SAPIENZA
UNIVERSITÀ DI ROMA

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

Linguaggi e Compilatori

Autore:
Simone Lidonnici

15 marzo 2025

Indice

E

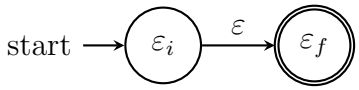
Esercizi

E.1 Esercizi su automi

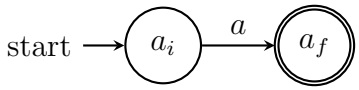
E.1.1 Trasformare un'espressione regolare in NFA

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 creaiamo 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 sequanzialmente 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:

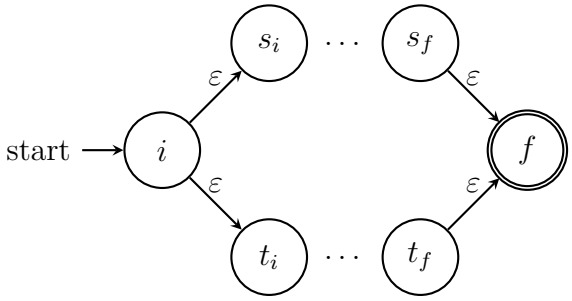
- Se stiamo visitando una foglia contente ε, costruiremo il seguente NFA:



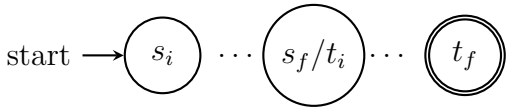
- Se stiamo visitando una foglia contente un simbolo $a \in \Sigma$, costruiremo il seguente NFA:



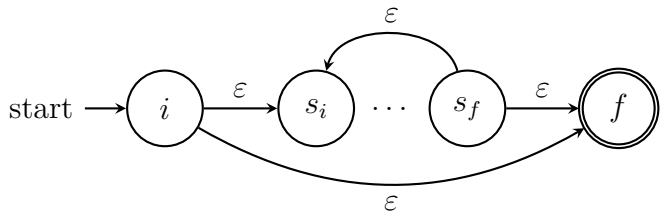
- Se stiamo visitando un nodo contente un'unione $s \cup t$, costruiremo il seguente NFA, aggiungendo gli stati i e f:



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



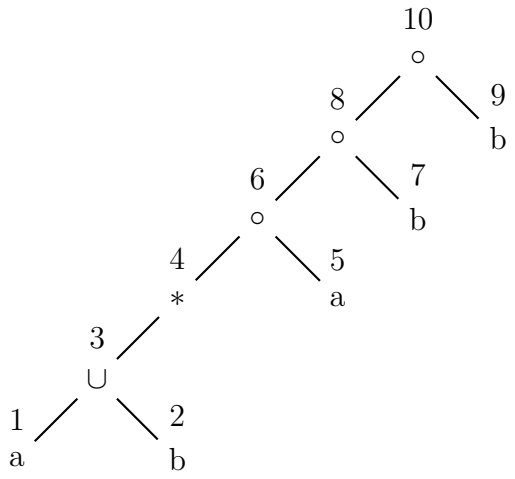
- Se stiamo visitando un nodo contente una star di Kleene s^*, costruiremo il seguente NFA, aggiungendo gli stati i e f:



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:

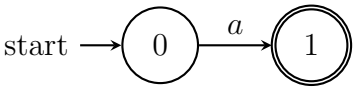
1. Una sola transizione con etichetta $a \in \Sigma$
2. Una o due transizioni con etichetta ε

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

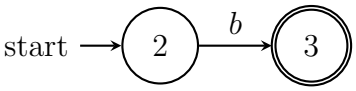


In questo esempio i nodi dell'albero sono numerati in base all'ordine in cui vanno visitati. Eseguiamo i passi dell'algoritmo, partendo dalla foglia in basso a sinistra:

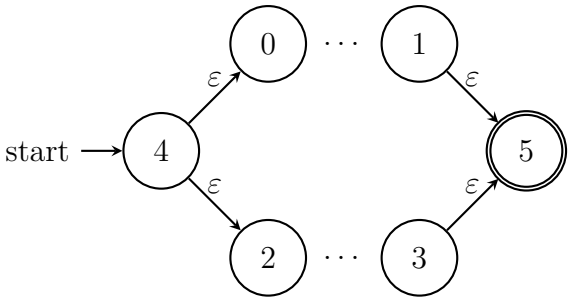
1. La foglia contiene a, quindi l'NFA corrispondente sarà:



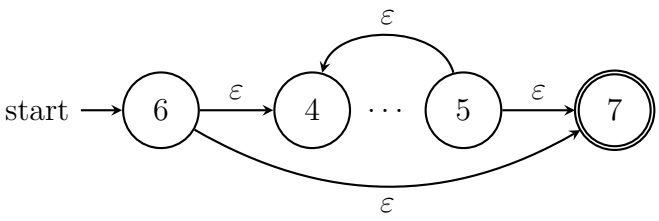
2. La foglia contiene b, quindi l'NFA corrispondente sarà:



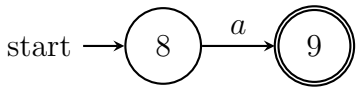
3. La foglia contiene un'unione, quindi l'NFA corrispondente a $a \cup b$ sarà:



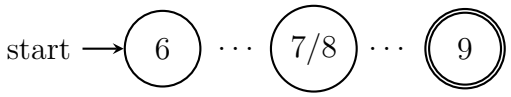
4. La foglia contiene una star di Kleene, quindi l'NFA corrispondente a $(a \cup b)^*$ sarà:



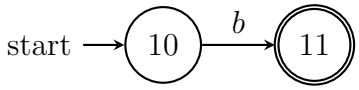
5. La foglia contiene a, quindi l'NFA corrispondente sarà:



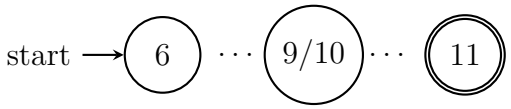
6. La foglia contiene una concatenazione, quindi l'NFA corrispondente a $(a \cup b)^* a$ sarà:



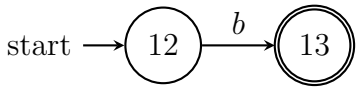
7. La foglia contiene b, quindi l'NFA corrispondente sarà:



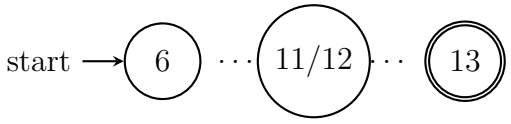
8. La foglia contiene una concatenazione, quindi l'NFA corrispondente a $(a \cup b)^* ab$ sarà:



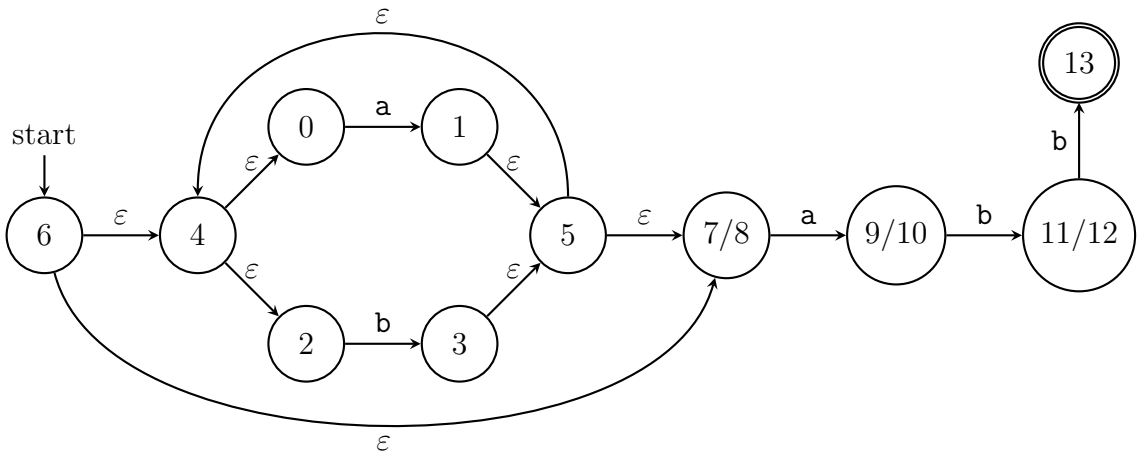
9. La foglia contiene b, quindi l'NFA corrispondente sarà:



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 ε-closure (o estensione) di uno stato $q \in Q_N$ come:

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

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 = {ε-closure(q_0N)}
    for R ∈ Q_D :  // anche quelli aggiunti durante il ciclo
        for a ∈ Σ :
            S = ε-closure(δ_N(R, a))
            if S ∉ Q_D :
                Aggiungo S a Q_D // Solitamente si numerano con A,B,...,Z
            Creo la transizione δ_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:

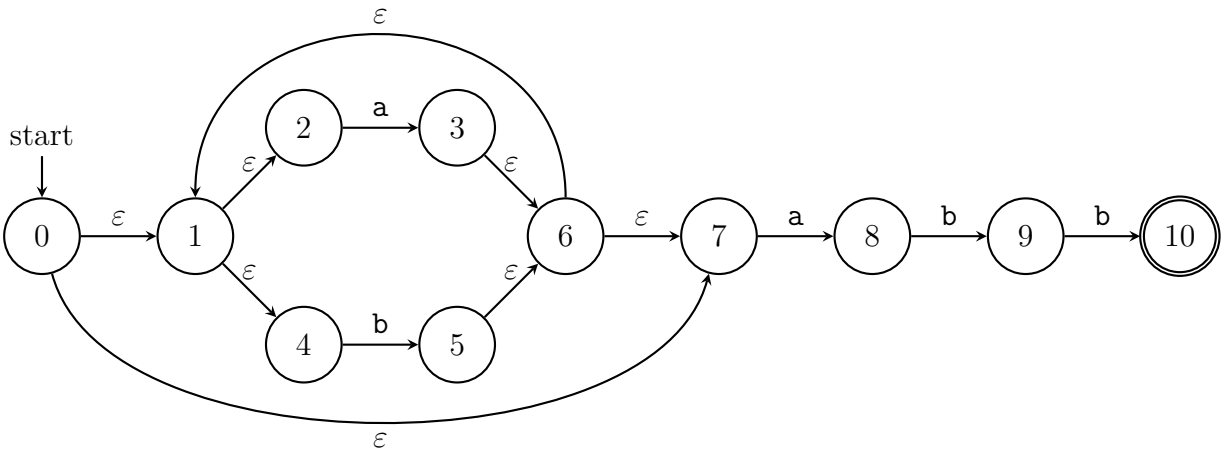
$$\delta_N(R, a) = \bigcup_{r \in R} \delta_N(r, a)$$

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:

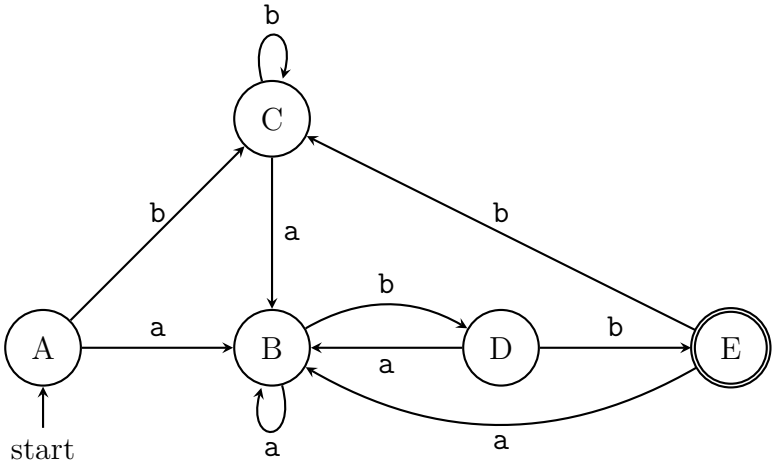
- Lo stato iniziale di D sarà ε-closure$(0) = \{0, 1, 2, 4, 7\} = A$
- Sullo stato A eseguiamo:
  1. ε-closure$(\delta_N(A, a)) = \varepsilon$-closure$(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$
     Avremo una transizione $\delta_D(A, a) = B$
  2. ε-closure$(\delta_N(A, b)) = \varepsilon$-closure$(\{5\})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

