



SAPIENZA  
UNIVERSITÀ DI ROMA

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

# Linguaggi e Compilatori

**Autore:**  
Simone Lidonnici

22 marzo 2025

# Indice

<b>E</b>	<b>Esercizi</b>	<b>1</b>
E.1	Esercizi su automi . . . . .	1
E.1.1	Trasformare un'espressione regolare in NFA . . . . .	1
E.1.2	Trasformare un NFA in DFA . . . . .	6
E.1.3	Minimizzare un DFA . . . . .	9
E.2	Esercizi su grammatiche . . . . .	12
E.2.1	Eliminare la ricorsione sinistra . . . . .	12
E.2.2	Fattorizzare una grammatica . . . . .	14
E.2.3	Calcolare FIRST e FOLLOW di variabili e stringhe . . . . .	17

# E

## Esercizi

### E.1 Esercizi su automi

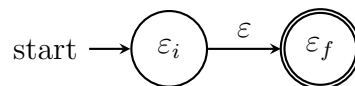
#### E.1.1 Trasformare un'espressione regolare in NFA

Data un'espressione regolare  $R$  con la lista di precedenze delle operazioni e la loro associatività, per prima cosa bisogna creare il suo Syntax Tree. Questo si può fare tramite una semplice operazione ricorsiva partendo dalla radice dell'albero:

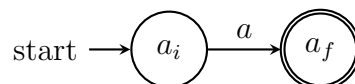
1. Trovare in  $R$  l'operazione eseguita per ultima, cioè quella con meno priorità e che appare nel punto opposto alla sua associatività. Ad esempio se l'operazione con meno priorità è l'unione ( $|$  o  $\cup$ ) ed ha associatività a sinistra l'operazione da scegliere sarà l'unione più a destra.
2. L'operazione scelta divide  $R$  in due sotto-espressioni  $R_1$  e  $R_2$ , cioè i due sottoalberi su cui rieseguire il punto 1.

Possiamo poi 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:

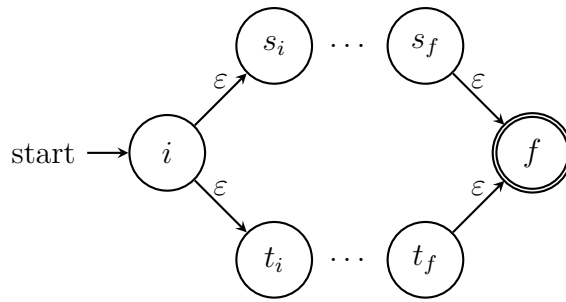
- Se stiamo visitando una foglia contenente  $\varepsilon$ , costruiremo il seguente NFA:



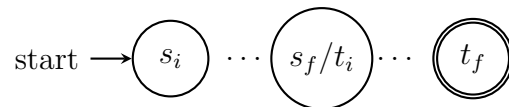
- Se stiamo visitando una foglia contenente un simbolo  $a \in \Sigma$ , costruiremo il seguente NFA:



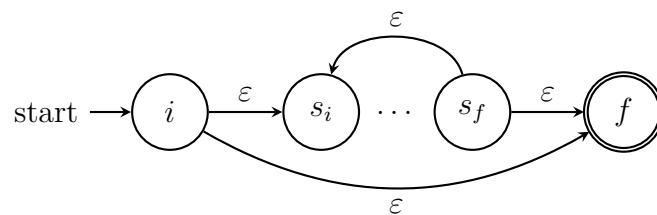
- Se stiamo visitando un nodo contenente un'unione  $s \cup t$ , costruiremo il seguente NFA, aggiungendo gli stati  $i$  e  $f$ :



- 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:

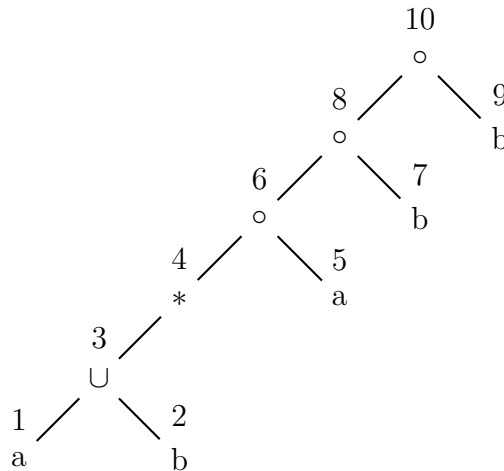
1. Una sola transizione con etichetta  $a \in \Sigma$
2. Una o due transizioni con etichetta  $\varepsilon$

