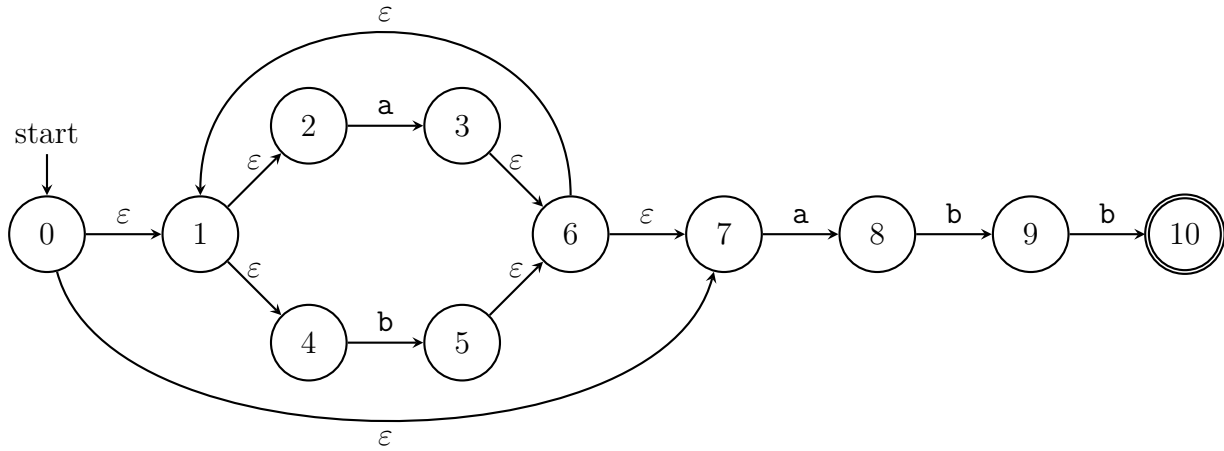
Gli stati accettanti in D sono tutti gli stati che contengono almeno uno stato accettante di N . Il DFA risultante può essere disegnato oppure rappresentato come tabella con le intestazioni:

Stati NFA	Stato DFA	a	b
$\{0,1,2,3\}$	A	B	C
$\{1,2\}$	B	C	A
$\{3,4\}$	C	A	C

Con l'alfabeto $\Sigma = \{a, b\}$ la tabella avrebbe le intestazioni come sopra e la casella nella colonna a e riga A rappresenta la transizione $\delta_D(A, a)$. Nel caso l'alfabeto avesse altri simboli bisognerebbe aggiungere una colonna per ogni simbolo dell'alfabeto.

Esempio:

Dato l'NFA per il linguaggio $L = \{(a \cup b)^*abb\}$:

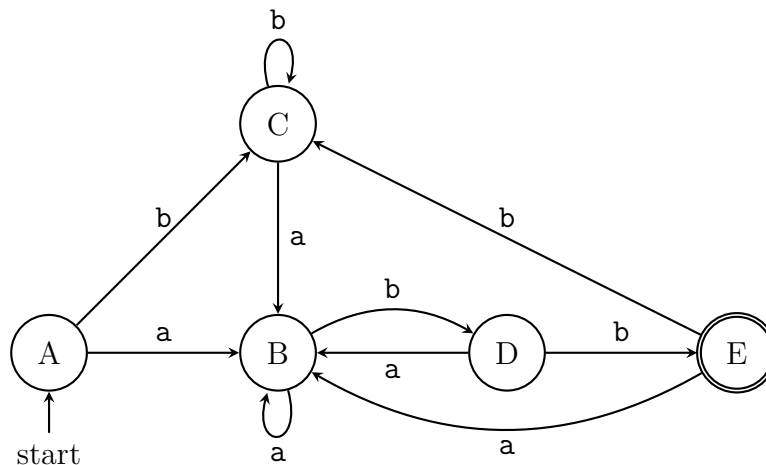


Eseguiamo i passaggi dell'algoritmo:

- Lo stato iniziale di D sarà $\varepsilon\text{-closure}(0) = \{0, 1, 2, 4, 7\} = A$
- Sullo stato A eseguiamo:
 1. $\varepsilon\text{-closure}(\delta_N(A, a)) = \varepsilon\text{-closure}(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$
Avremo una transizione $\delta_D(A, a) = B$
 2. $\varepsilon\text{-closure}(\delta_N(A, b)) = \varepsilon\text{-closure}(\{5\}) = \{1, 2, 4, 5, 6, 7\} = C$
Avremo una transizione $\delta_D(A, b) = C$
- Sullo stato B eseguiamo:
 1. $\varepsilon\text{-closure}(\delta_N(B, a)) = \varepsilon\text{-closure}(\{3, 8\}) = B$
Avremo una transizione $\delta_D(B, a) = B$
 2. $\varepsilon\text{-closure}(\delta_N(B, b)) = \varepsilon\text{-closure}(\{5, 9\}) = \{1, 2, 4, 5, 6, 7, 9\} = D$
Avremo una transizione $\delta_D(B, b) = D$
- Sullo stato C eseguiamo:
 1. $\varepsilon\text{-closure}(\delta_N(C, a)) = \varepsilon\text{-closure}(\{3, 8\}) = B$
Avremo una transizione $\delta_D(C, a) = B$
 2. $\varepsilon\text{-closure}(\delta_N(C, b)) = \varepsilon\text{-closure}(\{5\}) = C$
Avremo una transizione $\delta_D(C, b) = C$
- Sullo stato D eseguiamo:
 1. $\varepsilon\text{-closure}(\delta_N(D, a)) = \varepsilon\text{-closure}(\{3, 8\}) = B$
Avremo una transizione $\delta_D(D, a) = B$
 2. $\varepsilon\text{-closure}(\delta_N(D, b)) = \varepsilon\text{-closure}(\{5, 10\}) = \{1, 2, 4, 5, 6, 7, 10\} = E$
Avremo una transizione $\delta_D(D, b) = E$

- Sullo stato E eseguiamo:
 1. $\varepsilon\text{-closure}(\delta_N(E, a)) = \varepsilon\text{-closure}(\{3, 8\}) = B$
Avremo una transizione $\delta_D(E, a) = B$
 2. $\varepsilon\text{-closure}(\delta_N(E, b)) = \varepsilon\text{-closure}(\{5\}) = C$
Avremo una transizione $\delta_D(E, b) = C$
- Abbiamo finito gli stati da analizzare quindi l'algoritmo è terminato e lo stato accettante di D sarà E perchè è l'unico che contiene lo stato 10 di N .

Il DFA risultante rappresentato sotto forma di automa sarà quindi:



Sotto forma di tabella invece sarà:

Stati NFA	Stato DFA	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 3, 5, 6, 7, 10\}$	E	B	C

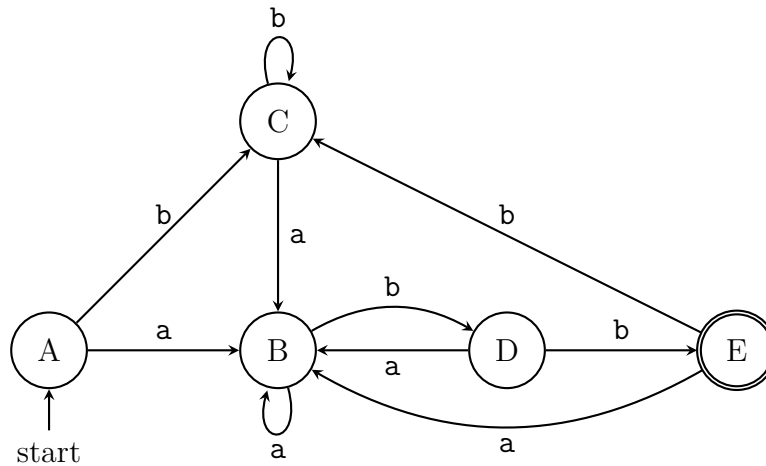
E.1.3 Minimizzare un DFA

Dato un DFA $D = (Q, \Sigma, \delta, q_0, F)$, possiamo minimizzare il numero di stati, creando un DFA equivalente D_{min} seguendo un algoritmo basato sulle partizioni di Q :

