# LINGUAGGI FORMALI

## Introduzione

Alfabeto : un qualunque insieme di simboli. (Tratteremo solo alfabeti finiti).

Esempio:

{a,b,c,,…,x,w,y,z}
{0.1.2.3.4.5.6.7.8.9}
{0,1}
…

Stringa (su un alfabeto) : una sequenza finita di simboli dell'alfabeto. (Frase, parola).

Esempio:

'stringa'
'1990'
'010001'
ε (stringa vuota)

**Convenzione** :

$$a^0 \underset{def}{=} \varepsilon$$

$$a^{n+1} \underset{def}{=} a^n a$$

Esempio:

εw = w

**Def.** : Σ alfabeto. Le stringhe su Σ si ottengono come segue:

1. ε e' una stringa su Σ;
2. se *x* e' una stringa su Σ, allora *xa* e' una stringa su Σ per ogni $a \in \Sigma$;
3. non c'e' altro modo per definire stringhe.

$\Sigma^*$ denota l'insieme di tutte le stringhe su Σ.

# Operazioni su stringhe

Concatenazione : $x$ , $y$ stringhe $\rightarrow xy$

Esempio:

$x = aaba \qquad y = bcc \qquad \rightarrow \qquad xy = aababcc$

Inversione : $x$ stringa $\rightarrow x^R$

$\varepsilon^R = \varepsilon$

Esempio :

$x = abcde \qquad \rightarrow x^R = edcba$

Prefisso, Suffisso, Sottostringa :

| | |
|---|---|
| $x$ e' prefisso di $xy$ | proprio se $x \neq xy$ |
| $y$ e' suffisso di $xy$ | proprio se $y \neq xy$ |
| $y$ e' sottostringa di $xyz$ | proprio se $y \neq xyz$ |

Lunghezza di una stringa :

$|\cdot| : \Sigma^* \rightarrow \mathrm{N}$

$$\begin{cases} |\varepsilon| = 0 \\ |ax| = 1 + |x| \quad con\ x \in \Sigma^* \end{cases}$$

Esempio :

$|a_1 a_2 ... a_k| = k$

# Linguaggi

- $\Sigma$ alfabeto
  $L \subseteq \Sigma^*$ e' un linguaggio
  Esempio :
  $\varnothing$ , $\{\varepsilon\}$ , $L^*$

- $\Sigma^+ = \Sigma^* - \{\varepsilon\}$
  Esempio :
  $\Sigma = \{0,1\}$
  $L_0 = \{0^n 1^n : n \in \mathrm{N}\}$

**Def.** : L gode della proprieta' prefissa (suffissa) se per ogni $x$ , $y \in$ L , $x$ non e' prefisso (suffisso) proprio di $y$.

Esempio :

$L = \{a^i b \mid i \geq 0\}$

L gode della proprieta' prefissa, ma non suffissa.

# Operazioni sui linguaggi

- $\cup$ . $\cap$ , \ , - (complementazione)

- concatenazione (prodotto) :
  $L_1 \subseteq \Sigma_1^*$ , $L_2 \subseteq \Sigma_2^*$
  $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$

- chiusura (di Kleene) $L^*$
  $L^0 = \{\varepsilon\}$
  $L^{n+1} = LL^n \qquad , n \geq 0$
  $L^* = \bigcup_{n \geq 0} L^n$

- chiusura positiva $L^+ = \bigcup_{n \geq 1} L^n$

**N.B.** $L^+ = LL^* = L^* L$
$L^* = L^+ \cup \{\varepsilon\}$

**Def.** : Siano $\Sigma_1$ , $\Sigma_2$ alfabeti, e sia *h* una applicazione

$h : \Sigma_1 \to \Sigma_2^*$

*h* puo' essere esteso alle stringhe. Si parlera' allora di omomorfismo indotto.

L'omomorfismo indotto da *h* su $\Sigma_1^*$ (ed indicato con *h* stesso) e' dato da:

1. $h(\varepsilon) = \varepsilon$
2. $h(xa) = h(x)h(a) \qquad \forall a \in \Sigma_1$

**Def.** : $h(L) = \{h(w) \mid w \in L\}$

Esempio :

$$h \begin{cases} 0 \to a & \Sigma_1 = \{0,1\} \\ 1 \to bb & \Sigma_2 = \{a,b\} \end{cases}$$

$$L = \{0^n 1^n : n \geq 1\}$$

$$h(L) = \{a^n b^{2n} \mid n \geq 1\}$$

**Def.** : Sia $h : \Sigma_1^* \to \Sigma_2^*$ un omomorfismo.

Si ponga $h^{-1}(y) = \{x \varepsilon \Sigma_1^* \mid h(x) = y\} \qquad (y \varepsilon \Sigma_2^*)$

$h^{-1} : \Sigma_2^* \to pow(\Sigma_1^*)$

e' detto omomorfismo inverso.

**Def.** : $L \subseteq \Sigma_2^*$

$h^{-1}(L) = \bigcup_{y \varepsilon L} h^{-1}(y) = \{x \in \Sigma_1^* \mid h(x) \in L\}$

Esempio :

Sia:

$h(0) = a \quad , \quad h(1) = a$

$h^{-1}(a) = \{0,1\}$

$h^{-1}(a^*) = \{0,1\}^*$

**N.B.** $a^* \equiv \{a\}^*$ per convenzione

Esempio :

$h(0) = a \quad , \quad h(1) = \varepsilon$

$h^{-1}(\varepsilon) = 1^*$

$h^{-1}(a^*) = 1^* 0 1^* \quad (= \{1^i 0 1^j \mid i, j \geq 0\})$

**Esercizio :**

Quali tra I seguenti linguaggi hanno la proprieta' prefissa (suffissa) :

1. $\varnothing$
2. $\{\varepsilon\}$
3. $\{a^n b^n \mid n \geq 1\}$
4. $L^*$ con L avente la proprieta' prefissa
5. $\{w \mid w \in \{a,b\}^* \wedge \#(w) = \#(w)\}$

# Linguaggi (Rappresentazione)

- Sia L un linguaggio su Σ.
- Se L e' finito, L puo' essere rappresentato enumerando tutti I suoi elementi.
- Se L e' infinito, occorrono dei metodi precisi e finiti per specificare gli elementi
  - Metodo generativo (Grammatiche)
  - Metodo riconoscitivo (Riconoscitori (nel senso di procedure)).

# Grammatiche

Grammatiche (di Chomsky) (phrase structure grammar).
(L'utilizzo delle grammatiche ha l'effetto di indurre nelle frasi del linguaggio una struttura molto utile per la traduzione)

**Def.** : G = (N,Σ,P,S)

| | |
|---|---|
| N | (alfabeto, simboli non terminali) |
| Σ | (alfabeto, simboli terminali) |
| $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^*$ | (produzioni) |
| $S \in N$ | (simbolo iniziale) |

(con $N \cap \Sigma = \varnothing$)

**Notazione** : $(\alpha,\beta) \in P$ viene indicata con $\alpha \to \beta$

Esempio 1 :

S→0A1
0A→00A1
A→ε

**Def.** : Forma Proposizionale
Data G = (N,Σ,P,S), le forme proposizionali si definiscono come segue:

1. S e' una forma proposizionale
2. Se $\alpha\beta\gamma$ e' una forma proposizionale e $\beta \to \delta$ e' in P, allora $\alpha\delta\gamma$ e' una forma proposizionale.

Una forma proposionale in $\Sigma^*$ e' detta frase o proposizione generata da G.
Il linguaggio generato da G, L(G), e' l'insieme delle frasi generate da G.

**Simbolismo** :

Derivazione diretta $\underset{G}{\Rightarrow}$

Se $\alpha\beta\gamma \in (N \cup \Sigma)^*$ *e* $\beta \to \delta \in P$ allora $\alpha\beta\gamma \underset{G}{\Rightarrow} \alpha\delta\gamma$

Derivazione non banale $\overset{+}{\underset{G}{\Rightarrow}}$

$\alpha \overset{+}{\underset{G}{\Rightarrow}} \beta$ se e solo se

$\exists \alpha_0, \alpha_1, ..., \alpha_n$ *con* $n \geq 1$ *tali che* $\alpha = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} ... \underset{G}{\Rightarrow} \alpha_{n-1} \underset{G}{\Rightarrow} \alpha_n = \beta$

(chiusura transitive di $\underset{G}{\Rightarrow}$)

Derivazione $\overset{*}{\underset{G}{\Rightarrow}}$

$\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ se e solo se

$\exists \alpha_0, \alpha_1, ..., \alpha_n$ *con* $n \geq 0$ *tali che* $\alpha = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} ... \underset{G}{\Rightarrow} \alpha_{n-1} \underset{G}{\Rightarrow} \alpha_n = \beta$

(chiusura transitiva e riflessiva di $\underset{G}{\Rightarrow}$)

Derivazione di lunghezza *k* $\overset{k}{\underset{G}{\Rightarrow}}$

$\alpha \overset{k}{\underset{G}{\Rightarrow}} \beta$ se e solo se

$\exists \alpha_0, \alpha_1, ..., \alpha_k$ *tali che* $\alpha = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} ... \underset{G}{\Rightarrow} \alpha_{k-1} \underset{G}{\Rightarrow} \alpha_k = \beta$

**N.B.** : $L(G) = \left\{ w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$

Esempio :

$S \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000111$

$S \overset{4}{\underset{G}{\Rightarrow}} 000111$

$S \overset{+}{\underset{G}{\Rightarrow}} 000111$

$S \overset{*}{\underset{G}{\Rightarrow}} 000111$

$000111 \in L(G_1)$

In particolare $L(G_1) = \{0^n 1^n \mid n \geq 1\}$

**Convenzione** : $\alpha \to \beta_1 \mid \beta_2 \mid ... \mid \beta_n$ denota le produzioni
$\alpha \to \beta_1 \quad \alpha \to \beta_2 \quad ... \quad \alpha \to \beta_n$

Esempio :

G: $S \rightarrow aSBC|abC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

$S \Rightarrow aSBC \Rightarrow aabCBC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow aabbcc.$

$L(G) = \{a^n b^n c^n \mid n \geq 1\}$

Esempio :

G: $S \rightarrow CD$

$C \rightarrow aCA|bCB|\varepsilon$

$D \rightarrow \varepsilon$

$AD \rightarrow aD$

$BD \rightarrow bD$

$Aa \rightarrow aA$

$Ab \rightarrow bA$

$Ba \rightarrow aB$

$Bb \rightarrow bB$

$L(G) = \{ww \mid w \in \{a,b\}^*\}$

# Classificazione delle Grammatiche

Sia G = (N,Σ,P,S)

1. Right-Linear : ogni produzione in P ha la forma
   $A \to xB \quad o \quad A \to x \quad con\ A,B \in N\ ,\ x \in \Sigma^*$
2. Context-free : ogni produzione in P ha la forma
   $A \to \alpha \quad con\ A \in N\ ,\ \alpha \in (N \cup \Sigma)^*$
3. Context-sensitive: ogni produzione in P ha la forma
   $\alpha \to \beta \quad con\ \alpha,\beta \in (N \cup \Sigma)^*, |\alpha| \le |\beta|$
   (non si puo' derivare ε)
4. Unrestricted: nessun vincolo.

