Dispense del corso di Linguaggi Formali e Automi

Marco Alfieri, Matteo Bianchi, Grazia D'Aversa, Andrea Fumagalli,
Emanuele Mornini, Dario Villa, Massimo Vitali

8 settembre 2002

Appunti tratti dalle lezioni
tenute dal prof. M. Goldwurm
nell'anno accademico 2001/2002

Indice

1 Nozioni di base

1.1 Introduzione e prime definizioni

Fra le strutture matematiche utilizzate nell'ambito dell'informatica, i linguaggi formali svolgono un ruolo particolarmente importante. In questa sezione introduciamo alcuni concetti relativi ai linguaggi ed alle principali operazioni che nel seguito avremo occasione di utilizzare.

Definizione 1.1 *Un* **alfabeto** *è un insieme finito di simboli (es. lettere) che indicheremo con* Σ.

Esempio. $\beta = \{0,1\}$ $A = \{a,b,c\}$ $\Omega = \{a,b,\ldots,z\}$ $P = \{(,),[,]\}$

Definizione 1.2 *Una* **parola** *è una concatenazione di simboli appartenenti ad un alfabeto* Σ*, oppure la parola priva di simboli che denotiamo con* ε *e che chiameremo* **parola vuota**.
Rappresentiamo con $\Sigma^\star$ *l'insieme di tutte le parole su* Σ*, mentre* Σ^+ *denota l'insieme* $\Sigma^\star - \{\varepsilon\}$.

Esempio. Dato l'alfabeto $\Sigma = \{a,b,c,d\}$ $x = aababc$ e $y = dacadc$ sono due parole costruite con simboli di Σ.

Esempio. $\{a,b,c\}^\star = \{\varepsilon, a, b, c, aa, ab, ac, bb, \ldots\}$ $\{a,b,c\}^+ = \{a, b, c, aa, ab, ac, bb, bc, \ldots\}$

Definizione 1.3 *La* **lunghezza di una parola x** *è il numero di occorrenze di simboli che compongono* x*. Se* x *è una parola su* Σ, $|x|$ *denota la sua lunghezza. Per definizione* $|\varepsilon| = 0$*. Quindi* $|\cdot|$ *denota una funzione che chiameremo funzione lunghezza di una parola e che è definita nel seguente modo:*

$$|\cdot| : \Sigma^\star \to N. \tag{1}$$

Definizione 1.4 *Un* **prefisso di x** *è una parola* $y \in \Sigma^\star$ *tale che* $x = yz$ *per qualche* $z \in \Sigma^\star$.

Esempio. $x = aabaca$ allora $aab, aa, \varepsilon, aabaca$ sono prefissi di x.

Definizione 1.5 *Un* **suffisso di x** *è una parola* $y \in \Sigma^\star$ *tale che* $x = zy$ *per qualche* $z \in \Sigma^\star$.

Esempio. $x = aabaca$ allora $aca, ca, \varepsilon, aabaca$ sono suffissi di x.

Definizione 1.6 *Il* **fattore di x** *è una parola* $w \in \Sigma^\star$ *tale che* $x = ywz$ *per qualche* $w, z \in \Sigma^\star$.

Esempio. $x = aabaca$ allora ab, bac, ac, ε sono fattori di x.

Definizione 1.7 *Una* **sottoparola di x** *è una parola* $y \in \Sigma^\star$ *tale che*

1. $x = a_1 a_2 \cdots a_n$ $\quad (a_i \in \Sigma, \forall i)$
2. $y = a_{j1} a_{j2} \cdots a_{jm}$ $\quad 1 \le j1 \le j2 \le \ldots \le n$

In altri termini, la parola y *si deriva da* x *eliminando un numero di simboli* h $(0 \le h \le n)$ *e mantenendo l'ordine dei simboli della parola di partenza.*

Esempio. $\Sigma = \{a,b,c,d,e\}$ $x = eabbacd$ $y = eabcd$

Definizione 1.8 Ordine lessicografico *Ricordiamo innanzitutto che una relazione di ordine totale $\leq$ su Σ è una relazione binaria che gode delle seguenti proprietà:*

1. $\forall a \in \Sigma \quad a \leq a$,
2. $a, b \in \Sigma$ *se* $a \leq b$ *e* $b \leq a \Rightarrow a = b$,
3. $a, b, c \in \Sigma$ *se* $a \leq b$ *e* $b \leq c \Rightarrow a \leq c$,
4. $\forall a, b \in \Sigma \quad a \leq b$ *oppure* $b \leq a$.