1. Partiamo con la partizione $\Pi = \{\{Q - F\}, \{F\}\}$
2. Per ogni insieme $P \in \Pi$ con $|P| \geq 2$ non ancora controllato (compresi quelli aggiunti durante il ciclo):
 - 2.1 Per ogni $a \in \Sigma$:
 - 2.1.1 Per ogni stato $q \in P$, controllo l'insieme P_i a cui appartiene lo stato r tale che $\delta(q, a) = r \in P_i$
 - 2.1.2 Se tutti gli stati $q \in P$ con input a non raggiungono lo stesso insieme P_i allora divido gli stati di P in base all'insieme che raggiungono
 - 2.1.3 Se tutti gli stati $q \in P$ con input a raggiungono lo stesso insieme P_i allora continuo con il prossimo insieme da controllare
3. Dopo aver controllato tutti gli insiemi ed aver trovato Π_{final} , per ogni insieme $P \in \Pi$ scelgo uno stato rappresentante.
4. Lo stato iniziale di D_{min} è lo stato rappresentante dell'insieme P per cui $s_0 \in P$
5. Gli stati finali di D_{min} sono tutti gli stati rappresentanti di insiemi P in cui esiste $q \in F$ tale che $q \in P$

Esempio:

Dato un DFA D non minimizzato:



Eseguiamo l'algoritmo:

1. $\Pi = \{\{A, B, C, D\}, \{E\}\}$

2. Per l'insieme $\{A, B, C, D\}$:

- 2.1 Per $a \in \Sigma$:

- 2.1.1 $\delta(A, a) = B \in \{A, B, C, D\}$

- 2.1.2 $\delta(B, a) = B \in \{A, B, C, D\}$

- 2.1.3 $\delta(C, a) = B \in \{A, B, C, D\}$

- 2.1.4 $\delta(D, a) = B \in \{A, B, C, D\}$

- 2.1.5 Vanno tutti nello stesso insieme, quindi va bene

- 2.2 Per $b \in \Sigma$:

- 2.2.1 $\delta(A, b) = C \in \{A, B, C, D\}$

- 2.2.2 $\delta(B, b) = D \in \{A, B, C, D\}$

- 2.2.3 $\delta(C, b) = C \in \{A, B, C, D\}$

- 2.2.4 $\delta(D, b) = E \in \{E\}$

- 2.2.5 Non vanno tutti nello stesso insieme quindi divido l'insieme aggiungendo a Π i sottoinsiemi $\{A, B, C\}$ e $\{D\}$

3. $\Pi = \{\{A, B, C\}, \{D\}, \{E\}\}$

4. Per l'insieme $\{A, B, C\}$:

- 4.1 Per $a \in \Sigma$:

- 4.1.1 $\delta(A, a) = B \in \{A, B, C\}$

- 4.1.2 $\delta(B, a) = B \in \{A, B, C\}$

- 4.1.3 $\delta(C, a) = B \in \{A, B, C\}$

- 4.1.4 Vanno tutti nello stesso insieme, quindi va bene

4.2 Per $b \in \Sigma$:

4.2.1 $\delta(A, b) = C \in \{A, B, C\}$

4.2.2 $\delta(B, b) = D \in \{D\}$

4.2.3 $\delta(C, b) = C \in \{A, B, C\}$

4.2.4 Non vanno tutti nello stesso insieme quindi divido l'insieme aggiungendo a Π i sottoinsiemi $\{A, C\}$ e $\{B\}$

5. $\Pi = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$

6. Per l'insieme $\{A, C\}$:

6.1 Per $a \in \Sigma$:

6.1.1 $\delta(A, a) = B \in \{B\}$

6.1.2 $\delta(C, a) = B \in \{B\}$

6.1.3 Vanno tutti nello stesso insieme, quindi va bene

6.2 Per $b \in \Sigma$:

6.2.1 $\delta(A, b) = C \in \{A, C\}$

6.2.2 $\delta(C, b) = C \in \{A, C\}$

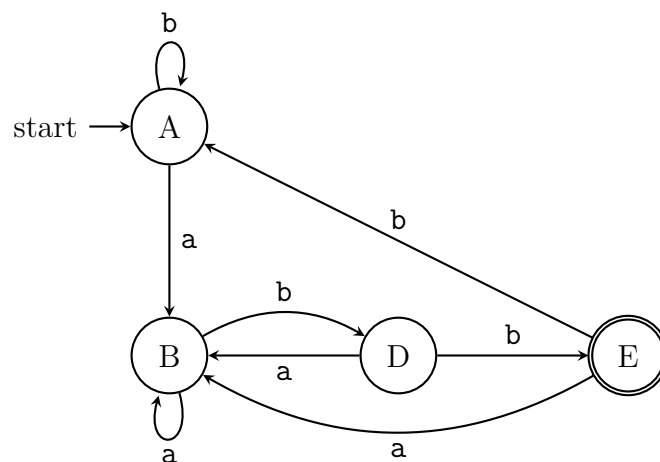
6.2.3 Vanno tutti nello stesso insieme, quindi va bene