**Esempio:**

Data l'espressione regolare  $(a \cup b)^*abb$  con le seguenti precedenze e associatività (l'operazione più in alto ha precedenza):

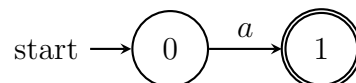
Operazione	Associatività
*	
$\cup$	Sinistra
$\circ$	Sinistra

Il Syntax Tree sarà ( $\circ$  indica la concatenazione):

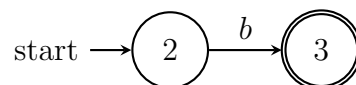


In questo esempio i nodi dell'albero sono numerati in base all'ordine in cui vanno visitati. Eseguiamo i passi dell'algoritmo, partendo dalla foglia in basso a sinistra:

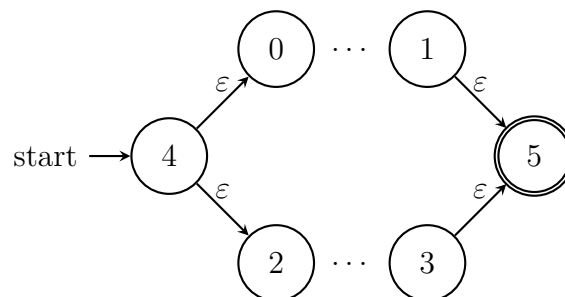
1. La foglia contiene  $a$ , quindi l'NFA corrispondente sarà:



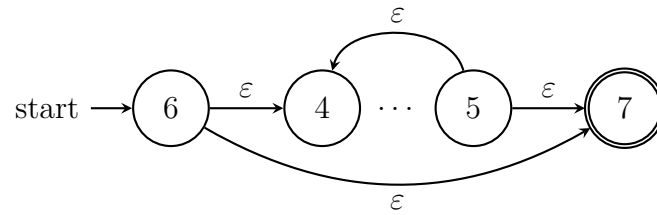
2. La foglia contiene  $b$ , quindi l'NFA corrispondente sarà:



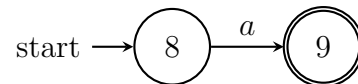
3. La foglia contiene un'unione, quindi l'NFA corrispondente a  $a \cup b$  sarà:



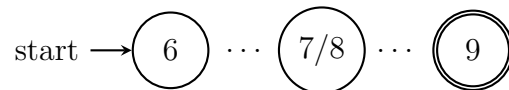
4. La foglia contiene una star di Kleene, quindi l'NFA corrispondente a  $(a \cup b)^*$  sarà:



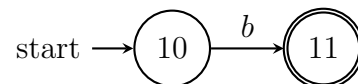
5. La foglia contiene  $a$ , quindi l'NFA corrispondente sarà:



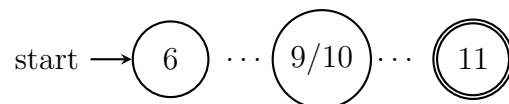
6. La foglia contiene una concatenazione, quindi l'NFA corrispondente a  $(a \cup b)^*a$  sarà:



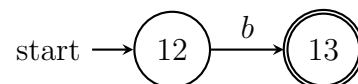
7. La foglia contiene  $b$ , quindi l'NFA corrispondente sarà:



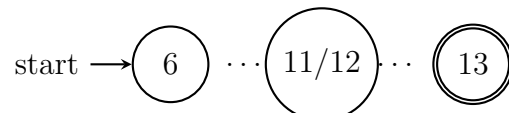
8. La foglia contiene una concatenazione, quindi l'NFA corrispondente a  $(a \cup b)^*ab$  sarà:



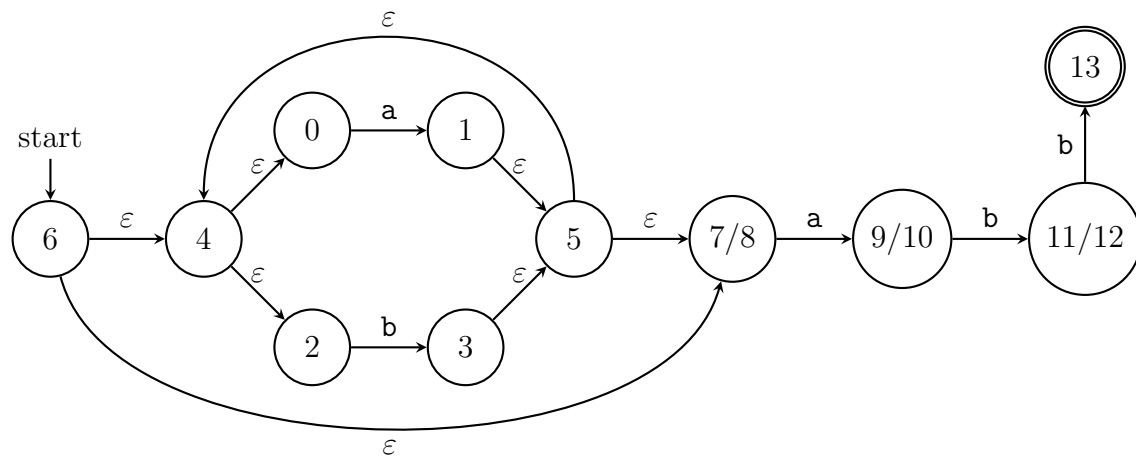
9. La foglia contiene  $b$ , quindi l'NFA corrispondente sarà:



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  $\varepsilon$ -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  $\varepsilon$ -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  $\varepsilon$ -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  $\delta_N$  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:

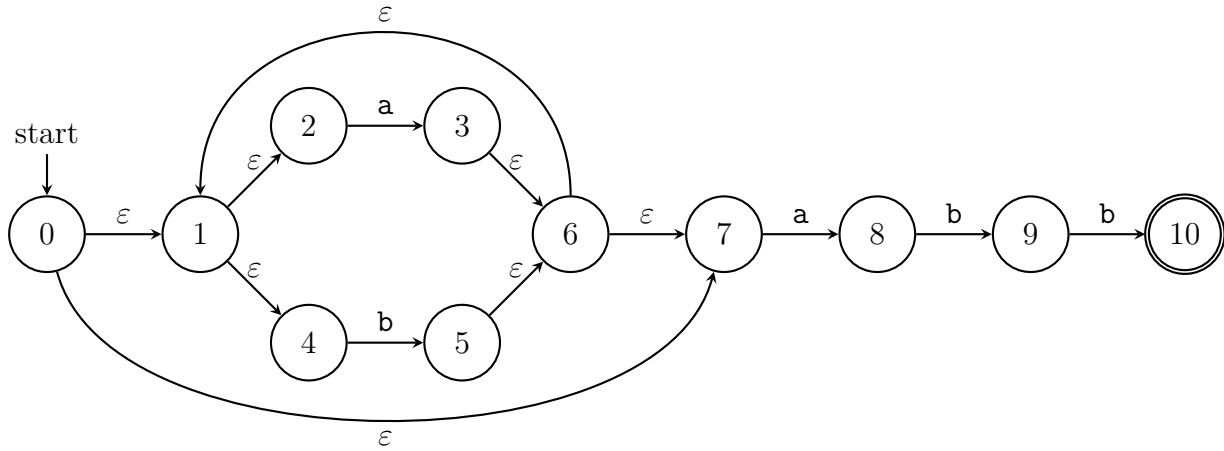
Stati NFA	Stato DFA	a	b
$\{0,1,2,3\}$	A	B	C
$\{1,2\}$	B	C	A
$\{3,4\}$	C	A	C

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  $L = \{(a \cup b)^*abb\}$ :

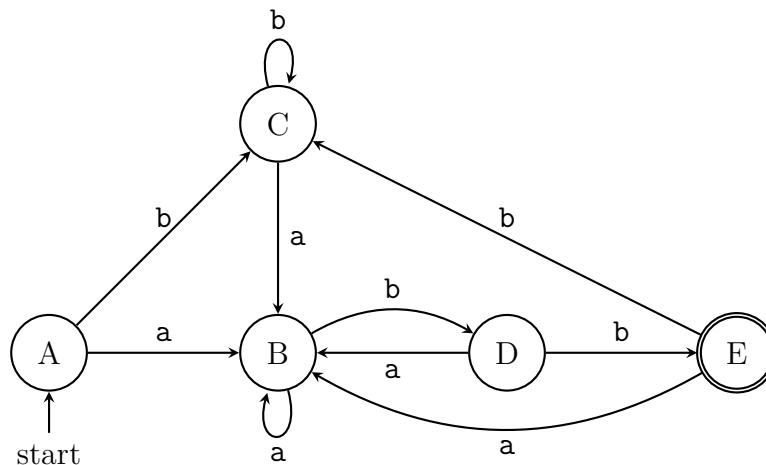


Eseguiamo i passaggi dell'algoritmo:

- 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

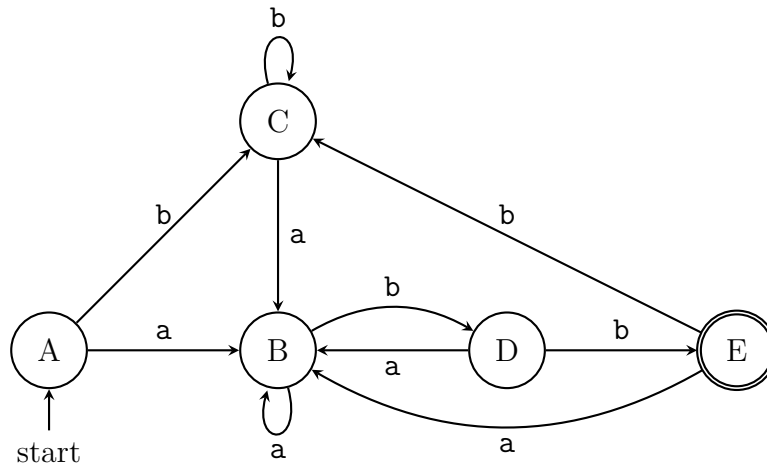
### E.1.3 Minimizzare un DFA

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  $\Pi_{final}$ , 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:

$$1. \Pi = \{\overbrace{\{A, B, C, D\}}^{S-F}, \overbrace{\{E\}}^F\}$$

2. Per l'insieme  $S - F$ :

	$a$	$b$
$A$	$S - F$	$S - F$
$B$	$S - F$	$S - F$
$C$	$S - F$	$S - F$
$D$	$S - F$	$F$

Divido l'insieme aggiungendo a  $\Pi$  i sottoinsiemi  $A = \{A, B, C\}$  e  $B = \{D\}$

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

4. Per l'insieme  $A$ :

	$a$	$b$
$A$	$A$	$A$
$B$	$A$	$B$
$C$	$A$	$A$

Divido l'insieme aggiungendo a  $\Pi$  i sottoinsiemi  $C = \{A, C\}$  e  $D = \{B\}$

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

6. Per l'insieme  $C$ :

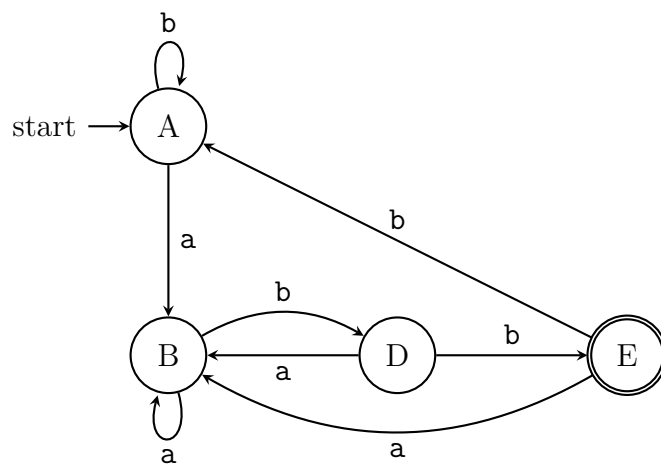
	$a$	$b$
$A$	$D$	$C$
$C$	$D$	$C$

Non ci sono discordanze quindi non divido ulteriormente questo insieme.