Data la relazione di ordine totale $\leq$ su Σ, chiamiamo **ordine lessicografico** *su Σ^+ la relazione d'ordine totale $\leq_L$ su Σ^+ tale che:*

$$\forall x, y \in \Sigma^+ \quad x \leq_L y \quad se$$

$$x = uav \text{ e } y = ubw \text{ dove } \begin{cases} u, v, w \in \Sigma^\star \\ a, b \in \Sigma \\ a \leq b \text{ e } a \neq b \end{cases}$$

oppure

$$\exists z \in \Sigma^\star \text{ tale che } xz = y$$

(Quest'ultima, vale nel caso in cui x è prefisso di y)

Esempio.

$x = bacb$	$y = bacc$	$x \leq_L y$
$x = bac$	$y = bacd$	$x \leq_L y$

Definizione 1.9 Ordine militare(ranking): *E' definito su $\Sigma^\star$ e lo denotiamo con $\leq_R$.*

$$\forall x, y \in \Sigma^\star \quad x \leq_R y \quad se:$$
$$|x| < |y|$$
$$oppure$$
$$|x| = |y| \quad e \quad x \leq_L y$$

E' facile provare che per ogni alfabeto Σ, per ogni ordine totale $\leq$ su Σ e per ogni parola $x \in \Sigma^+$, l'insieme $\{y \in \Sigma^+ : y \leq_R x\}$ è finito. Quindi l'insieme delle parole che precede x è finito.

Definizione 1.10 Rank: *E' la funzione rank: $\Sigma^\star \to N^+$, dove $N^+ = N - \{0\}$, tale che per ogni $x \in \Sigma^\star$, rank(x) rappresenta il numero di $y \in \Sigma^\star$ tali che $y \leq_R x$.*

Definizione 1.11 Unrank: *E' la funzione unrank: $N^+ \to \Sigma^\star$ che è definita in modo tale che rank (unrank(n)) = n per ogni $n \in N$.*

1.2 Linguaggi e relative operazioni

Linguaggio: Un linguaggio L definito su Σ è un sottoinsieme di $\Sigma^\star$ ovvero $L \subseteq \Sigma^\star$. Ad esempio $\emptyset, \{\varepsilon\}, \Sigma, \Sigma^\star$ sono tutti linguaggi.
Per ogni $A, B \subseteq \Sigma^\star$ definiamo le seguenti operazioni:

1. **Unione:** l'insieme $A \cup B = \{x \in \Sigma^\star : x \in A \text{ oppure } x \in B\}$;
2. **Intersezione:** l'insieme $A \cap B = \{x \in \Sigma^\star : x \in A \text{ e } x \in B\}$;
3. **Complementare:** l'insieme $A^C = \{x \in \Sigma^\star : x \notin A\}$;
4. **Differenza:** l'insieme $A \backslash B$, denotato anche con $A - B = \{x \in \Sigma^\star : x \in A \text{ e } x \notin B\}$;
5. **Prodotto:** l'insieme $A \cdot B = \{x \in \Sigma^\star : \exists y \in A, \exists z \in B : x = yz\}$