- Sullo stato E eseguiamo:

  1. ε-closure$(\delta_N(E, a)) = \varepsilon$-closure$(\{3, 8\}) = B$
     Avremo una transizione $\delta_D(E, a) = B$
  2. ε-closure$(\delta_N(E, b)) = \varepsilon$-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:

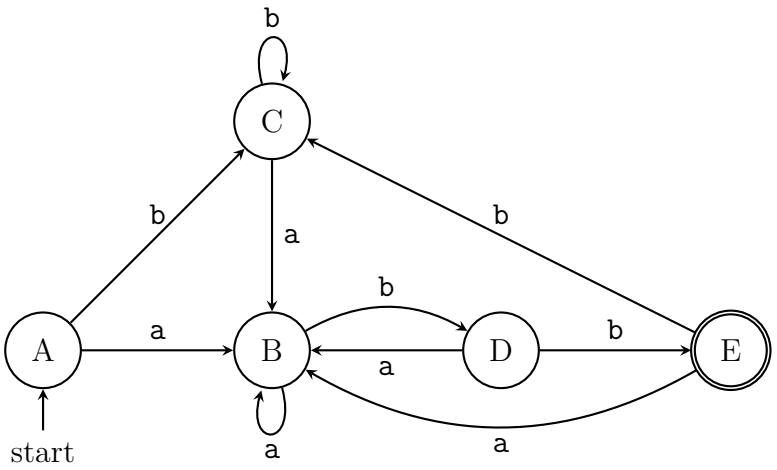
Stati NFA	Stato DFA	a	b
{0,1,2,4,7}	A	B	C
{1,2,3,4,6,7,8}	B	B	D
{1,2,4,5,6,7}	C	B	C
{1,2,4,5,6,7,9}	D	B	E
{1,2,3,5,6,7,10}	E	B	C

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

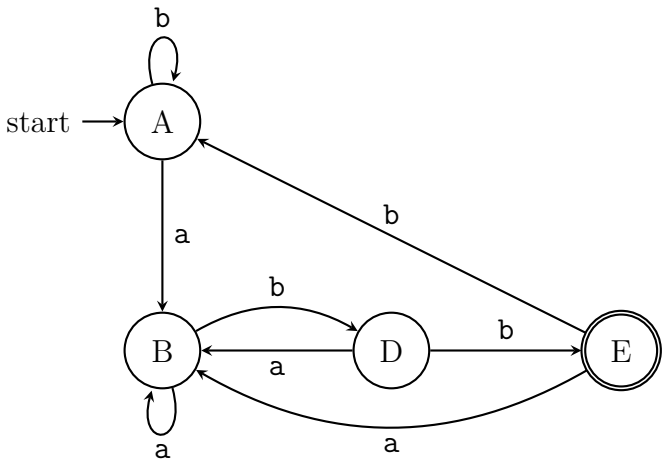
6.2.3 Vanno tutti nello stesso insieme, quindi va bene

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:

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

Il simbolo $\overset{+}{\Rightarrow}$ 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:

$$A \to A\alpha_1 | A\alpha_2 | \dots | A\alpha_n$$
$$A \to \beta_1 | \dots | \beta_m$$

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:

$$A \to \beta_1 A' | \dots | \beta_m A'$$
$$A' \to \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon$$