Right-Linear ⊆ Context-Free
Context-Free \ {ε-produzioni} ⊆ Context-Sensitive
Context-Free \ {ε-produzioni} ⊆ Unrestricted
Context-Sensitive + {ε-produzioni} ⊆ Unrestricted

Esempio :

Right-Linear

$\underline{num} \to 0\ \underline{num} \mid 1\ \underline{num} \mid \ldots \mid 9\ \underline{num} \mid 0 \mid 1 \mid \ldots \mid 9 \mid$

$L(G) = \{0,1,...,9\}^+$

Context-Free

$S \to 0S1 \qquad S \to 01$

$\{0^n 1^n \mid n \ge 1\}$

Context-Sensitive

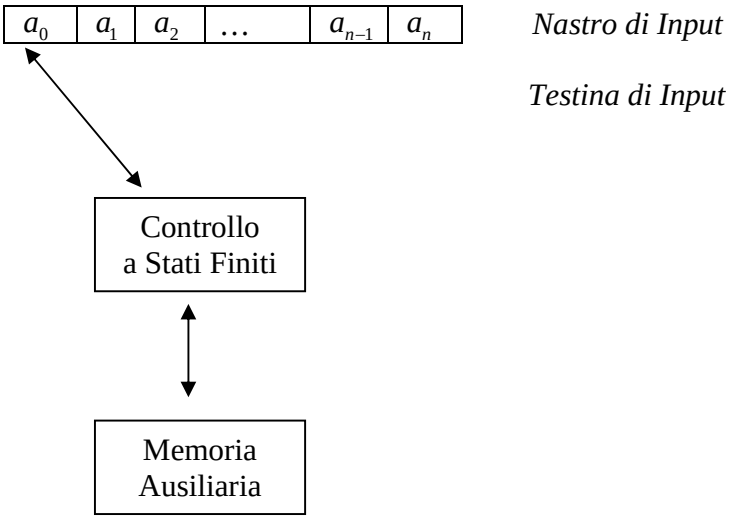
$\{ww \mid w \in \{a,b\}^*, w \ne \varepsilon\}$

$\{a^n b^n c^n \mid n \ge 1\}$

Unrestricted → Linguaggi ricorsivamente enumerabili.

# Riconoscitori

Un riconoscitore e' una procedura per definire un insieme.




Un Riconoscitore consiste di tre componenti principali:

- Nastro di input
- Controllo a stati finiti
- Memoria ausiliaria

} Testina di input (interfaccia)

Nastro di Input :

- Sequenza lineare di locazioni, ove ciascuna locazione puo' contenere un solo simbolo dell'alfabeto di input;
- Le estremita' del nastro possono essere marcate da speciali simboli (**marker**) con tutte le possibili varianti;
- Testina di Input :
  - puo' leggere una locazione alla volta ad un dato istante;
  - puo' muoversi di una locazione a destra o a sinistra, oppure rimanere ferma (se puo' muoversi solo a destra, il riconoscitore si dira' **unidirezionale,** altrimenti **bidirezionale**);
  - solitamente il nastro di input e' **Read-Only**, ma puo' essere anche **Read-Write**.

Memoria ausiliaria :

- Qualunque memoria dati (nastro, stack,…);
- Contiene sequenze finite di simboli da un alfabeto finito di memoria;
- Tipicamente si hanno le funzioni **Store-Fetch**:
  - **Fetch** : applicazione dall'insieme delle possibili configurazioni di memoria ad un insieme finito di simboli di informazione

(Esempio: nel caso di una memoria tipo stack l'operazione Fetch puo' accedere solo al simbolo in cima allo stack).

- **Store** : applicazione dalla memoria + stringa di controllo della memoria (descrive come la memoria potra' essere modificata).
  (Esempio : memoria stack
  $g : \Gamma^+ \times \Gamma^* \to \Gamma^*$ *tale che* $g(Z_1Z_2...Z_n, Y_1Y_2...Y_k) = Y_1Y_2...Y_kZ_1Z_2...Z_n$).

**N.B.** In generale il tipo di un riconoscitore e' determinato dal genere di memoria ausiliaria utilizzata. (Esempio : Automa a Pila (pushdown automaton)).

Controllo a stati finiti :

- Programma per determinare il comportamento del riconoscitore. Consiste in:
  - Un insieme finito di stati;
  - Una applicazione che descrive come gli stati cambiano in funzione del simbolo letto e dell'informazione ottenuta con un'operazione di fetch;

Un riconoscitore opera mediante una sequenza di **mosse**:

- Legge un simbolo in input (dalla testina di input)
- Legge un simbolo di informazione (fetch)
- In base anche allo stato attuale determina quale mossa effettuare.
- Una mossa consiste nelle seguenti operazioni:
  - Muovere a destra o a sinistra la testina di input, ovvero lasciarla ferma;
  - Immagazzinare informazione nella memoria ausiliaria;
  - Cambiare lo stato del controllo.

Il comportamento di un riconoscitore si definisce meglio in termini di configurazioni del riconoscitore.

Configurazione : descrizione dello stato del riconoscitore.

Tripla:

- Stato del controllo finito;
- Contenuto del nastro di input e posizione della testina di input;
- Contenuto della memoria.

Configurazione iniziale :

- Stato del controllo : iniziale;
- Testina di input : posizionata sul carattere piu' a sinistra;
- Memoria : predefinita.

Configurazione finale :

- Stato del controllo : su uno dei possibili stati finali
- Testina di input : oltre il contenuto del nastro di input
- Memoria : soddisfa certe condizioni.

Il controllo finito di un riconoscitore puo' essere **Deterministico** oppure **Non Deterministico**:

- E' **deterministico** quando da ogni configurazione si puo' passare ad al piu' una configurazione.

- E’ **non deterministico** quando vi sono delle configurazioni per le quali ci sono piu’ mosse possibili.

Un riconoscitore **accetta una stringa *w* di input** se partendo dalla configurazione iniziale con *w* nel nastro di input c’e’ una sequenza di mosse che porta ad una configurazione finale.

Il **linguaggio definito da un riconoscitore** e’ l’insieme delle parole accettate dal riconoscitore stesso.

Per ogni classe di grammatiche di Chomsky c’e’ una classe di riconoscitori che definisce la stessa classe di linguaggi:

1. Right-Linear $\rightarrow$ Automa a Stati Finiti (unidir. , det.);
2. Context-Free $\rightarrow$ Automa a Pila (unidir. , non det.);
3. Context-Sensitive $\rightarrow$ Automa Linear Bounded (bidir. , non det.);
4. Unrestricted $\rightarrow$ Macchina di Turing (bidir, , det.).

# Insiemi Regolari

Espressioni Regolari.
Sia Σ un alfabeto. Le espressioni regolari su Σ sono definite ricorsivamente come segue:

1. $\underline{\varnothing}$ e' una espressione regolare
2. $\underline{\varepsilon}$ e' una espressione regolare
3. $\underline{a}$ e' una espressione regolare
4. se α , β sono espressioni regolari tali sono: $(\alpha \cup \beta)$ , $(\alpha \bullet \beta)$ , $\alpha^*$
5. un'espressione e' regolare se cio' puo' essere dedotto da (1)-(4).

Esempio :

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

$$\underline{0}^* \quad , \quad \underline{0}^* \underline{1}^* \underline{0}^* \quad , \quad (\underline{01}) \cup (\underline{101})^* \quad , \quad \ldots$$

Alle espressioni regolari possono essere associati degli insiemi, in maniera naturale:
$\alpha \to \langle \alpha \rangle \subseteq \Sigma^*$.

$$\langle \underline{\varnothing} \rangle \to \varnothing$$

$$\langle \underline{\varepsilon} \rangle \to \{\varepsilon\}$$

$$\langle \underline{a} \rangle \to \{a\} \quad , \quad \forall a \in \Sigma$$

$$\langle (\alpha \cup \beta) \rangle \to \langle \alpha \rangle \cup \langle \beta \rangle$$

$$\langle (\alpha \bullet \beta) \rangle \to \langle \alpha \rangle \bullet \langle \beta \rangle$$

$$\langle \alpha^* \rangle \to \langle \alpha \rangle^*$$

**Def.** : Un linguaggio $R \subseteq \Sigma^*$ dicesi regolare se esiste un'espressione regolare α su Σ tale che $R = \langle \alpha \rangle$.

Esempio :

$$\langle \underline{01} \rangle = \{01\}$$

$$\langle \underline{0}^* \rangle = \{0\}^* = \{0^n \mid n \in \mathrm{N}\}$$

$$\langle (\underline{0} \cup \underline{1})^* \rangle = \{0,1\}^*$$

$$\langle (\underline{0} \cup \underline{1})^* \underline{011} \rangle = \{w \in \{0,1\}^* \mid w \textit{ ha suffisso } 011\}$$

...

Ovviamente gli insiemi regolari sono **chiusi** rispetto alle operazioni di ∪, • (concatenazione), * (chiusura).

# Automi a Stati Finiti

Un automa a stati finiti puo' essere considerato come un riconoscitore senza memoria memoria ausiliaria, unidirezionale (ad ogni mossa la testina di input non puo' restare ferma), deterministico, con nastro di input read-only.

**Def.** : Formalmente un automa finito *m* e' denotato da una quintupla $(Q,\Sigma,\delta,q_1,F)$ dove:

- insieme di stati $Q=\{q_1,...,q_n\}$
- alfabeto $\Sigma=\{s_1,...,s_n\}$
- funzione $\delta: Q\times\Sigma\to Q$ funzione di transizione
- insieme $F\subseteq Q$ stati finali
- stato $q_1\in Q$ stato iniziale

La funzione δ induce una funzione

$$\delta^*: Q\times\Sigma^*\to Q$$

tale che:

1. $\delta^*(q_i,\varepsilon)=q_i$ (questa condizione non permette di cambiare stato senza leggere un simbolo di input).
2. $\delta^*(q_i,us_j)=\delta(\delta^*(q_i,u),s_j)$ *con* $u\in\Sigma^*$

La (2) ci dice in che stato ci troviamo dopo aver letto la stringa non vuota $us_j$ : trova lo stato $p=\delta^*(q_i,u)$ dopo aver letto *u* e poi calcola $\delta^*(q_i,s_j)$.

**Nota**: Poiche' $\delta^*(q,a)=\delta(\delta^*(q,\varepsilon),a)=\delta(q,a)$

dalla (2) con *w*=ε

non ci e' disaccordo fra $\delta$ e $\delta^*$ relativamente agli argomenti per i quali entrambe sono definite. Si puo' quindi anche usare $\delta$ al posto di $\delta^*$.