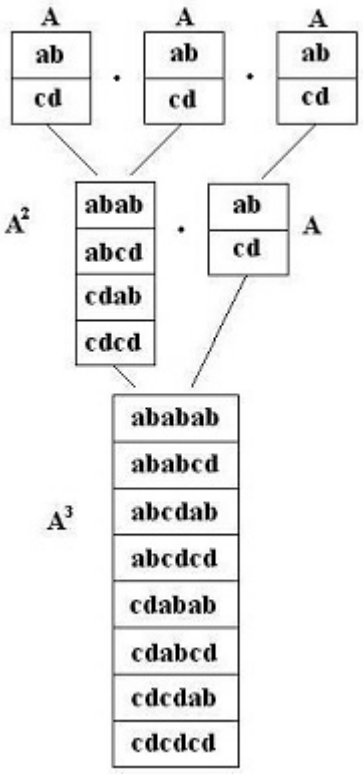
   Esempio.
   $\{a\} \cdot \{a, b, c\}^\star = \{a, aa, ab, ac, aaa, aab, \dots\}$
   $\{a, b, c\}^\star \cdot \{b\} = \{b, ab, bb, cb, aab, aac, \dots\}$

6. **Potenza:** per ogni linguaggio $A \subseteq \Sigma^\star$ definiamo $A^0 = \{\varepsilon\}, A^1 = A, A^2 = A \cdot A, \dots$, ovvero:

$$\forall k \in N, A^k = \begin{cases} \{\varepsilon\} & k = 0 \\ A & k = 1 \\ A \cdot A^{k-1} & k > 1 \end{cases} \tag{2}$$

   Questo significa che $A^k = \{\omega \in \Sigma^\star : \omega = \alpha_1 \alpha_2 \dots \alpha_k \text{ con } \alpha_i \in A, \forall i = 1, \dots, k\}$ per ogni $k \geq 1$.

   Esempio.
   Σ^2 è l'insieme delle parole su Σ di lunghezza 2.
   Σ^3 è l'insieme delle parole su Σ di lunghezza 3.
   $\vdots$
   Σ^k è l'insieme delle parole su Σ di lunghezza k.



rappresentazione dell'operazione potenza dell'insieme $A = \{ab, cd\}$.

7. **Operazione * :** Unione di tutte le potenze di un linguaggio. Per ogni linguaggio $A \subseteq \Sigma^\star$ denotiamo con $A^\star$ l'insieme delle sue potenze, ovvero

$$A^\star = \bigcup_{k=0}^{\infty} A^k. \tag{3}$$

Osserviamo che l'insieme delle parole in Σ coincide proprio con:

$$\Sigma^\star = \bigcup_{k=0}^{\infty} \Sigma^k. \tag{4}$$

8. **Operazione + :** Unione di tutte le potenze positive di un linguaggio. Per ogni $A \subseteq \Sigma^\star$ definiamo

$$A^+ = \bigcup_{k=1}^{\infty} A^k \quad \text{e quindi} \quad A^+ = A^\star - \{\varepsilon\} \tag{5}$$

1.3 Semigruppo, Monoide e Semianello

Definizione 1.12 Semigruppo: *E' un insieme S dotato di una legge di composizione associativa* $\odot$ *che rappresentiamo con la coppia* $< S, \odot >$.

Ricordiamo che una legge di composizione $\odot$ su S è una funzione che associa ad ogni coppia $x, y \in S$, un elemento $x \odot y \in S$. Per esempio $< \Sigma^+, \cdot >$ è un semigruppo dove $\cdot$ è l'operazione di concatenazione tra parole.
Osserviamo che l'operazione $\cdot$ è associativa, ma non commutativa; infatti per ogni $x, y, z \in \Sigma^\star$ vale $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ ma in generale $x \cdot y$ è diverso da $y \cdot x$.

Definizione 1.13 Monoide: *E' un insieme M dotato di un'operazione associativa* $\odot$ *e di un elemento unitario 1 rispetto a* $\odot$ *che denotiamo con la tripla* $< M, \odot, 1 >$.

Ricordiamo che per ogni $x \in M, \quad x \odot 1 = 1 \odot x = x$.

Esempio. L'insieme $\Sigma^\star$ di tutte le parole forma il monoide $<\Sigma^\star, \cdot, \varepsilon>$. Questo è chiamato **monoide libero** su Σ. Questo monoide è detto libero perchè ogni parola si ottiene in un solo modo come concatenazione di simboli dell'alfabeto; ovvero, data una parola formata da simboli, non è possibile riottenerla con simboli diversi o concatenando gli stessi in modo differente.