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

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

Una grammatica  $G = (V, \Sigma, R, S)$  è fattorizzabile a sinistra se ha delle regole nella forma:

$$A \rightarrow \alpha\beta_1|\alpha\beta_2$$

Per fattorizzarla possiamo usare l'algoritmo:

1.  $\forall A \in V$ :

1.1  $P(A) = \{(A \rightarrow \alpha) \in R\}$

1.2 Scelgo  $P' \subseteq P(A)$  tale che (si possono scegliere anche più di un insieme contemporaneamente sulla stessa variabile  $A$  per eliminare tutti i prefissi):

- Esiste un prefisso  $\pi$  comune a tutte le regole in  $P'$
- $\pi \neq \varepsilon$
- $|P'| \geq 2$
- Non esiste  $P''$  tale che  $P' \subset P''$  e  $P''$  ha le stesse proprietà (1-3) sopra (il prefisso può essere sia il più lungo che quello che compare più volte, conviene considerare sempre il più lungo)

1.3 Se  $P'$  non esiste passiamo alla prossima variabile

1.4 Se  $P'$  esiste creiamo una nuova variabile  $A' \notin V$  e sostituiamo le regole in  $P'$  aggiungendo le regole:

$$\begin{aligned} A &\rightarrow \pi A' \\ A' &\rightarrow \beta_1 | \dots | \beta_n \end{aligned}$$

Per cui  $\beta_1, \dots, \beta_n$  appartengono alle regole  $A \rightarrow \pi\beta_i$ .

2. Se nell'ultimo ciclo la grammatica è cambiata, rieseguiamo il ciclo sulle variabili (comprese quelle aggiunte nel ciclo precedente), sennò abbiamo la grammatica finale fattorizzata.



**Esempio:**

Data una grammatica non fattorizzata  $G$ :

$$\begin{aligned} A &\rightarrow AaCb|AaCa|ABc|BCbB|BAC|BCbC|aa|a \\ B &\rightarrow CbC|CAa|CbA|BAc|BAb|\varepsilon \\ C &\rightarrow CBa|AaC|CBC|BaB|BaA|c \end{aligned}$$

Per fattorizzarla eseguiamo l'algoritmo (per le variabili non scritte si sottointende che non esista un insieme  $P' \subset P(A)$ ):

1. Per  $A$ :1.1 Prendiamo  $P(A)$ :

$$A \rightarrow AaCb|AaCa|ABc|BCbB|BAC|BCbC|aa|a$$

1.2 Scegliamo  $\pi = AaC$  e modifichiamo le regole aggiungendo  $A'$ :

$$\begin{aligned} A &\rightarrow AaCA'|ABc|BCbB|BAC|BCbC|aa|a \\ A' &\rightarrow b|a \end{aligned}$$

1.3 Scegliamo  $\pi = BCb$  e modifichiamo le regole aggiungendo  $A''$ :

$$\begin{aligned} A &\rightarrow AaCA'|ABc|BCbA''|BAC|aa|a \\ A'' &\rightarrow B|C \end{aligned}$$

1.4 Scegliamo  $\pi = a$  e modifichiamo le regole aggiungendo  $A'''$ :

$$\begin{aligned} A &\rightarrow AaCA'|ABc|BCbA''|BAC|aA''' \\ A''' &\rightarrow a|\varepsilon \end{aligned}$$

1.5 Scegliamo  $\pi = B$  e modifichiamo le regole aggiungendo  $A''''$ :

$$\begin{aligned} A &\rightarrow AaCA'|ABc|BA''''|aA''' \\ A'''' &\rightarrow CbA''|AC \end{aligned}$$

1.6 Scegliamo  $\pi = A$  e modifichiamo le regole aggiungendo  $A'''''$ :

$$\begin{aligned} A &\rightarrow AA'''''|BA''''|aA''' \\ A''''' &\rightarrow aCA'|Bc \end{aligned}$$

2. Per  $B$ :2.1 Prendiamo  $P(B)$ :

$$B \rightarrow CbC|CAa|CbA|BAc|BAb|\varepsilon$$

2.2 Scegliamo  $\pi = Cb$  e modifichiamo le regole aggiungendo  $B'$ :

$$\begin{aligned} B &\rightarrow CbB'|CAa|BAc|BAb|\varepsilon \\ B' &\rightarrow C|A \end{aligned}$$

2.3 Scegliamo  $\pi = BA$  e modifichiamo le regole aggiungendo  $B''$ :

$$\begin{aligned} B &\rightarrow CbB'|CAa|BAB''|\varepsilon \\ B'' &\rightarrow c|b \end{aligned}$$

2.4 Scegliamo  $\pi = C$  e modifichiamo le regole aggiungendo  $B'''$ :

$$\begin{aligned} B &\rightarrow CB'''|BAB''|\varepsilon \\ B''' &\rightarrow bB'|Aa \end{aligned}$$