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 $A \overset{+}{\Rightarrow} A$
2. G non ha regole $A \to \varepsilon$ (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 → A_j α con A_i → βα per cui A_j → β
        Elimina ricorsione immediata a sinistra se esiste
    return G
```

Esempio:
Data la grammatica G con regole:

$$S \to Aa|b$$
$$A \to Ac|Sd|\varepsilon$$

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 $A \to Sd$ utilizzando le regole $S \to Aa|b$ aggiungendo le regole $A \to Aad|bd$, facendo diventare la grammatica:

$$S \to Aa|b$$
$$A \to Ac|Aad|bd|\varepsilon$$

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

$$A \to bdA'|A'$$
$$A' \to cA'|adA'|\varepsilon$$

4. La gramatica diventa quindi:

$$S \to Aa|b$$
$$A \to bdA'|A'$$
$$A' \to cA'|adA'|\varepsilon$$

E.2.2 Fattorizzare una grammatica

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

$$A \to \alpha\beta_1 | \alpha\beta_2$$

Per fattorizzarla possiamo usare l'algoritmo:

1. $\forall A \in V$:
   1.1 $P(A) = \{(A \to \alpha) \in R\}$
   1.2 Scelgo $P' \subseteq P(A)$ tale che:
   - Esiste un prefisso π 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

   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:

   $$A \to \pi A'$$
   $$A' \to \beta_1 | \ldots | \beta_n$$

   Per cui $\beta_1, \ldots, \beta_n$ appartengono alle regole $A \to \pi\beta_i$.
2. Se nell'ultimo ciclo la grammatica è cambiata, rieseguiamo il ciclo sulle variabili, sennò abbiamo la grammatica finale fattorizzata.

Esempio:
Data una grammatica non fattorizzata G:

$$A \to AaCb|AaCa|ABc|BCbB|BAC|BCbC|aa|a$$
$$B \to CbC|CAa|CbA|BAc|BAb|\varepsilon$$
$$C \to CBa|AaC|CBC|BaB|BaA|c$$

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

1. Ciclo 1:

   1.1 Per A:

      1.1.1 Prendiamo $P(A)$:

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

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

      $$A \to AA'|BCbB|BAC|BCbC|aa|a$$
      $$A' \to aCb|aCa|Bc$$

   1.2 Per B:

      1.2.1 Prendiamo $P(B)$:

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

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

      $$B \to CB'|BAc|BAb|\varepsilon$$
      $$B' \to bC|Aa|bA$$

   1.3 Per C:

      1.3.1 Prendiamo $P(C)$:

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

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

      $$C \to CBC'|AaC|BaB|BaA|c$$
      $$C' \to a|C$$

   1.4 La grammatica ottenuta alla fine del ciclo 1 è:

   $$A \to AA'|BCbB|BAC|BCbC|aa|a$$
   $$A' \to aCb|aCa|Bc$$
   $$B \to CB'|BAc|BAb|\varepsilon$$
   $$B' \to bC|Aa|bA$$
   $$C \to CBC'|AaC|BaB|BaA|c$$
   $$C' \to a|C$$

2. Ciclo 2:

2.1 Per A:

2.1.1 Prendiamo $P(A)$:

$$A \to AA'|BCbB|BAC|BCbC|aa|a$$

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

$$A \to AA'|BA''|aa|a$$
$$A'' \to CbB|AC|CbC$$

2.2 Per A':

2.2.1 Prendiamo $P(A')$:

$$A' \to aCb|aCa|Bc$$

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

$$A' \to aCA'''|Bc$$
$$A''' \to b|a$$

2.3 Per B:

2.3.1 Prendiamo $P(B)$:

$$B \to CB'|BAc|BAb|\varepsilon$$

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

$$B \to CB'|BAB''|\varepsilon$$
$$B'' \to c|b$$

2.4 Per B':

2.4.1 Prendiamo $P(B')$:

$$B' \to bC|Aa|bA$$

2.4.2 Scegliamo $\pi = b$ e modifichiamo le regole aggiungendo B''':

$$B' \to bB'''|Aa$$
$$B''' \to C|A$$

2.5 Per C:

2.5.1 Prendiamo $P(C)$:

$$C \to CBC'|AaC|BaB|BaA|c$$

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

$$C \to CBC'|AaC|BaC''|c$$
$$C'' \to B|A$$

2.6 La grammatica ottenuta alla fine del ciclo 2 è:

$$A \to AA'|BA''|aa|a$$
$$A' \to aCA'''|Bc$$
$$A'' \to CbB|AC|CbC$$
$$A''' \to b|a$$
$$B \to CB'|BAB''|\varepsilon$$
$$B' \to bB'''|Aa$$
$$B'' \to c|b$$
$$B''' \to C|A$$
$$C \to CBC'|AaC|BaC''|c$$
$$C' \to a|C$$
$$C'' \to B|A$$

3. Ciclo 3:

3.1 Per A:

3.1.1 Prendiamo $P(A)$:

$$A \to AA'|BA''|aa|a$$

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

$$A \to AA'|BA''|aA''''$$
$$A'''' \to a|\varepsilon$$

3.2 Per A'':

3.2.1 Prendiamo $P(A'')$:

$$A'' \to CbB|AC|CbC$$

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

$$A'' \to CbA'''''|AC$$
$$A''''' \to B|C$$

3.3 La grammatica ottenuta alla fine del ciclo 3 è:

$$A \to AA'|BA''|aA''''$$
$$A' \to aCA'''|Bc$$
$$A'' \to CbA'''''|AC$$
$$A''' \to b|a$$
$$A'''' \to Ba|c$$
$$A''''' \to B|C$$
$$B \to CB'|BAB''|\varepsilon$$
$$B' \to bB'''|Aa$$
$$B'' \to c|b$$
$$B''' \to C|A$$
$$C \to CBC'|AaC|BaC''|c$$
$$C' \to a|C$$
$$C'' \to B|A$$

4. Ciclo 4:

   4.1 Per nessuna variabile esiste un insieme $P' \subset P(A)$, quindi abbiamo finito l'algoritmo.

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:

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

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 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 → Y_1 ... Y_k) ∈ R :
            for a ∈ Σ :
                for i in [1,k] :
                    if a ∈ FIRST(Y_i) and ε ∈ FIRST(Y_1) ∩ ··· ∩ FIRST(Y_{i-1}) :
                        FIRST(X)+={a}
            if ε ∈ FIRST(Y_1) ∩ ··· ∩ FIRST(Y_k) :
                FIRST(X)+={ε}
    return FIRST(X)
```