Definizione 1.14 Semianello: *E' un insieme S dotato di due operazioni* $+, \cdot$ *e dei due elementi neutri rispettivi 0, 1 tale che l'operazione + (somma) è associativa e commutativa, l'operazione* $\cdot$ *(prodotto) è associativa e valgono le leggi distributive della somma rispetto al prodotto. Tale struttura si rappresenta con la notazione* $< S, +, \cdot, 0, 1 >$.

Esempio.

$$< P(\Sigma^\star), \cup, \cdot, \emptyset, \{\varepsilon\} >$$

Dove denotiamo con $P(\Sigma^\star)$ la famiglia di tutti i sottoinsiemi di $\Sigma^\star$. Osserviamo che:

1. $(A \cup B) \cdot C = (A \cdot C) \cup (B \cdot C)$,
2. $A \cdot (B \cup C) = (A \cdot B) \cup (B \cdot C)$.

e che il prodotto di linguaggi non è commutativo.

1.4 Funzione coppia

La **funzione coppia** stabilisce una corrispondenza biunivoca tra coppie di elementi e l'insieme dei numeri naturali.
Per semplicità consideriamo il caso di coppie di numeri appartenenti ad N^+ anche se in realtà questa funzione può essere estesa ad una qualsiasi coppia di elementi. Rappresentiamo sui due lati di una matrice infinita rispettivamente i due insiemi N^+ che descrivono i possibili elementi della coppia (Fig. 2).

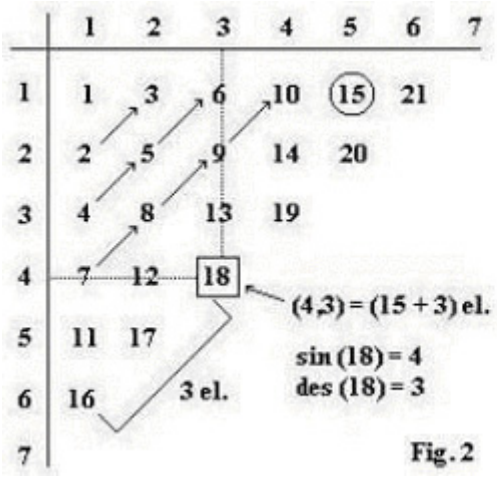


Fig. 2

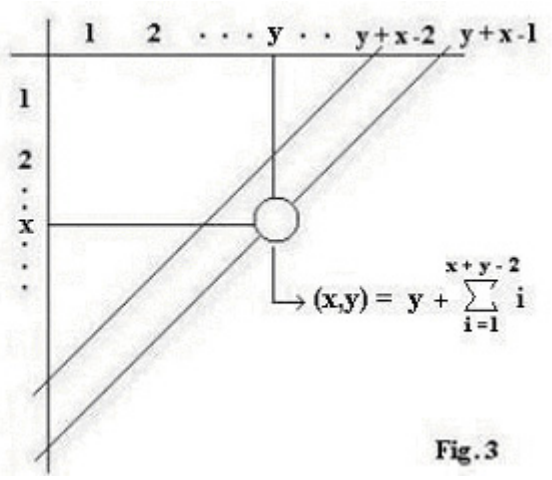


Fig. 3

A questo punto la numerazione delle coppie di elementi viene effettuata diagonalizzando a cominciare dal primo elemento in alto a sinistra (1) e proseguendo con la posizione immediatamente sotto (2) e continuando a numerare seguendo la diagonale ascendente (3), e così via...
In questo modo, ad ogni coppia di elementi x, y $\in$ N_+ viene associato un numero naturale che denotiamo proprio con l'espressione (x,y). In questo modo abbiamo definito una corrispondenza biunivoca:

$$(,) : N_+ \times N_+ \to N_+$$

che chiamiamo funzione coppia. Notiamo che l' n-sima diagonale contiene n elementi, quindi per trovare l'elemento maggiore di ciascuna basta fare la somma dei primi n numeri naturali (come in fig.2, 15 = 1+2+3+4+5). Per trovare il valore della coppia (x,y), dobbiamo conoscere il massimo elemento k della diagonale precedente e sommargli poi la coordinata y (come in fig.2, 18 = 15+3). Per calcolare quindi k, sapendo che si trova sulla diagonale numero "x + y – 2", dobbiamo sommare i primi "x + y – 2" numeri naturali. Per far ciò utilizziamo la famosa formula di Gauss:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{6}$$