*m* **accetta** *u* se $\delta^*(q_1,u)\in F$
*m* **rigetta** *u* se $\delta^*(q_1,u)\notin F$

**Def.** : Linguaggio accettato da *m* : $L(m)=\{u\in\Sigma^*\mid\delta^*(q_1,u)\in F\}$.

**Def.** :Un linguaggio e' AF-regolare se e' accettato da qualche automa finito *m*.

Ad un automa finito puo' essere associato in modo del tutto naturale un grafo etichettato , detto diagramma di transizione.

Esempio :

$m$ :

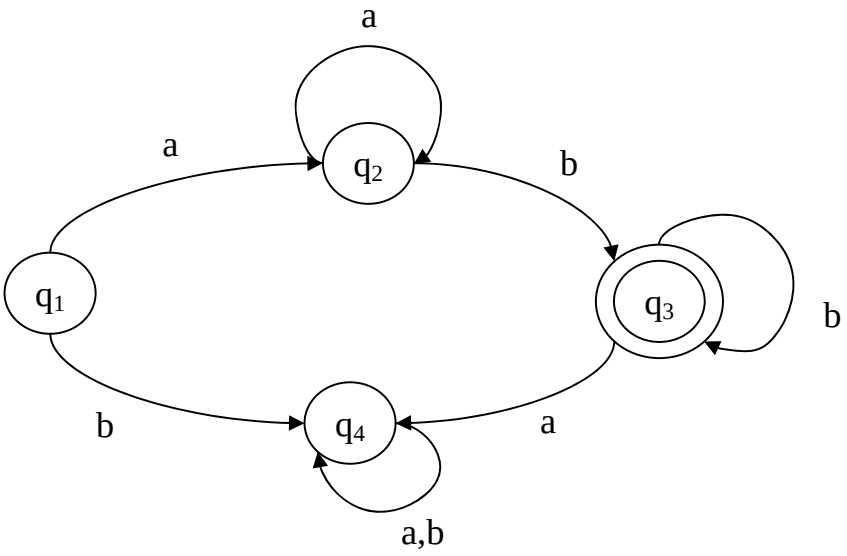
$\Sigma = \{a,b\}$

$Q = \{q_1,q_2,q_3,q_4\}$

$F = \{q_3\}$

| $\delta$ | a | b |
|---|---|---|
| $q_1$ | $q_2$ | $q_4$ |
| $q_2$ | $q_2$ | $q_3$ |
| $q_3$ | $q_4$ | $q_3$ |
| $q_4$ | $q_4$ | $q_4$ |

Il diagramma di transizione e' il seguente:




$L(m) = \{a^n b^m \mid n,m > 0\}$

Non tutti i linguiaggi sono AF-regolari:

Esempio :

$L = \{a^n b^n\}$ non e' AF-regolare

# Automi finiti non deterministici

Il concetto di non determinismo e' importante per due ragioni:

1. gli automi a stati finiti non deterministici (NFA) rendono piu' semplice molte dimostrazioni;
2. in genere non c'e' corrispondenza fra determinismo e non determinismo (ad esempio negli automi a pila).

**Def.** : Un Automa finito non deterministico $m$ e' denotato da una quintupla $(Q,\Sigma,\delta,q_1,F)$ dove:

- insieme di stati $Q=\{q_1,...,q_n\}$
- alfabeto $\Sigma=\{s_1,...,s_n\}$
- funzione $\delta: Q\times\Sigma\rightarrow pow(Q)$ funzione di transizione
- insieme $F\subseteq Q$ stati finali
- stato $q_1\in Q$ stato iniziale

La funzione δ induce una funzione

$$\delta^*: Q\times\Sigma^*\rightarrow pow(Q)$$

tale che:

1. $\delta^*(q_i,\varepsilon)=\{q_i\}$
2. $\delta^*(q_i,us_j)=\bigcup_{q\in\delta^*(q_i,u)}\delta(q,s_j)$ $\quad con\ s_j\in\Sigma, q_i\in Q$

$m$ **accetta** $u$ se $\delta^*(q_1,u)\cap F\neq\varnothing$

**Def.** : Linguaggio accettato da $m$ : $L(m)=\{u\in\Sigma^* \mid \delta^*(q_1,u)\cap F\neq\varnothing\}$.

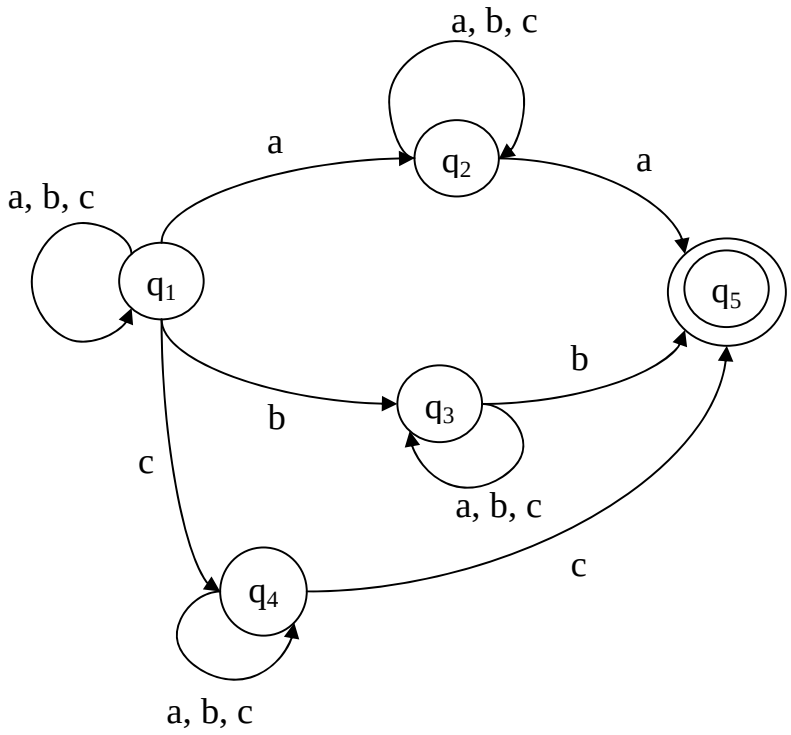
Esempio :

$m = (\{q_1,q_2,q_3,q_4,q_5\},\{a,b,c,\},\delta,q_1,\{q_5\})$

La tabella di transizione e' la seguente:

| $\delta$ | a | b | c |
|---|---|---|---|
| $q_1$ | $\{q_1,q_2\}$ | $\{q_1,q_3\}$ | $\{q_1,q_4\}$ |
| $q_2$ | $\{q_2,q_5\}$ | $\{q_2\}$ | $\{q_2\}$ |
| $q_3$ | $\{q_3\}$ | $\{q_3,q_5\}$ | $\{q_3\}$ |
| $q_4$ | $\{q_4\}$ | $\{q_4\}$ | $\{q_4,q_5\}$ |
| $q_5$ | $\varnothing$ | $\varnothing$ | $\varnothing$ |

Il diagramma di transizione e' il seguente:




$L(m) = \{x : x = ys \quad ed\ s\ appare\ in\ y\}$

*L(m)* e' l'insieme delle parole in $\{a,b,c\}^*$ tali che l'ultimo simbolo appare anche nella prima parte della stringa.

Calcoliamo alcuni valori:

$$\delta^*(q_1, aca) = \bigcup_{q \in \delta^*(q_1, ac)} \delta(q, a)$$

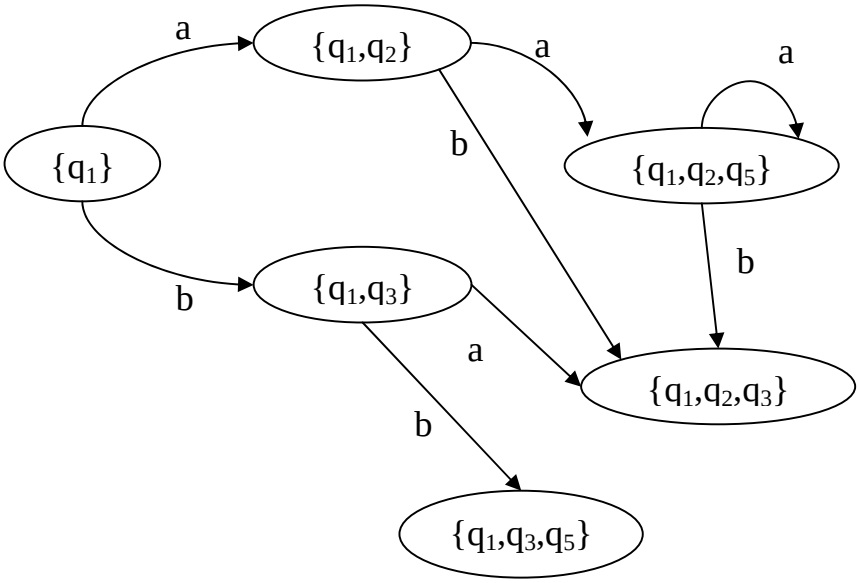
$$\delta^*(q_1, ac) = \bigcup_{q \in \delta(q_1, a)} \delta(q, c) = \delta(q_1, c) \cup \delta(q_2, c) = \{q_1, q_4\} \cup \{q_2\} = \{q_1, q_4, q_2\}$$

Quindi:

$$\delta^*(q_1, aca) = \delta(q_1, a) \cup \delta(q_4, a) \cup \delta(q_2, a) = \{q_1, q_2\} \cup \{q_4\} \cup \{q_2, q_5\} = \{q_1, q_2, q_4, q_5\}$$

Quindi $\delta^*(q_1, aca) \cap F \neq \varnothing \quad \Rightarrow \quad aca \in L(m)$.

**N.B.** : $\forall q \in Q, w \in \Sigma^* \quad \delta^*(q, w)$ e' l'insieme degli stati che e' possibile raggiungere nel diagramma di transizione a partire da *q* seguendo un cammino etichettato *w*.




**Osservazione** : FA $\Rightarrow$ NFA (ovvio)
*m* FA induce $\overline{m}$ NFA con $\bar{\delta}(q, s) = \{\delta(q, s)\}$.

**Def.** :Un Linguaggio accettato da un AFN si dice AFN-regolare.

Per l'osservazione precedente si ha:

Teorema :
Se L e' AF-regolare allora e' AFN-regolare.

Viceversa, si ha:

Teorema :
Se L e' AFN-regolare allora e' AF-regolare.
**Dim**:
Sia L=L(*m*) con *m* AFN
*m*:

$Q = \{q_1,...,q_m\}$

$q_1$

$F \subseteq Q$

$\delta$

Costruiremo $\tilde{m}$ (AF) tale che $L(\tilde{m}) = L(m) = L$.
Idea : gli stati di $\tilde{m}$ rappresentano insiemi di stati di *m*.

$\tilde{Q} = \{Q_1, Q_2, ..., Q_{2^m}\} = pow(Q)$

dove :

$Q_1 = \{q_1\}$ e' stato iniziale

$\tilde{F} = \{Q_i \mid Q_i \cap F \neq \varnothing\}$

$\tilde{\delta}(Q_i, s) \underset{def}{=} \bigcup_{q \in Q_i} \delta(q,s)$.