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

Algoritmo: Calcolo FIRST di una stringa

```
def Calc_FIRST(α):
    FIRST(α)=FIRST(X_1)−{ε}
    for i in [2,n] :
        if ε ∈ FIRST(X_{i-1}) :
            FIRST(α)+=FIRST(X_i)−{ε}
    if ε ∈ FIRST(X_n) :
        FIRST(α)+={ε}
    return FIRST(α)
```

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

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

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 → αBβ) ∈ R :  // per una qualsiasi A
            FOLLOW(B)+=FIRST(β)−{ε}
            if ε ∈ FIRST(β) :
                FOLLOW(B)+=FOLLOW(A)
        for (A → αB) ∈ R :  // per una qualsiasi A
            FOLLOW(B)+=FOLLOW(A)
    return FOLLOW(A)
```

Esempio:
Data la grammatica G:

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

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 $(E \to TE')$:
     - 2.1.1 $\text{FOLLOW}(T) + = \text{FIRST}(E') - \{\varepsilon\}$:
       $$\text{FOLLOW}(T) = \{+\}$$
     - 2.1.2 $\varepsilon \in \text{FIRST}(E')$ 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 $(E' \to +TE')$:
     - 2.2.1 Non effettua nessun cambiamento negli insiemi FOLLOW
   - 2.3 Per la regola $(T \to FT')$:
     - 2.3.1 $\text{FOLLOW}(F) + = \text{FIRST}(T') - \{\varepsilon\}$:
       $$\text{FOLLOW}(F) = \{*\}$$
     - 2.3.2 $\varepsilon \in \text{FIRST}(T')$ 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 ($T' \to *FT'$):

2.4.1 Non effettua nessun cambiamento negli insiemi FOLLOW

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

2.5.1 FOLLOW(E)+ = FIRST(")") − {ε}:

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

3. Ciclo 2:

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

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

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

3.1.2 FOLLOW(E')+ = FOLLOW(E):

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

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

3.2.1 Non effettua nessun cambiamento negli insiemi FOLLOW

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

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

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

3.3.2 FOLLOW(T')+ = FOLLOW(T):

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

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

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 variabii saranno quindi:

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