Formalizzando il tutto definiamo la funzione coppia in questo modo:

$$\forall x, y \in N^+ \quad (x,y) = \left(\sum_{j=1}^{x+y-2} j \right) + y = \frac{(x+y-2)(x+y-1)}{2} + y \tag{7}$$

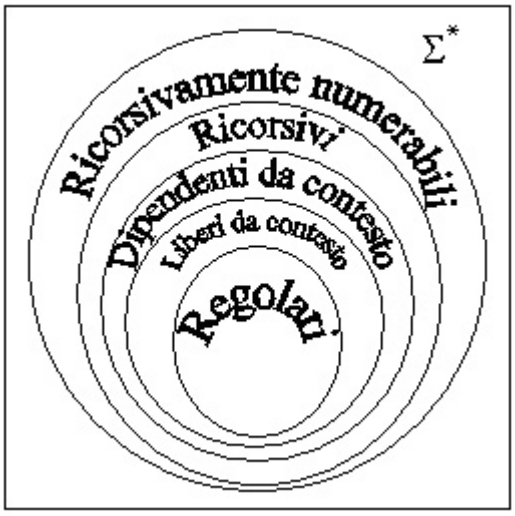
Viceversa possiamo introdurre due funzioni sin e des definite su N_+ a valori in N_+ tale che per ogni $x, y \in N^+$ se $n = (x, y)$ allora sin(n) = x e des(n) = y.

$$Sin(n) = n - Max\left\{ \frac{k(k+1)}{2} : \frac{k(k+1)}{2} < n \right\} + 1 \tag{8}$$

$$Des(n) = Max\left\{ k : \frac{k(k+1)}{2} < n \right\} - sin(n) + 2 \tag{9}$$

2 Linguaggi formali e grammatiche

Tra i concetti principali alla base dell'informatica, il concetto di linguaggio riveste un ruolo particolarmente importante. Anche se da un punto di vista matematico un linguaggio è semplicemente un insieme di stringhe su un dato alfabeto, nell'ambito dell'informatica siamo interessati a linguaggi, generalmente infiniti, le cui stringhe sono caratterizzate da qualche particolare proprietà: ad esempio, il linguaggio dei programmi C è costituito da tutte le stringhe di simboli che sono accettate da un compilatore C senza che venga rilevato alcun errore sintattico. Lo studio formale dei linguaggi è una parte importante dell'informatica teorica e trova applicazione in varie direzioni. La prima, più evidente, è appunto lo studio delle proprietà sintattiche dei programmi, cioè lo studio dei metodi di definizione della sintassi di un linguaggio di programmazione, dei metodi per la verifica che un programma soddisfi le proprietà sintattiche volute e dei metodi di traduzione da un linguaggio ad un altro (tipicamente da un linguaggio di programmazione ad alto livello al linguaggio macchina).



2.1 Linguaggi ricorsivi e ricorsivamente numerabili

Definizione 2.1 *Dato un alfabeto Σ diciamo che un linguaggio $L \subseteq \Sigma^\star$ è* **ricorsivamente numerabile** *se esiste una procedura P che stampa tutte le parole di L. Denotiamo con RN l'insieme di tutti i linguaggi ricorsivamente numerabili.*

Definizione 2.2 *Un linguaggio $L \subseteq \Sigma^\star$ è detto invece* **ricorsivo** *se esiste un algoritmo (procedura scritta in un qualunque linguaggio di programmazione che termina sempre) che su input $x \in \Sigma^\star$ restituisce il valore 1 se $x \in L$, altrimenti 0. Diciamo inoltre che questo algoritmo riconosce L.*

Proposizione 2.3 *Ogni linguaggio ricorsivo è ricorsivamente numerabile.*

Dimostrazione. Se $L \subseteq \Sigma^\star$ è ricorsivo, allora esiste un algoritmo A che riconosce L e quindi possiamo definire la seguente procedura:

for $x \in \Sigma^\star$ (in ordine militare) do
  if $x \in L$ then stampa x

Tale procedura stampa tutte le parole di L e quindi L è ricorsivamente numerabile.