Lemma 1 :

Sia $R \subseteq \tilde{Q}$. Allora $\tilde{\delta}\left(\bigcup_{Q_i \in R} Q_i, s\right) = \bigcup_{Q_i \in R} \tilde{\delta}(Q_i, s)$. (Senza dim.)

Lemma 2 :
Per ogni $u \in \Sigma^*$ si ha: $\tilde{\delta}^*(Q_i, u) = \bigcup_{q \in Q_i} \delta^*(q,u)$. (Senza dim.)

Lemma 3 :
$L(m) = L(\tilde{m})$.
**Dim.** :
$u \in L(\tilde{m}) \Leftrightarrow \tilde{\delta}^*(Q_1, u) \in F$ ma per il lemma 2 si ha :
$\tilde{\delta}^*(Q_1, u) = \bigcup_{q \in Q_1} \delta^*(q,u) = \delta^*(q_1, u)$ poiche' $Q_1 = \{q_1\}$.
Quindi:

$$u \in L(\tilde{m}) \Leftrightarrow \tilde{\delta}^*(Q_1, u) \in \tilde{F}$$
$$\Leftrightarrow \delta^*(q_1, u) \cap F \neq \varnothing$$
$$\Leftrightarrow u \in L(m)$$

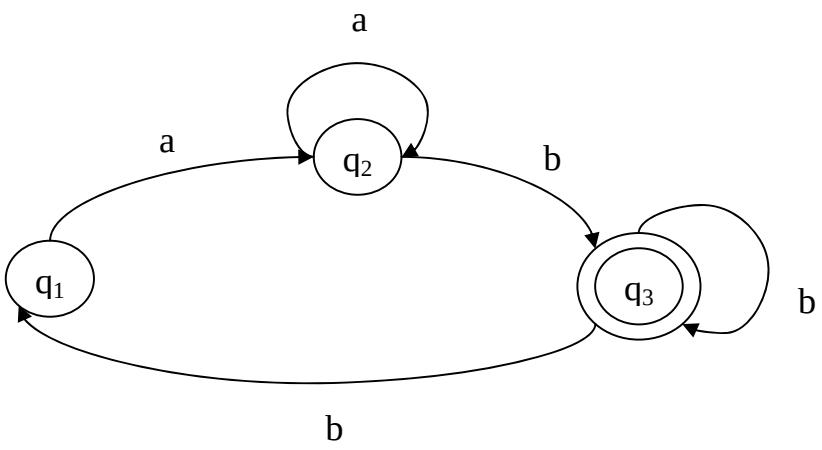
□

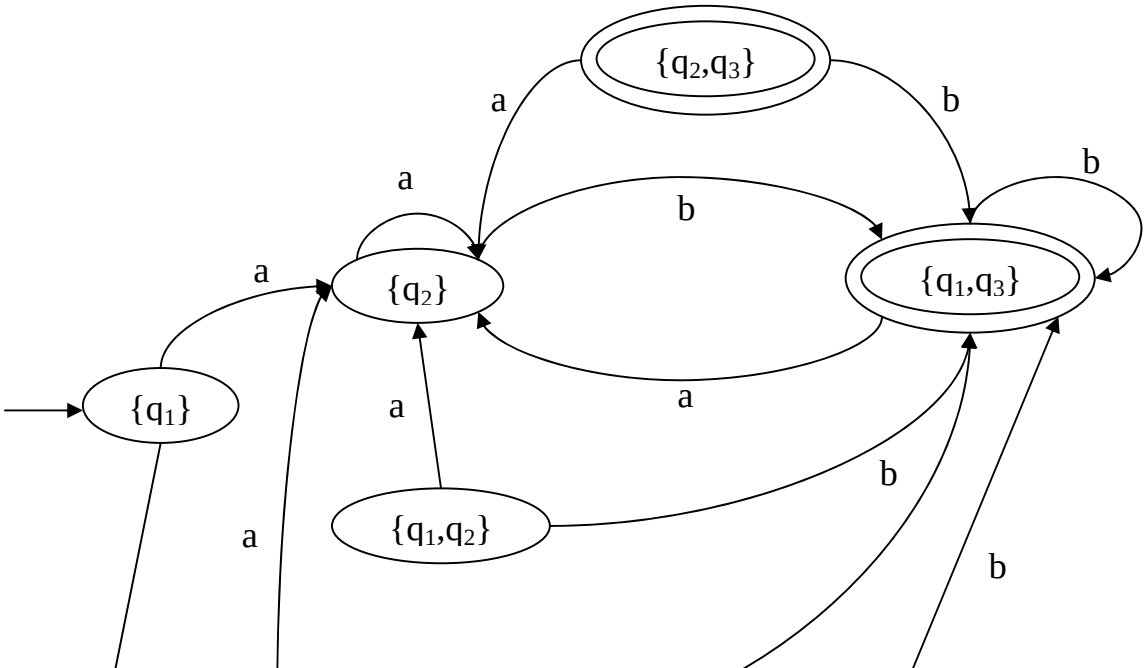
Il teorema segue dal lemma 3.

□

Teorema :
Un linguaggio e' AF-regolare se e solo se e' AFN-regolare.
**Dim.** :
La dimostrazione e' costruttiva:

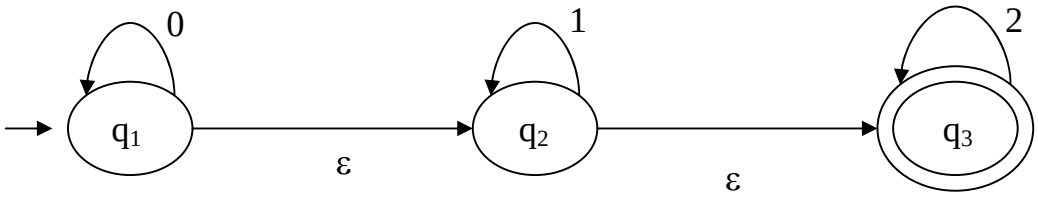
Esempio :
$L = \{a^{n_1} b^{m_1} \ldots a^{n_k} b^{m_k} \mid n_1, m_1, \ldots, n_k, m_k > 0\}$



# Automi Finiti Non Deterministici con ε-Transizioni

Esempio :




$L = \{0^i 1^j 2^l \mid i, j, l \geq 0\}$

**Def.** : Un automa finito non deterministico con ε-transizioni e' una quintupla $m = (Q, \Sigma, \delta, q_1, F)$ tale che:
$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow pow(Q)$.

Esempio :

| $\delta$ | 0 | 1 | 2 | ε |
|---|---|---|---|---|
| $q_1$ | $\{q_1\}$ | $\varnothing$ | $\varnothing$ | $\{q_2\}$ |
| $q_2$ | $\varnothing$ | $\{q_2\}$ | $\varnothing$ | $\{q_3\}$ |
| $q_3$ | $\varnothing$ | $\varnothing$ | $\{q_3\}$ | $\varnothing$ |

L'intenzione e' di far si' che $\delta(q,a)$ consista di tutti gli stati *p* tali che c'e' una transizione etichettata *a* da "*q*" a "*p*", dove *a* e' ε oppure un simbolo in Σ.

Adesso estendiamo la funzione di transizione δ alla funzione $\hat{\delta}$ definita da $Q \times \Sigma^*$ in $2^Q$. L'intenzione e' quella di fare in modo che $\hat{\delta}(q,w)$ contenga tutti gli stati *p* tali che esiste un cammino da "*q*" a "*p*" etichettato *w* tale che include, al limite, archi etichettati ε. Nel costruire $\hat{\delta}$ sara' importante calcolare l'insieme degli stati raggiungibili da un dato stato *q* usando soltanto ε-transizioni.

**Def.** : ε-closure(*q*) = insieme di tutti i nodi "*p*" per cui c'e' un cammino da "*q*" a "*p*" passante per ε archi solamente.

**Def.** : ε-closure(*P*) = $\bigcup_{q \in P} \varepsilon - closure(q)$.

Adesso possiamo definire $\hat{\delta}$:

**Def.** : $\hat{\delta} : Q \times \Sigma^* \to pow(Q)$

1. $\hat{\delta}(q,\varepsilon) = \varepsilon - closure(q)$
2. $\forall w \in \Sigma^*, a \in \Sigma \qquad \hat{\delta}(q,wa) = \varepsilon - closure(P)$

   dove:

   $P = \{p \mid per\ qualche\ r \in \hat{\delta}(q,w), p\ e'\ in\ \delta(r,a)\} =$

   $= \{p \mid p \in \delta(r,a)\ per\ qualche\ r \in \hat{\delta}(q,w)\}$

**Nota** : In questo caso $\hat{\delta}(q,a)$ non e' necessariamente uguale a $\delta(q,a)$ poiche' $\hat{\delta}(q,a)$ include tutti gli stati raggiungibili da *q* con cammini etichettati "*a*" (includendo cammini con archi etichettati "ε"), mentre $\delta(q,a)$ contiene soltanto quelli stati raggiungibili da *q* con archi etichettati "*a*".
In modo simile $\hat{\delta}(q,\varepsilon)$ non e' necessariamente uguale a $\delta(q,\varepsilon)$. E' necessario quindi distinguere fra δ e $\hat{\delta}$ quando si parla di un NFA con ε-transizioni.

**Def.** : $L(m) \underset{def}{=} \{w \varepsilon \Sigma^* \mid \hat{\delta}(q_1, w) \cap F \neq \varnothing\}$.

Esempio :

$\varepsilon - closure(q_1) = \{q_1, q_2, q_3\}$

$\varepsilon - closure(q_2) = \{q_2, q_3\}$

$\varepsilon - closure(q_3) = \{q_3\}$

**Nota** : $q \in \varepsilon - closure(q) \quad \forall q \in Q$.

Esempio :

$$\hat{\delta}(q_1,\varepsilon) = \varepsilon - closure(q_1) = \{q_1,q_2,q_3\}$$

$$\hat{\delta}(q_1,0) = \varepsilon - closure\Big(\delta\big(\delta(q_1,\varepsilon)\big)0\Big) =$$

$$= \varepsilon - closure\Big(\delta\big(\{q_1,q_2,q_3\},0\big)\Big) =$$

$$= \varepsilon - closure\Big(\delta(q_1,0) \cup \delta(q_2,0) \cup \delta(q_3,0)\Big) =$$

$$= \varepsilon - closure\Big(\{q_1\} \cup \varnothing \cup \varnothing\Big) =$$

$$= \varepsilon - closure\Big(\{q_1\}\Big) = \{q_1,q_2,q_3\}$$

$$\hat{\delta}(q_1,01) = \varepsilon - closure\Big(\delta\big(\delta(q_1,0)\big)1\Big) =$$

$$= \varepsilon - closure\Big(\delta\big(\{q_1,q_2,q_3\},1\big)\Big) =$$

$$= \varepsilon - closure\Big(\{q_2\}\Big) = \{q_2,q_3\}$$

**Nota** : $\hat{\delta}(q_1,0) \neq \delta(q_1,0)$.

# Equivalenza tra Automi Finiti Non Deterministici con e senza ε-Transizioni

Teorema :
Se L e' accettato da un NFA con ε-transizioni, allora L e' accettato da un NFA senza ε-transizioni.