3. Per  $C$ :

3.1 Prendiamo  $P(C)$ :

$$C \rightarrow CBa|AaC|CBC|BaB|BaA|c$$

3.2 Scegliamo  $\pi = CB$  e modifichiamo le regole aggiungendo  $C'$ :

$$\begin{aligned} C &\rightarrow CBC'|AaC|BaB|BaA|c \\ C' &\rightarrow a|C \end{aligned}$$

3.3 Scegliamo  $\pi = Ba$  e modifichiamo le regole aggiungendo  $C''$ :

$$\begin{aligned} C &\rightarrow CBC'|AaC|BaC''|c \\ C'' &\rightarrow B|A \end{aligned}$$

4. Nessun altra variabile ha delle regole con prefissi comuni, quindi l'algoritmo termina.

La grammatica fattorizzata ottenuta è:

$$\begin{aligned} A &\rightarrow AA''''|BA''''|aA''' \\ A' &\rightarrow b|a \\ A'' &\rightarrow B|C \\ A''' &\rightarrow a|\varepsilon \\ A'''' &\rightarrow CbA''|AC \\ A'''' &\rightarrow aCA'|Bc \\ B &\rightarrow CB'''|BAB''|\varepsilon \\ B' &\rightarrow C|A \\ B'' &\rightarrow c|b \\ B''' &\rightarrow bB'|Aa \\ C &\rightarrow CBC'|AaC|BaC''|c \\ C' &\rightarrow a|C \\ C'' &\rightarrow B|A \end{aligned}$$

### E.2.3 Calcolare FIRST e FOLLOW di variabili e stringhe

Data una grammatica  $G$  e una stringa  $\alpha \in (V \cup \Sigma)^*$ , si ha che:

$$\text{FIRST}(\alpha) = \{u \in \Sigma \mid \exists \beta \in (V \cup \Sigma)^* \quad \alpha \xRightarrow{*} u\beta\}$$

Per poter calcolare l'insieme FIRST di una stringa, bisogna prima calcolare gli insiemi FIRST delle variabili e terminali che la compongono. Per un generico  $X \in (V \cup \Sigma)$  si può calcolare  $\text{FIRST}(X)$  con l'algoritmo:

---

**Algoritmo:** Calcolo FIRST di una variabile o terminale

---

```
def Calc_FIRST(X):
    if X ∈ Σ :
        FIRST(X)={X}
    if X ∈ V :
        for (X → Y1...Yk) ∈ R :
            for a ∈ Σ :
                for i in [1,k] :
                    if a ∈ FIRST(Yi) and ε ∈ FIRST(Y1) ∩ ... ∩ FIRST(Yi-1) :
                        FIRST(X)+={a}
                if ε ∈ FIRST(Y1) ∩ ... ∩ FIRST(Yk) :
                    FIRST(X)+={ε}
    return FIRST(X)
```

---

Nel caso di una stringa  $\alpha = X_1 \dots X_n$  si può calcolare  $\text{FIRST}(\alpha)$  con l'algoritmo:

---

**Algoritmo:** Calcolo FIRST di una stringa

---

```
def Calc_FIRST(α):
    FIRST(α)=FIRST(X1)-{ε}
    for i in [2,n] :
        if ε ∈ FIRST(Xi-1) :
            FIRST(α)+=FIRST(Xi)-{ε}
    if ε ∈ FIRST(Xn) :
        FIRST(α)+={ε}
    return FIRST(α)
```

---

Per una variabile  $A \in V$ , si ha che:

$$\text{FOLLOW}(A) = \{u \in \Sigma \mid \exists \alpha, \beta \in (V \cup \Sigma)^* \quad S \xRightarrow{*} \alpha A u \beta\}$$

Si può calcolare l'insieme FOLLOW di una variabile con l'algoritmo:

---

**Algoritmo:** Calcolo FOLLOW di una variabile

---

```
def Calc_FOLLOW(A):
    FOLLOW(S)={$}
    while FOLLOW(A) viene modificato :
        for  $(A \rightarrow \alpha B \beta) \in R$  : // per una qualsiasi A
            FOLLOW(B)+=FIRST( $\beta$ )-{ $\epsilon$ }
            if  $\epsilon \in \text{FIRST}(\beta)$  :
                FOLLOW(B)+=FOLLOW(A)
        for  $(A \rightarrow \alpha B) \in R$  : // per una qualsiasi A
            FOLLOW(B)+=FOLLOW(A)
    return FOLLOW(A)
```

---

**Esempio:**

Data la grammatica  $G$ :