7. Abbiamo finito gli insiemi da controllare e quindi $\Pi_{final} = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$

8. Lo stato iniziale è il rappresentante dell'insieme $\{A, C\} = A$

9. Lo stato finale è il rappresentante dell'insieme $\{E\}$

Il DFA minimizzato D_{min} sarà:



E.2 Esercizi su grammatiche

E.2.1 Eliminare la ricorsione sinistra

Una grammatica $G = (V, \Sigma, R, S)$ ha una ricorsione sinistra se:

$$\exists A \in V | A \xRightarrow{+} A\alpha \quad \alpha \in (V \cup \Sigma)^*$$

Il simbolo $\xRightarrow{+}$ indica la derivazione in almeno un passo, se $A \Rightarrow A\alpha$ (in esattamente un passo) allora si dice che la grammatica ha una **ricorsione immediata a sinistra**.

Per eliminare una ricorsione immediata a sinistra nella forma:

$$\begin{aligned} A &\rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n \\ A &\rightarrow \beta_1 | \dots | \beta_m \end{aligned}$$

Per cui $\forall i, \alpha_i, \beta_i \in (V \cup \Sigma)^*$ e per cui il primo simbolo non è A .

Per eliminare la ricorsione immediata a sinistra modifico le regole facendo diventare la grammatica:

$$\begin{aligned} A &\rightarrow \beta_1 A' | \dots | \beta_m A' \\ A' &\rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon \end{aligned}$$

Per eliminare invece la ricorsione a sinistra generica, possiamo utilizzare un algoritmo se la grammatica segue due regole:

1. G non contiene cicli, cioè non esiste $A \xRightarrow{+} A$
2. G non ha regole $A \rightarrow \varepsilon$ (in alcuni casi potrebbe non essere un problema)

L'algoritmo per eliminare la ricorsione sinistra è (considerando $n = |V|$):

Algoritmo: Eliminazione ricorsione a sinistra

```
def Elimina_Ricorsione(G):
    Ordina V
    for i in [1,n] :
        for j in [1,i-1] :
            | Sostituisco le regole  $A_i \rightarrow A_j\alpha$  con  $A_i \rightarrow \beta\alpha$  per cui  $A_j \rightarrow \beta$ 
            | Elimina ricorsione immediata a sinistra se esiste
    return G
```

Esempio:

Data la grammatica G con regole:

$$\begin{aligned} S &\rightarrow Aa|b \\ A &\rightarrow Ac|Sd|\varepsilon \end{aligned}$$

Applichiamo l'algoritmo:

1. Ordiniamo $V = \{S, A\}$
2. Per S non esistono variabili precedenti e non c'è ricorsione immediata
3. Per A :
 - 3.1 Sostituiamo la regola $A \rightarrow Sd$ utilizzando le regole $S \rightarrow Aa|b$ aggiungendo le regole $A \rightarrow Aad|bd$, facendo diventare la grammatica:

$$\begin{aligned} S &\rightarrow Aa|b \\ A &\rightarrow Ac|Aad|bd|\varepsilon \end{aligned}$$

- 3.2 Eliminiamo la ricorsione immediata a sinistra togliendo le regole di A aggiungendo una variabile A' e le regole:

$$\begin{aligned} A &\rightarrow bdA'|A' \\ A' &\rightarrow cA'|adA'|\varepsilon \end{aligned}$$

4. La grammatica diventa quindi:

$$\begin{aligned} S &\rightarrow Aa|b \\ A &\rightarrow bdA'|A' \\ A' &\rightarrow cA'|adA'|\varepsilon \end{aligned}$$

E.2.2 Fattorizzare una grammatica