Teorema:
Un insieme e' AF-Regolare se e sole se e' accettato da un automa finito non deterministico con o senza ε-transizioni.

□

# Proprieta' di Chiusura degli Insiemi AF-Regolari

Teorema:
Se L ed $\tilde{L}$ sono AF-Regolari, allora $L \cup \tilde{L}$ e' AF-Regolare.
**Dim**:
$L = L(m) \quad \tilde{L} = L(\tilde{m})$

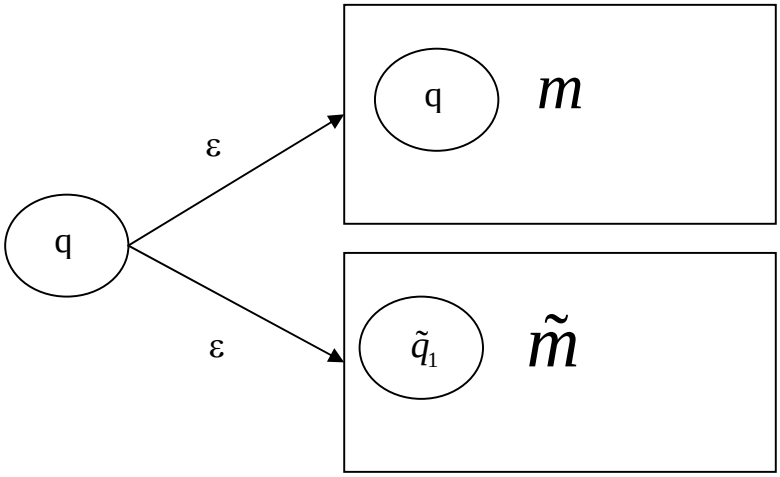
$m = (Q, \Sigma, \delta, q_1, F)$

$\tilde{m} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_1, \tilde{F})$

$con \quad Q \cap \tilde{Q} = \varnothing$

Si consideri : $m_{\cup} = (Q \cup \tilde{Q} \cup \{q_0\}, \Sigma, \delta_{\cup}, q_0, F \cup \tilde{F})$ AFN con ε-transizioni, con :

$\delta_{\cup}\big|_{Q\times\Sigma} = \delta \quad \delta_{\cup}\big|_{\tilde{Q}\times\Sigma} = \tilde{\delta} \quad e \quad \delta_{\cup}(q_0, \varepsilon) = \{q_1, \tilde{q}_1\}$

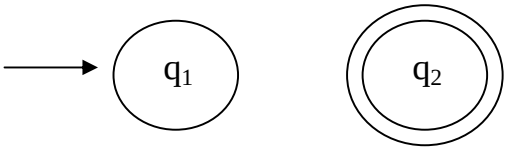



Chiaramente : $L(m_{\cup}) = L(m) \cup L(\tilde{m}) = \tilde{L} \cup L$.

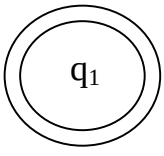
□

Teorema:
$\varnothing, \{\varepsilon\}$ sono AF-Regolari.
**Dim**:

Accetta $\varnothing$




Accetta $\{\varepsilon\}$




□

Teorema:

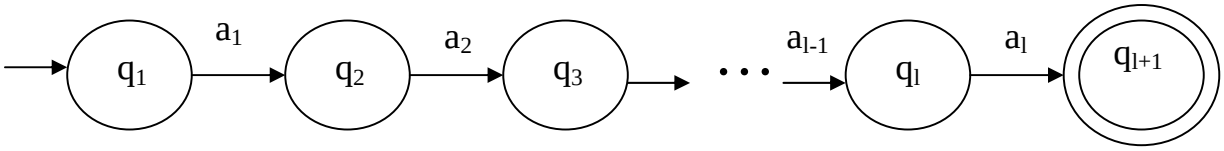
Sia $u \in \Sigma^*$. Allora $\{u\}$ e' AF-Regolare.

**Dim**:

Se $u = \varepsilon$ allora segue dal teorema precedente.

Altrimenti $u = a_1 a_2 ... a_l \quad (l \geq 1), a_i \in \Sigma$.

Sia *m*:




Chiaramente: $L(m) = \{u\}$.

□

Corollario:

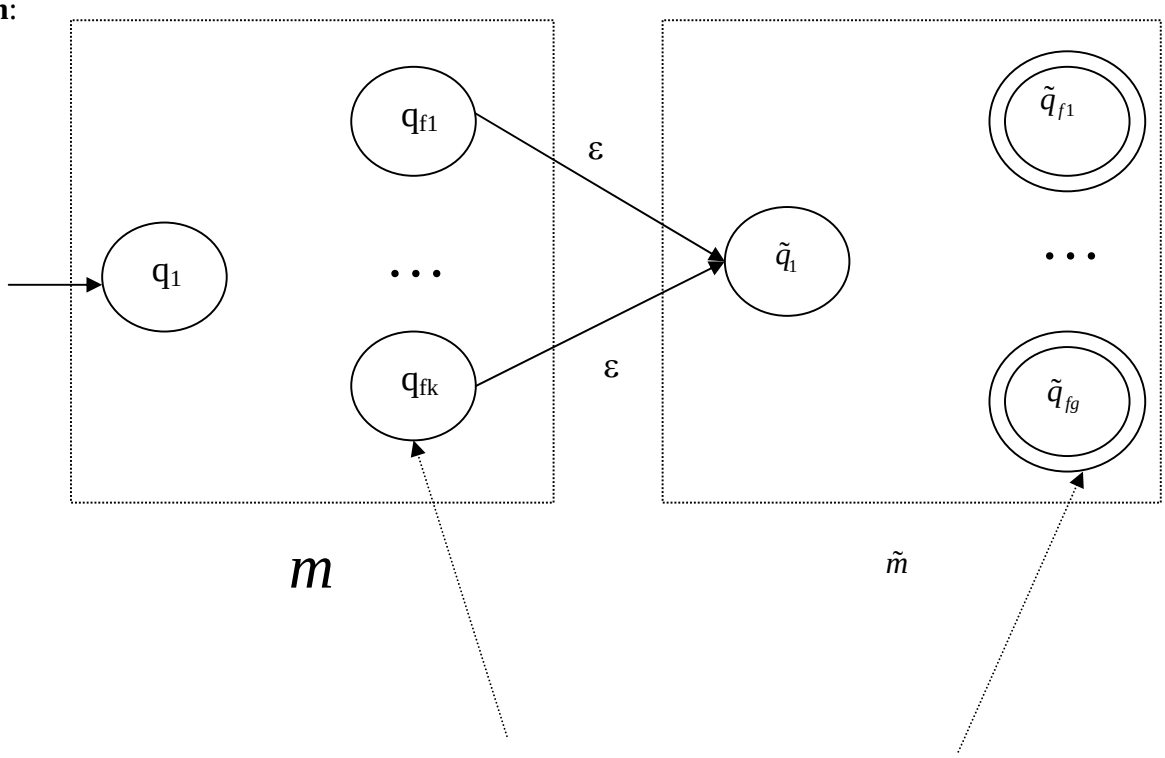
Ogni sottoinsieme finito di $\Sigma^*$ e' AF-Regolare.

**Dim**:

Segue dal teorema precedente e dalla proprieta' di linearita' rispetto a U.

□

Teorema:
Se $L$, $\tilde{L}$ sono AF-Regolari allora $L \cdot \tilde{L}$ e' AF-Regolare.
**Dim**:




$Q \cap \tilde{Q} = \varnothing$

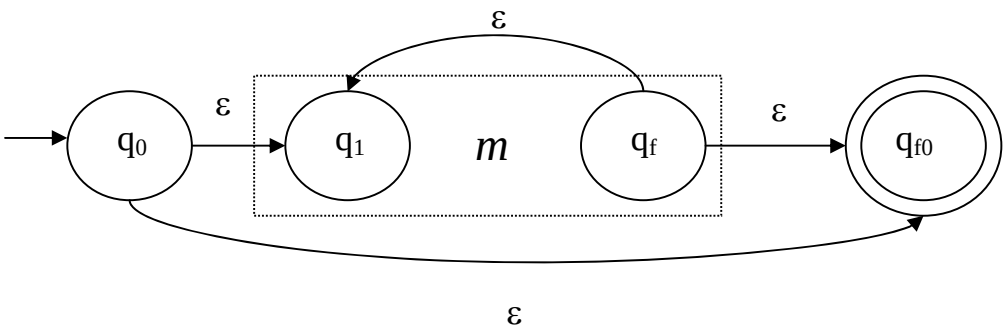
$\dot{F} = \tilde{F}$

$\dot{q}_1 = q_1$

Chiaramente : $L\left(\dot{m}\right) = L(m) \bullet L(\tilde{m}) = L \bullet \tilde{L}$.

□

Teorema:
Se L e' AF-Regolare, tale e' $L^*$.
**Dim**:



Si ha: $L\left(\overset{*}{m}\right)=\left(L(m)\right)^{*}=L^{*}$.

□

In particolare abbiamo visto:

1. $\varnothing,\{\varepsilon\}$ sono AF-Regolari
2. se $L_1,L_2$ sono AF-Regolari allora:
   a. $L_1\cup L_2$ e' AF-Regolare;
   b. $L_1\bullet L_2$ e' AF-Regolare;
   c. $L_1^{*}$ e' AF-Regolare.

Ne segue quasi immediatamente:

Teorema:

Ogni linguaggio regolare e' AF-Regolare.

**Dim**:

Se L e' regolare, allora esiste *r* espressione regolare tale che $L=\langle r\rangle$. Il teorema si dimostra per induzione sul numero di operatori che compaiono bell'espressione regolare *r*:

Se *n*=0 allora $r=\varnothing$ oppure $r=\varepsilon$ oppure $r=\underline{a}$ e quindi $L=\varnothing$ oppure $L=\{\varepsilon\}$ oppure $L=\{a\}$ e abbiamo gia' visto che questi insiemi sono tutti AF-Regolari.