Proposizione 2.4 *Se $L \subseteq \Sigma^\star$ è ricorsivo, allora L^C è ricorsivo.*

Dimostrazione. Se esiste un algoritmo A che riconosce tutte le parole $x \in L$, invertendo i valori di output otteniamo un algoritmo per riconoscere L^C.

Proposizione 2.5 *Per ogni $L \subseteq \Sigma^\star, L \in RN$ se e solo se esiste una procedura M (scritta in un qualunque linguaggio di programmazione) che su input $x \in \Sigma^\star$ soddisfa le seguenti proprietà:*

1. *M si ferma se $x \in L$,*
2. *M non si ferma se $x \notin L$.*

Dimostrazione.

- Se $L \in RN$ allora esiste una procedura P che stampa tutte le parole di L.
  Definiamo allora la seguente procedura M:

  INPUT: $x \in \Sigma^\star$

  ```
  begin
      r = no
      i = 1
      while r = no do
          y = i-esima parola stampata da P
          if y = x then r = si
                   else i = i + 1
      return 1
  end
  ```

  Tale procedura soddisfa le condizioni 1. e 2. dell'enumerato. Diciamo anche che M è una procedura di semidecisione per il linguaggio L.

- Viceversa, supponiamo che esista una procedura M che su input $x \in \Sigma^\star$ soddisfi le condizioni 1. e 2. dell'enunciato e dimostriamo che $L \in RN$. Definiamo quindi una procedura P:

  for $n \in N$ do
    i = sin(n)
    j = des(n)
    x = unrank(i)
    if M su input x termina esattamente in j passi then
      print x

  Si verifica facilmente che $x \in L \Leftrightarrow x$ è stampato da P. Infatti se $x \in L$ allora M su input x si ferma dopo k passi per qualche k; quindi quando $n =< rank(x), k >, P$ stampa x. Viceversa se $x \notin L, M$ non si ferma mai su input x e quindi P non stampa x. Questo prova che il linguaggio L appartiene a RN.

Si può inoltre provare che l'insieme dei linguaggi ricorsivamente numerabili è a sua volta numerabile. Basta elencare le procedure di semidecisione associate:

$$\begin{array}{cccccc} P_1 & P_2 & P_3 & \dots & P_n & \dots \\ \downarrow & \downarrow & \downarrow & & \downarrow & \\ L_1 & L_2 & L_3 & \dots & L_n & \dots \end{array}$$

Per la proposizione precedente tali procedure $P_1, P_2, \dots, P_n$ esistono sempre. Si può provare anche che esistono procedure $P_1, P_2, \dots, P_n$ di semidecisione per $L_1, L_2, \dots, L_n$.

Proposizione 2.6 *Esistono linguaggi ricorsivamente numerabili che non sono ricorsivi.*

Dimostrazione. Vogliamo dimostrare che esiste un linguaggio $L \in RN$ che non appartiene a R. Definiamo il linguaggio L tale che $L = \{x : x \in L_{rank(x)}\}$. Denotiamo ora con x_n la parola di $\Sigma^\star$ tale che $rank(x_n) = n$; L risulta così formato da tutte le parole x_n tale che $x_n \in L_n$. Si verifica che $L \in RN$, perchè si può costruire una procedura che stampa tutti i suoi elementi (si dimostri la proprietà per esercizio).
Consideriamo ora $A = L^C = \{x_n : x_n \notin L_n\}$; se per assurdo $A \in RN$ esisterebbe $k \in N$ tale che $A = L_k$. Ci chiediamo allora se $x_k \in A$ e si verifica:

$$\begin{cases} \text{se } x_k \in A \Rightarrow x_k \notin L_k \\ \text{se } x_k \notin A \Rightarrow x_k \in L_k \end{cases}$$

e quindi non èpossibile che un elemento appartenga ad A e allo stesso tempo non vi appartenga. Abbiamo in questo modo dimostrato che $A \notin RN$. Quindi $L \in RN$ e non appartiene a R , in quanto il suo complementare A non appartiene ad R . Di conseguenza R è propriamente incluso in RN.