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' | \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' | \varepsilon \\ F &\rightarrow (E) | \text{id} \end{aligned}$$

Possiamo calcolare l'insieme FIRST delle variabili eseguendo l'algoritmo:

- $\text{FIRST}(F) = \{ (, \text{id} \}$
- $\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, \text{id} \}$
- $\text{FIRST}(E') = \{ +, \varepsilon \}$
- $\text{FIRST}(T') = \{ *, \varepsilon \}$

Ora eseguiamo l'algoritmo per calcolare l'insieme FOLLOW delle variabili (per tutte le operazioni non scritte si da per scontato che non modifichino nessun insieme):

$$1. \text{ FOLLOW}(E) = \{ \$ \}$$

2. Ciclo 1:

2.1 Per la regola  $(E \rightarrow TE')$ :

$$2.1.1 \text{ FOLLOW}(T)+ = \text{FIRST}(E') - \{ \varepsilon \}:$$

$$\text{FOLLOW}(T) = \{ + \}$$

$$2.1.2 \varepsilon \in \text{FIRST}(E') \text{ quindi } \text{FOLLOW}(T)+ = \text{FOLLOW}(E):$$

$$\text{FOLLOW}(T) = \{ +, \$ \}$$

$$2.1.3 \text{ FOLLOW}(E')+ = \text{FOLLOW}(E):$$

$$\text{FOLLOW}(E') = \{ \$ \}$$

2.2 Per la regola  $(E' \rightarrow +TE')$ :

2.2.1 Non effettua nessun cambiamento negli insiemi FOLLOW

2.3 Per la regola  $(T \rightarrow FT')$ :

$$2.3.1 \text{ FOLLOW}(F)+ = \text{FIRST}(T') - \{ \varepsilon \}:$$

$$\text{FOLLOW}(F) = \{ * \}$$

$$2.3.2 \varepsilon \in \text{FIRST}(T') \text{ quindi } \text{FOLLOW}(F)+ = \text{FOLLOW}(T):$$

$$\text{FOLLOW}(F) = \{ *, +, \$ \}$$

$$2.3.3 \text{ FOLLOW}(T')+ = \text{FOLLOW}(T):$$

$$\text{FOLLOW}(T') = \{ +, \$ \}$$

2.4 Per la regola  $(T' \rightarrow *FT')$ :

2.4.1 Non effettua nessun cambiamento negli insiemi FOLLOW

2.5 Per la regola  $(F \rightarrow (E))$ :

2.5.1  $\text{FOLLOW}(E)^+ = \text{FIRST}(",") - \{\varepsilon\}$ :

$$\text{FOLLOW}(E) = \{\$, \}$$

3. Ciclo 2:

3.1 Per la regola  $(E \rightarrow TE')$ :

3.1.1  $\varepsilon \in \text{FIRST}(E')$  quindi  $\text{FOLLOW}(T)^+ = \text{FOLLOW}(E)$ :

$$\text{FOLLOW}(T) = \{+, \$, \}$$

3.1.2  $\text{FOLLOW}(E')^+ = \text{FOLLOW}(E)$ :

$$\text{FOLLOW}(E') = \{\$, \}$$

3.2 Per la regola  $(E' \rightarrow +TE')$ :

3.2.1 Non effettua nessun cambiamento negli insiemi FOLLOW

3.3 Per la regola  $(T \rightarrow FT')$ :

3.3.1  $\varepsilon \in \text{FIRST}(T')$  quindi  $\text{FOLLOW}(F)^+ = \text{FOLLOW}(T)$ :

$$\text{FOLLOW}(F) = \{*, +, \$, \}$$

3.3.2  $\text{FOLLOW}(T')^+ = \text{FOLLOW}(T)$ :

$$\text{FOLLOW}(T') = \{+, \$, \}$$

3.4 Per la regola  $(T' \rightarrow *FT')$ :

3.4.1 Non effettua nessun cambiamento negli insiemi FOLLOW

4. Ciclo 3:

4.1 Nessuna regola effettua cambiamenti negli insiemi FOLLOW

5. L'ultimo ciclo non ha fatto cambiamenti agli insiemi FOLLOW quindi l'algoritmo termina

Gli insiemi FOLLOW delle variabili saranno quindi:

- $\text{FOLLOW}(F) = \{*, +, \$, \}$
- $\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{\$, \}$
- $\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{+, \$, \}$