Se *n*>1 allora *r* si puo' essere di tre tipi diversi:

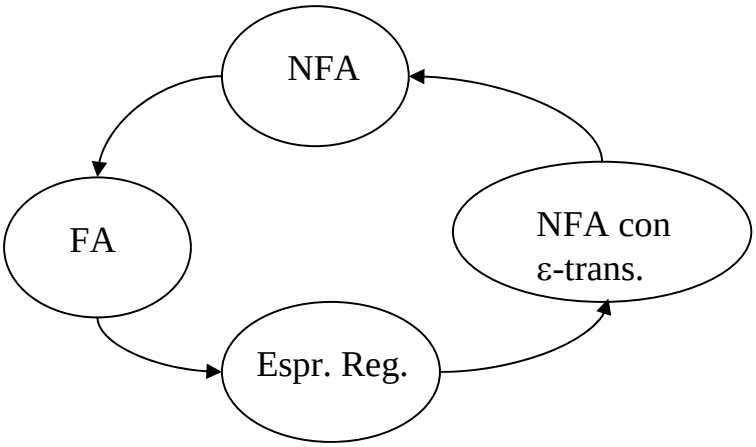
1. $r=\alpha\cup\beta$
2. $r=\alpha\beta$
3. $r=\alpha^{*}$

dove $\alpha,\beta$ sono espressioni regolari con un numero di operatori strettamente minore del numero di operatori di *r*. Per ipotesi induttiva allora $\langle\alpha\rangle,\langle\beta\rangle$ sono insiemi AF-Regolari. Quindi:

1. $L=\langle r\rangle=\langle\alpha\cup\beta\rangle=\langle\alpha\rangle\cup\langle\beta\rangle$.
   Per la proprieta' di chiusura degli insiemi AF-Regolari rispetto all'unione segue il teorema;
2. $L=\langle r\rangle=\langle\alpha\beta\rangle=\langle\alpha\rangle\langle\beta\rangle$.
   Il teorema segue per la proprieta' di chiusura degli insiemi AF-Regolari rispetto alla concatenazione;
3. $L=\langle r\rangle=\langle\alpha^{*}\rangle=\langle\alpha\rangle^{*}$.
   …

□

Si ha anche il viceversa.




Teorema (di Kleene):
Ogni linguaggio AF-Regolare e' regolare. (Senza dim.)

# Grammatiche Regolari

**Def.** : Una grammatica e' detta regolare se tutte le sue produzioni hanno la forma;

$U \to aV$ oppure $U \to a$, con $U,V \in N$ ed $a \in \Sigma$.

Teorema:

Sia $L$ un linguaggio regolare. Allora esiste una grammatica regolare $G$ tale che $L(G) = L \setminus \{\varepsilon\}$.

**Dim**:

Sia $L = L(m)$ *con* $m = (Q, \Sigma, \delta, q, F)$ AF, dove $Q = \{q_1, ..., q_m\}$, $\Sigma = \{s_1, ..., s_n\}$.

Costruiremo $G$ con $N=Q$, $T=\Sigma$, simbolo iniziale $q_1$. Le produzioni sono:

1. $q_i \to s_r q_j$ *ogniqualvolta* $\delta(q_i, s_r) = q_j$
2. $q_i \to s_r$ *ogniqualvolta* $\delta(q_i, s_r) \in F$

Ovviamente $G$ e' regolare.

Dimostriamo che $L(G) = L \setminus \{\varepsilon\}$:

Sia $u \in L \setminus \{\varepsilon\}$, $u = s_{i_1} s_{i2} ... s_{il} s_{i_{l+1}}$

si ha: $\delta(q_1, s_{i_1}) = q_{j_1}, \delta(q_{j_1}, s_{i_2}) = q_{j_2}, ..., \delta(q_{j_l}, s_{i_{l+1}}) = q_{j_{l+1}} \in F$ .

Quindi $G$ contiene le produzioni:

$q_1 \to s_{i_1} q_{j_1}, q_{j_1} \to s_{i_2} q_{j_2}, ..., q_{j_l} \to s_{i_{l+1}}$.

Queste, applicate in sequenza, danno:

$q_i \Rightarrow s_{i_1} q_{j_1} \Rightarrow s_{i_1} s_{i_2} q_{j_2} \Rightarrow ... \Rightarrow s_{i_1} ... s_{i_l} q_{j_l} \Rightarrow s_{i_1} ... s_{i_l} s_{i_{l+1}}$ cioe' $q_1 \overset{*}{\Rightarrow} u \quad u \in L(G)$.

D'altra parte, se $u \in L(G)$, allora I passaggi dimostrativi precedenti possono essere facilmente invertiti e si dimostra che $\delta^*(q_1, u) \in F$. Pertanto $L(G) = L(m) \setminus \{\varepsilon\}$.

□

Teorema:

Sia $G$ una grammatica regolare, allora $L(G)$ e' un linguaggio regolare.

**Dim**:

Siano $V_1, V_2, ..., V_k$ le variabili di $G$ (con $S = V_1$) e siano $s_1, s_2, ..., s_n$ i simboli terminali di $G$.

Si costruisca l'automa finito non deterministico $m$ con stati $V_1, V_2, ..., V_k, W$, dove $V_1$ e' lo stato iniziale e $W$ e' l'unico stato accettante.

Si ponga:

$$\delta_1(V_i, s_r) = \{V_j \mid V_i \to s_r V_j \; e' \; in \; P\}$$

$$\delta_2(V_i, s_r) = \begin{Bmatrix} \{W\} se V_i \to s_r \; e' \; in \; P \\ \varnothing \; altrimenti \end{Bmatrix}$$

Sia $\delta(V_i, s_r) = \delta_1(V_i, s_r) \cup \delta_2(V_i, s_r)$ la funzione di transizione di $m$.

Rimane da dimostrare che $L(m)=L(G)$. Pertanto:

$V_1 \Rightarrow s_{i_1} V_{j_1} \Rightarrow s_{i_1} s_{i_2} V_{j_2} \Rightarrow ... \Rightarrow s_{i_1} ... s_{i_l} V_{j_l} \Rightarrow s_{i_1} ... s_{i_l} s_{i_{l+1}}$

con $G$ contenente le produzioni:

$V_1 \to s_{i_1} V_{j_1}, V_{j_1} \to s_{i_2} V_{j_2}, ..., V_{j_l} \to s_{i_{l+1}}$.

Quindi:

$V_{j_1} \in \delta(V_1, s_{i_1})$

$V_{j_2} \in \delta(V_{j_1}, s_{i_2})$

,...,

$V_{j_l} \in \delta(V_{j_{l-1}}, s_{i_l})$

$w \in \delta(V_{j_l}, s_{i_{l+1}})$

da cui segue $w \in \delta^*(V_1, u)$, cioe' $u \in L(m)$.
Il viceversa si dimostra in maniera del tutto analoga.

□

Teorema:
Sia $G$ una grammatica right-linear. Allora $L(G)$ e' regolare.
**(Senza dim**.)

Quindi ogni grammatica lineare a destra ha una grammatica regolare equivalente (a meno della parola vuota ε). Talvolta infatti le grammatiche regolari sono cosi' definite:
***G* regolare se e' lineare a destra oppure lineare a sinistra**.

## Pumping Lemma (Lemma di Iterazione):

Sia $L=L(m)$, con $m$ AF con $n$ stati. Sia $x \in L$, con $|x| \geq n$. Allora possiamo scrivere:

$x = uvw \quad con\ v \neq \varepsilon \quad e\ uv^{[i]}w \in L, \forall i = 0,1,2,...$

**Dim**:

Siano $x_1, x_2, ..., x_k$ (con $k = |x|$) I prefissi di $x$ di lunghezza 1,2,…,$k$ rispettivamente.

Si consideri l'insieme di stati:

$Q_0 = \{q_i = \delta^*(q_0, x_i) \mid i = 1,2,...,k\}$.

Chiaramente gli stati $q_0, q_1, ..., q_k$ non possono essere tutti distinti ($|Q| = n$).

Ne segue che esistono $0 \leq i_0 < i_1 \leq n$ tali che $q_{i_0} = q_{i_1}$, cioe' $\delta^*(q_0, x_{i_0}) = \delta^*(q_0, x_{i_1})$.

Siano $u,v,w$ tali che

$u = x_{i_0}$

$x_{i_1} = uv$

$x = uvw$

Chiaramente $v \neq \varepsilon$.

Inoltre:

$\delta^*(q_0, uw) = \delta^*(\delta^*(q_0, u), w) = \delta^*(\delta^*(q_0, uv), w) = \delta^*(q_0, uvw) = \delta^*(q_0, x)$

cioe' $uw \in L(m)$.

E si ha anche:

$\delta^*(q_0, uv^{[i]}w) = \delta^*(\delta^*(q_0, uv), v^{[i-1]}w) = \delta^*(\delta^*(q_0, u), v^{[i-1]}w) = \delta^*(q_0, uv^{[i-1]}w) = \delta^*(q_0, uvw)$

(per induzione).

Quindi $uv^{[i]}w \in L(m), \forall i = 0,1,2,...$

□

## Applicazioni del Pumping Lemma

Uso del Pumping Lemma per dimostrare che certi linguaggi **non sono regolari**.

1. Sia $L$ il linguaggio di cui si voglia dimostrare la non regolarita';
2. Si fissi $n$ arbitrariamente grande;
3. Si scelga una stringa $z$ ( in funzione di $n$ ) in $L$;
4. Si consideri un modo qualsiasi di spezzare $z$ in $u,v,w$ (cioe' $z=uvw$) $|uw| < n \quad |v| \geq 1$;
5. Si dimostri che esiste $i$ tale che $uv^i w \notin L$.

Esempio:

$L = \{a^m b^m \mid m > 0\}$

Dato $n>0$, si consideri $z = a^n b^n$.

Sia $z=uvw$, si hanno tre casi:

1. $u \in aa^*$
2. $v \in bb^*$
3. $v \in aa^* bb^*$

In ogni caso e' immediato verificare che $uv^2 w \notin L$. Ne segue che $L$ non e' regolare.

Esempio:

$L = \{0^{i^2} \mid i \text{ e' un intero}, i \geq 1\}$, non e' regolare.

Posto $z = 0^{n^2}$ si divida $z=uvw$ con $1 \leq |v| \leq n$. Allora posto $i=2$ si puo' vedere che $uv^i w \notin L$. Infatti si ha:

$$|uvw| < |uv^2 w| = |uvw| + |v| = n^2 + |v| \leq n^2 + n < (n+1)^2$$

perche' $|v| \geq 1$

da cui $n^2 = |uvw| < |uv^2 w| < (n+1)^2$ cioe' $uv^2 w \notin L$

(non c'e alcun intero che sia un quadrato perfetto compreso fra $n^2$ ed $(n+1)^2$).

# Teorema di Myhill-Nerode e Minimizzazione di Automi Finiti

**Def.**: Sia $L \subseteq \Sigma^*$ . Scriviamo $xR_L y \quad (x,y \in \Sigma^*)$ se
$\forall z \in \Sigma^* \quad xz \in L \Leftrightarrow yz \in L$.

$R_L$ e' una relazione di equivalenza ( e' cioe' riflessiva, simmetrica e transitiva).
$R_L$ (come tutte le relazioni di equivalenza) consente di partire $\Sigma^*$ in insiemi mutuamente disgiunti detti classi di equivalenza (relative ad $R_L$).
Il numero di classi di equivalenza dicesi indice della relazione.

**Def.**: Una relazione di equivalenza $R$ (su $\Sigma^*$) e' detta invariante a destra (rispetto alla concatenazione) se e' tale che $xRy$ implica $xzRyz$ per ogni $z \in \Sigma^*$.

**Def.**: Dato un automa finito $m$ (deterministico), possiamo definire la relazione $R_m$ su $\Sigma^*$: $x R_m y$ se e solo se $\delta^*(q_1, x) = \delta^*(q_1, y)$ (dove $q_1$ e' lo stato iniziale di $m$).

Lemma:
Per ogni linguaggio $L \subseteq \Sigma^*$ e per ogni automa finito $m$ su $\Sigma^*$ le relazioni $R_L$ ed $R_m$ sono invarianti a destra.

Teorema (Myhill-Nerode):
Le seguenti affermazioni sono equivalenti:

1. $L \subseteq \Sigma^*$ e' regolare.
2. L e' l'unione di alcune classi di equivalenza di una relazione di equivalenza invariante a destra di indice finito.
3. La relazione $R_L$ ha indice finito.

# Grammatiche e Linguaggi Context-Free

**Def.**: Una Grammatica $G=(N,\Sigma,P,S)$ e' detta context-free se ogni produzione in $P$ e' del tipo $A\to\alpha$, con $A\in N$ ed $\alpha \in (N\cup\Sigma)^*$.
I linguaggi generati da una grammatica context-free si dicono linguaggi context-free.

Le grammatiche context-free si prestano bene a descrivere costrutti ricorsivi, come le strutture a blocchi dei linguaggi di programmazione.

Esempio:

<statement> → <block>
<block> → begin <list_of_statements> end
<list_of_statements> → <statement>; <list_of_statements> | ε.

Esempio:

E → E + E | E – E
E → E * E | E \ E
E → (E)
E → id

**Def.** : Una derivazione $S\Rightarrow\alpha_1\Rightarrow\alpha_2\Rightarrow\ldots\Rightarrow\alpha_{n-1}\Rightarrow\alpha_n$ si dice leftmost (sinistra) se:
$\forall i=1,\ldots,n-1 \quad si\ ha \quad \alpha_i=uX\beta_i \quad e \quad \alpha_{i+1}=u\gamma\beta_i, \quad con\ u\in\Sigma^*, X\in N, (X\to\gamma)\ in\ P$
Si dice rightmost (destra) se :
$\forall i=1,\ldots,n-1 \quad si\ ha \quad \alpha_i=\beta_iXu \quad e \quad \alpha_{i+1}=\beta_i\gamma u, \quad con\ u\in\Sigma^*, X\in N, (X\to\gamma)\ in\ P$
Nel primo caso si scrive : $\alpha_i \underset{L}{\overset{*}{\Longrightarrow}} \alpha_j \quad (i<j)$.
Nel secondo caso si scrive : $\alpha_i \underset{R}{\overset{*}{\Longrightarrow}} \alpha_j \quad (i<j)$.

**Def.** : Data una grammatica context-free $G=(N,\Sigma,P,S)$, un albero di derivazione $\Gamma$ ( o di parsing) per $G$ e' un albero $\Gamma$ tale che:

1. ogni vertice e' etichettato da un simbolo in $N\cup\Sigma\cup\{\varepsilon\}$;
2. la radice e' etichettata con $S$ ;
3. ogni vertice interno e' etichettato con $A$ e se $\nu_1,\ldots,\nu_k$ sono I discendenti diretti di $\nu$, in ordine da sinistra a destra, con etichette $X_1,\ldots,X_k$ rispettivamente ($X_i\in N\cup\Sigma\cup\{\varepsilon\}$) allora $A\to X_1\ldots X_k$ deve essere una produzione in $P$;
4. se il vertice $\nu$ ha etichetta $\varepsilon$, allora $\nu$ e' una foglia, ed e' l'unico discendente del proprio genitore.

La parola che si ottiene leggendo le etichette delle foglie di un albero di derivazione $\Gamma$ (da sinistra a destra) si dice prodotto di $\Gamma$.

Teorema:

Sia $G=(N,\Sigma,P,S)$ una grammatica context-free. Allora $S \overset{*}{\Longrightarrow} \alpha$ se e solo se c'e' un albero di derivazione per $G$ con prodotto $\alpha$.In particolare c'e' una corrispondenza biunivoca tra gli alberi di derivazione e le derivazioni sinistre (derivazioni destre).

**Dim**:

(per induzione).

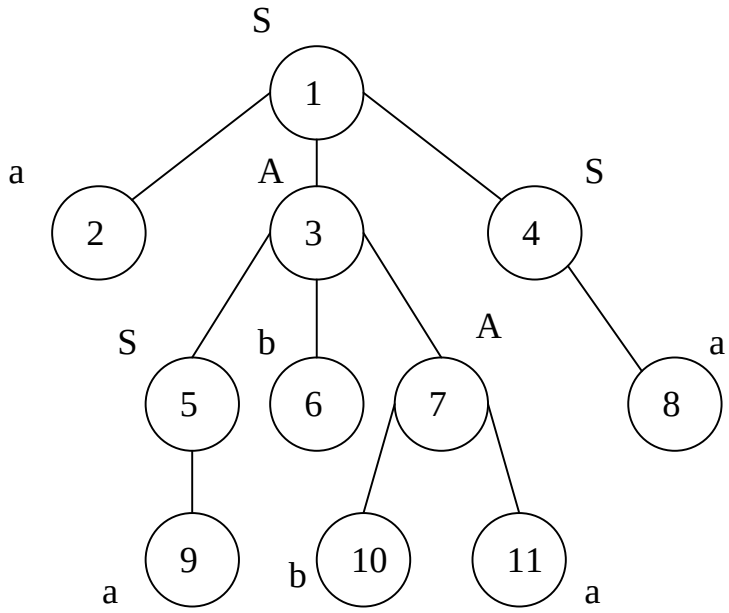
□

Esempio:

$G = (\{S,A\},\{a,b\},P,S)$

S→aAS | a

A→SbA | SS | ba




La derivazione leftmost corrispondente a quest'albero e':

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$

La derivazione rightmost corrispondente e':

$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$

ε-Produzioni

**Def.**: Una ε-produzione e' una produzione del tipo A$\rightarrow\varepsilon$.

Se $A \overset{*}{\Longrightarrow} \varepsilon$, A si dice annullabile.

**Def.**: Una grammatica context-free G=(N,Σ,P,S) e' detta in forma normale di Chomsky se ciascuna delle sue produzioni ha una delle due forme:

A→BC oppure A→a

con A,B,C∈N e a∈Σ.

Teorema:

C'e' un algoritmo che trasforma una data grammatica context-free priva di ε-produzioni in una grammatica context-free equivalente in forma normale di Chomsky.

**Dim**:

Utilizzando il precedente teorema, possiamo supporre che la data grammatica G=(N,Σ,P,S) non contenga produzioni unitarie. Per ogni simbolo terminale a∈Σ introduciamo una nuova variabile $X_a$. Quindi sostituiamo ogni produzione A→α per cui α∉Σ (α non e' un singolo terminale) con la produzione $A \to \alpha'$ in cui $\alpha'$ e' ottenuto sostituendo in α ogni terminale $a$ con la corrispondente variabile $X_a$. Inoltre aggiungiamo tutte le produzioni $X_a \to a$.

Chiaramente la grammatica ottenuta e' equivalente a quella data.

Sia $\overline{G} = (\overline{N}, \Sigma, \overline{P}, S)$ e facciamo vedere che $L(G) = L(\overline{G})$. Banalmente $L(G) \subseteq L(\overline{G})$. Per dimostrare l'inclusione inversa facciamo vedere che se $A \underset{\overline{G}}{\Rightarrow} w$ con $A \in N, w \in \Sigma^*$, allora $A \overset{*}{\underset{G}{\Rightarrow}} w$. La dimostrazione per induzione nel numero di passi della derivazione in $\overline{G}$. Il risultato e' banale per derivazioni di un solo passo. Supponiamolo vero per derivazioni fino a $k$ passi e sia $A \overset{*}{\underset{\overline{G}}{\Rightarrow}} w$ una derivazione di ($k$+1) passi. Il primo passo deve essere della forma $A \to B_1 B_2 ... B_m \quad con\ m > 2$. Noi allora possiamo scrivere $w = w_1 w_2 ... w_m$ dove $B_i \overset{*}{\underset{\overline{G}}{\Rightarrow}} w_i \quad 1 \le i \le m$. Se $B_i$ e' del tipo $X_{a_i}$ per qualche terminale $a_i$, allora $w_i$ deve essere $a_i$. Allora, per la costruzione di $\overline{P}$, c'e' una produzione $A \to C_1 C_2 ... C_m$ in P dove $C_i = B_i$ se $B_i$ e' una variabile di N e $C_i = a_i$ se $B_i \in \overline{V} - V$ (cioe' se $B_i$ e' una nuova variabile introdotta). Per quei $B_i$ in V, sappiamo che ;a derivazione $B_i \overset{*}{\underset{\overline{G}}{\Rightarrow}} w_i$ prende meno di $k$ passi e quindi per ipotesi induttiva $B_i \overset{*}{\underset{G}{\Rightarrow}} w_i$ (cioe' $C_i \overset{*}{\underset{G}{\Rightarrow}} w_i$) e quindi banalmente $A \overset{*}{\underset{G}{\Rightarrow}} w$.

Inoltre tutte le sue produzioni hanno una delle due forme:

$$X \to X_1 X_2 ... X_k \quad , k \ge 2, X_1 X_2 ... X_k \in N \cup \{X_a : a \in \Sigma\}$$
$$X \to a \quad , a \in \Sigma, X \in N \cup \{X_a : a \in \Sigma\}$$

Per ottenere una grammatica in forma normale di Chomsky e' sufficiente eliminare le produzioni del tipo $X \to X_1 X_2 ... X_k \quad , k > 2$, introducendo nuove variabili $Z_1, Z_2, ..., Z_{k-2}$ ed aggiungendo le produzioni:

$X \to X_1 Z_1$

$Z_1 \to X_2 Z_2$

...

$Z_{k-3} \to X_{k-2} Z_{k-2}$

$Z_{k-2} \to X_{k-1} X_k$

Si ottiene cosi' una grammatica context-free $G'$ in forma normale di Chomsky e tale che $L(G') = L(G)$ (quest'ultima affermazione si dimostra analogamente a quanto fatto sopra).

□

Esempio:

S→aXbY
X→aX | a
Y→bY | b
Z→a
(eliminazione di X→Z con aggiunta di X→a)

Passo 2) Introduzione variabili $X_a$ e $X_b$ con relativa sostituzione

S→ $X_a$ X $X_b$ Y
X→ $X_a$ X | a
Y→ $X_b$ Y | b
$X_a$ →a
$X_b$ →b
Z→a

Passo 3) Eliminazione produzioni $A \to \alpha \quad con \ |\alpha| > 2$

$G'$ :

S→ $X_a Z_1$
$Z_1$→X $Z_2$
$Z_2 \to X_b$ Y
X→ $X_a$ X | a
Y→ $X_b$ Y | b
$X_a$ →a
$X_b$ →b
Z→a

$G'$ e' in forma normale di Chomsky.

Grammatiche Ambigue ed Inerentemente Ambigue

**Def.**: Una grammatica G e' detta ambigua quando esiste $w \in L(G)$ per cui vi sono due distinte derivazioni sinistre (o alberi di derivazione).

Esempio:

$E \rightarrow E+E \mid E-E \mid a$

Allora : $E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E-E \Rightarrow a+a-E \Rightarrow a+a-a$

$E \Rightarrow E-E \Rightarrow E+E-E \Rightarrow a+E-E \Rightarrow a+a-E \Rightarrow a+a-a$

Esempio:

$S \rightarrow A;\ S \rightarrow B;\ A \rightarrow a;\ B \rightarrow a.$

Con accorgimenti ad hoc spesso e' possibile eliminare l'ambiguita'.

**Def.**: Un CFL per cui ogni CFG e' ambigua e'detto un context-free language inerentemente ambiguo.

Ma vi sono pure grammatiche inerentemente ambigue che non e' possibile disambiguare.

Esempio:

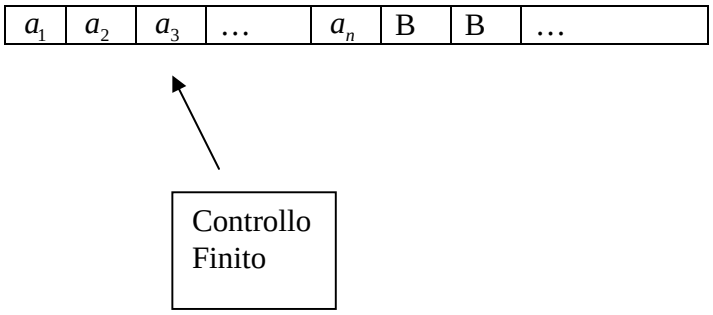
Sia $L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$.

Si dimostra che ogni grammatica G generante L e' ambigua.

Questo esempio, a parte il fatto che 'e di per se' interessante, viene usato per dimostrare che e' indecidibile il problema dell' "inerente ambiguita' " per linguaggi context-free.

# Macchine di Turing

Il modello standard di macchina di Turing era un controllo finito, un nastro di input, diviso in celle, e una testina che prende in considerazione una cella del nastro alla volta. Il nastro ha una prima cella piu' a sinistra ma e' infinito a destra. Ogni cella del nastro puo' contenere esattamente un simbolo scelto fra un numero finito di simboli di nastro. Inizialmente le $n$ celle piu' a sinistra, per qualche $n \geq 0$ (finito), contengono l'input che e' una stringa di simboli scelti nell'insieme dei simboli di input che e' un sottoinsieme dei simboli di nastro. Le rimanenti celle del nastro, in numero infinito, contengono un simbolo di nastro speciale, il blank, che non e' un simbolo di input.




In un movimento la TM, in funzione del simbolo letto dalla testina e dallo stato del controllo finito,

1) cambia stato;
2) stampa un simbolo sulla cella di nastro letta, sostituendo il simbolo che era scritto li', e
3) muove la sua testina di una cella a destra o a sinistra.

Formalmente:

**Def.**: Una Turing Machine e' denotata da $m = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, dove

- Q e' un insieme finito di stati
- $\Gamma$ e' l'insieme dei simboli di nastro (finito)
- $B \in \Gamma$ e' il blank
- $\Sigma \subseteq \Gamma$ ($B \notin \Sigma$) e' l'insieme dei simboli di input
- $\delta$ e' la funzione di transizione da $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (eventualmente non definita per qualche argomento)
- $q_0 \in Q$ e' lo stato iniziale
- $F \subseteq Q$ e' l'insieme degli stati finali.

**Def.**: Denotiamo una descrizione istantanea (ID) della TM $m$ con $\alpha_1 q \alpha_2$.
Qui $q$, lo stato attuale di $m$, e' un elemento di Q; $\alpha_1\alpha_2$ e' la stringa in $\Gamma^*$ che rappresenta il contenuto del nastro fino al simbolo diverso da blank piu' a destra. (Osserva che il simbolo blank puo' occorrere in $\alpha_1\alpha_2$). Per evitare confusione

assumiamo che Q∩Γ∅. Infine assumiamo che la testina stia leggendo il simbolo piu' a sinistra di $\alpha_2$ o, se $\alpha_2$=ε, sta leggendo un blank.

**Def.**: Definiamo un movimento di *m* come segue:
Sia $X_1X_2...X_{i-1}qX_i...X_n$ una ID. Supponiamo che $\delta(q,X_i)=(p,Y,L)$, dove se *i*-1=*n*, allora $X_i$ e' considerato B. Se *i*=1, allora non esiste una ID successiva perche' alla testina di lettura non e' permesso di uscire fuori dal margine sinistro del foglio. Se *i*>1 allora scriviamo
$X_1X_2...X_{i-1}qX_i...X_n \vdash_m X_1X_2...X_{i-2}pX_{i-1}YX_{i+1}...X_n$.
Tuttavia, se qualche suffisso di $X_{i-1}YX_{i+1}...X_n$ e' completamente bianco, quel suffisso viene cancellato.
Se invece supponiamo che $\delta(q,X_i)=(p,Y,R)$ allora scriviamo:
$X_1X_2...X_{i-1}qX_i...X_n \vdash_m X_1X_2...X_{i-2}X_{i-1}YpX_{i+1}...X_n$. (*)
Nota che nel caso in cui *i*-1=*n* , la stringa $X_1...X_n$ e' vuota e il lato destro della (*) e' piu' lungo di quello sinistro.

**Def.**: Se due ID sono in relazione $\vdash_m$ , noi diciamo che la seconda risulta dalla prima mediante un movimento. Se una ID risulta da un'altra per mezzo di un numero finito di movimenti, eventualmente anche 0, allora le due ID sono in relazione $\vdash_m^*$.

**Def.**: Il linguaggio accettato dalla TM, $m=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ e':
$$L(m)=\left\{ w \mid w \in \Sigma^* \text{ e } q_0w \vdash_m^* \alpha_1p\alpha_2 \quad \text{per qualche } p \in F, \alpha_1,\alpha_2 \in \Gamma^* \right\}$$

Data una TM che riconosce un linguaggio L, assumiamo senza perdita di generalita' che la TM si fermi, cioe' non abbia nessun movimento successivo possibile, appena l'input e' accettato.

Esempio:
Consideriamo adesso una TM che accetta il linguaggio $L=\{0^n1^n \mid n \geq 1\}$.
*m* lavora in questo modo:
Inizialmente in nastro contiene $0^n1^n$ seguiti da infiniti blank. Ripetutamente, *m* rimpiazza il simbolo "0" piu' a sinistra con X, si muove a destra fino al simbolo "1" piu' a sinistra, lo rimpiazza con una Y, si muove a sinistra per trovare la X piu' a destra, poi si muove di una cella a destra fino al simbolo "0" piu' a sinistra e ripete il ciclo.
Se tuttavia, mentre cerca un "1" *m* trova un blank al suo posto, allora *m* si ferma senza accettare. Se, dopo aver cambiato un "1" in Y, *m* non trova piu' "0" allora *m* testa se rimangono ancora "1", accettando se non ce ne sono piu'.

Sia $Q=\{q_0,q_1,q_2,q_3,q_4\},\Sigma=\{0,1\},\Gamma=\{0,1,X,Y,B\},F=\{q_4\}$ ,
Informalmente, ogni stato rappresenta una istruzione o un gruppo di istruzioni in un programma: nello stato $q_0$ si entra inizialmente e anche immediatamente prima di ogni

rimpiazzamento di uno “0” piu’ a sinistra, con una X. Lo stato $q_1$ e’ usato per cercare a destra, soltanto “0” e “Y”, fino a quando si trova l’ “1” piu’ a sinistra.
Se si trova un “1” lo cambia in Y ed entra nello stato $q_2$ . Lo stato $q_2$ cerca a sinistra una X, e appena la trova entra nello stato $q_0$ e il ciclo ricomincia.
Quando *m* cerca a destra nello stato $q_1$, se sono incontrati un “B” oppure “X” prima di un “1”, l’input e’ respinto: o ci sono troppi “0” oppure l’input non e’ O*1*.
Lo stato $q_0$ ha un altro ruolo: se lo stato $q_2$ ha trovato la X piu’ a destra e questa X e’ seguita immediatamente da una Y, allora gli “0” sono esauriti. Allora quando $q_0$ legge una Y, entra nello stato $q_3$ per vedere se rimangono ancora “1”; se gli Y sono seguiti da un B allora entra in $q_4$ e accetta, altrimenti la stringa e’ rifiutata.

Funzione di transizione di *m*:

| stato | 0 | 1 | X | Y | B |
|---|---|---|---|---|---|
| $q_0$ | ($q_1$,X,R) | - | - | ($q_3$,Y,R) | - |
| $q_1$ | ($q_1$,0,R) | ($q_2$,Y,L) | - | ($q_1$,Y,R) | - |
| $q_2$ | ($q_2$,0,L) | - | ($q_0$,X,R) | ($q_2$,Y,L) | - |
| $q_3$ | - | - | - | ($q_3$,Y,R) | ($q_4$,B,R) |
| $q_4$ | - | - | - | - | - |

Vediamo adesso una computazione di *m*. Se per esempio consideriamo l’input *x*=0011 abbiamo questa computazione

$$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash$$

$$X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash$$

$$X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 \vdash X X Y Y B q_4$$

Linguaggi Ricorsivamente Enumerabili

**Def.**: Un linguaggio che e' accettato da una TM e' detto ricorsivamente enumerabile.

Il termine enumerabile deriva dal fatto che i linguaggi r.e. sono precisamente quei linguaggi le cui stringhe possono essere numerate (date una dietro l'altra come in una lista) da una TM.

La classe dei linguaggi r.e. e' molto grande ed include propriamente i context-free language.

La classe dei linguaggi r.e. include alcuni linguaggi per I quali noi non possiamo determinare meccanicamente l'appartenenza o meno di una stringa al linguaggio: se L(*m*) e' un tale linguaggio, allora ogni TM che riconosce L(*m*) potrebbe non fermarsi su qualche input non in L(*m*). Fino a quando *m* lavora su un certo input *w* non potremo mai dire se *m* accettera' *w* se noi la lasciamo lavorare abbastanza o se invece *m* lavorera' all'infinito.

E' conveniente distinguere una sottoclasse dei linguaggi r.e., chiamata la classe dei linguaggi ricorsivi: sono quei linguaggi accettati da almeno una TM che si ferma su tutti gli input (nota che la fermata puo' essere preceduta o meno dall'accettazione).

Si puo' dimostrare che la classe dei linguaggi ricorsivi e' un sottoinsieme proprio della classe dei linguaggi r.e.