La dimostrazione precedente mostra anche la seguente proprietà:

Proposizione 2.7 *RN non è chiuso rispetto al complemento.*

2.2 Grammatiche

Intuitivamete una grammatica è un insieme di regole che definiscono un linguaggio. Nel nostro contesto una grammatica è un sistema formale che consente di generare tutte le parole di un linguaggio. Una grammatica viene rappresentata da una quartupla $G =< V, \Sigma, S, P >$ nella quale:

- V è un alfabeto di simboli chiamati variabili (es. $\{A, B, C, ...\}$),
- Σ è un alfabeto di simboli chiamati terminali (es. $\{a, b, c, ...\}$) tale che $\Sigma \cap V = \emptyset$,
- $S \in V$ è un simbolo particolare detto simbolo iniziale di G,
- P è un insieme finito di elementi detti produzioni. Ogni produzione è un'espressione della forma $(\alpha \to \beta)$ dove $\alpha, \beta \in (V \cup \Sigma^\star)$, e inoltre $\alpha \neq \varepsilon$.

Questi sono due esempi di gramatiche:

$$G_1 =< \{S\}, \{a, b\}, S, \{S \to aSb, S \to \varepsilon\} >$$
$$G_2 =< \{A, B, C, S\}, \{a, b, c, d\}, S, P >$$
$$P = \{S \to ABC, A \to aAB, A \to a, BC \to CB, B \to BB, B \to b, C \to c\}$$

Definizione 2.8 *Data la grammatica* $G =< V, \Sigma, S, P >$ *chiamiamo* **relazione di derivazione in un passo**, *la relazione* $\Rightarrow_G$ *tale che:*

1. $\Rightarrow_G \quad \subseteq (V \cup \Sigma)^\star \times (V \cup \Sigma)^\star$
2. $\forall x, y \in (V \cup \Sigma)^\star$, $x \Rightarrow_G y \quad se$
   - $x = \gamma\alpha\delta$,
   - $y = \gamma\beta\delta$,
   - $(\alpha \to \beta) \in P$.

   dove $\quad \alpha, \beta, \gamma, \delta \in (V \cup \Sigma)^\star$

Esempio. Sia G la grammatica definita nel secondo punto degli esempio precedente, allora

$$AAaABC \Rightarrow_G AaABaABC \Rightarrow_G AaABaACB$$

dove $A \to aAB$ e $BC \to CB$ sono le produzioni applicate.

Definizione 2.9 *La relazione di derivazione* $\Rightarrow_G^\star$ *, è la chiusura riflessiva e transitiva di* $\Rightarrow_G$

1. $x \in (V \cup \Sigma)^\star \quad x \Rightarrow_G^\star x$
2. $\forall x, y \ (con\ x \neq y) \in (V \cup \Sigma)^\star \quad x \Rightarrow_G^\star y \ se$
   $\exists \alpha_1, \alpha_2 \ldots \alpha_n \in (V \cup \Sigma)^\star : \alpha_1 \Rightarrow_G \alpha_2 \ldots \Rightarrow_G \alpha_n \ (con\ x = \alpha_1 \ e\ y = \alpha_n)$

Esempio. Continuando l'esempio precedente, si verifica che:

$$S \Rightarrow_G ABC \Rightarrow_G aABBC \Rightarrow_G aaBBC \Rightarrow_G aaBCB \Rightarrow_G aaBBCB \Rightarrow_G \ldots \Rightarrow_G aabbcb$$
e quindi
$$S \Rightarrow_G^\star aabbcb$$

Definizione 2.10 *Ogni grammatica* $G =< V, \Sigma, S, P >$ *genera il linguaggio così definito*

$$L(G) = \{x \in \Sigma^\star : S \Rightarrow_G^\star x\} \tag{10}$$

Esempio. Consideriamo i seguenti esempi:

1. $L = \{x = a^n b^n : n \in N^+\}$

   $G_1 =< \{S, A, B\}, \{a, b\}, S, \{S \to AB, AB \to AABB, A \to a, B \to b\} >$
   $G_2 =< \{S, \}, \{a, b\}, S, \{S \to ab, ab \to aabb\} >$
   $G_3 =< \{S, \}, \{a, b\}, S, \{S \to aSb, S \to ab\} >$

   G_1, G_2, G_3 sono tre diverse grammatiche che generano lo stesso linguaggio L.

2. $L = \{a, b\}^\star$

   Una grammatica che genera questo linguaggio è

   $G =< \{S\}, \{a, b\}, S, \{S \to \varepsilon, S \to bS, S \to aS\} >$

3. $L = \{x = a^n b^n c^n : n \in N^+\}$

   Una grammatica che genera questo linguaggio è

   $G =< \{S, A, B, C\}, \{a, b, c\}, S, \{S \to ABC, S \to ASBC, A \to a, CB \to BC, aB \to abbB \to bb, bC \to bc, cC \to cc\} >$

2.3 Tipi di grammatiche

Le grammatiche vengono classificate a seconda del tipo di produzioni. Le grammatiche più generali senza alcun vincolo sulle produzioni sono chiamate *grammatiche di tipo 0.*

Teorema 2.11 *La classe dei linguaggi generati dalle grammatiche di tipo 0 coincide con la classe RN dei linguaggi ricorsivamente numerabili.*

2.3.1 Grammatiche dipendenti da contesto

Una grammatica $G =< V, \Sigma, S, P >$ si dice *dipendente da contesto*(context-sensitive) se per ogni $(\alpha \to \beta) \in P$ verifica $|\alpha| \leq |\beta|$. Le grammatiche dipendenti da contesto vengono anche chiamate di tipo 1.
Un linguaggio $L \subseteq \Sigma^\star$ si dice dipendente da contesto (context-sensitive) se esiste una grammatica di tipo 1 che genera L.
L'esempio 3 del paragrafo precedente, descrive una grammatica dipendente da contesto.

2.3.2 Grammatiche libere da contesto

Una grammatica $G =< V, \Sigma, S, P >$ si dice *libera da contesto* (context-free) se P contiene solo produzioni della forma $A \to \beta$ con $A \in V$ e $\beta \in (V \cup \Sigma)^+$.
Le grammatiche dipendenti da contesto vengono anche chiamate di tipo 2. Un linguaggio $L \subseteq \Sigma^\star$ si dice *libero da contesto* (context-free) se generato da una grammatica di tipo 2.

Proprietà 2.12 *Se un linguaggio L è libero da contesto allora L è dipendente da contesto.*

Esempio. La grammatiche $G =< \{S\}, \{a, b\}, S, \{S \to aSb, S \to ab\} >$ che genera il linguaggio $L = \{x = a^n b^n : n \in N^+\}$, è una grammatica libera da contesto.

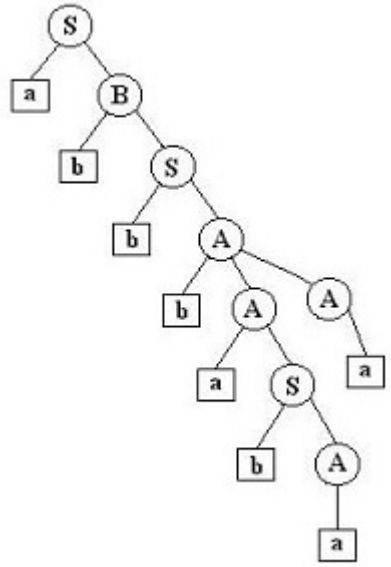
Definizione 2.13 *Un* albero di derivazione *per una grammatica libera da contesto G = <V, Σ, S, P> è un albero ordinato con nodi etichettati da elementi di (V ∪ Σ) tale che:*

- *La radice è etichettata da S,*

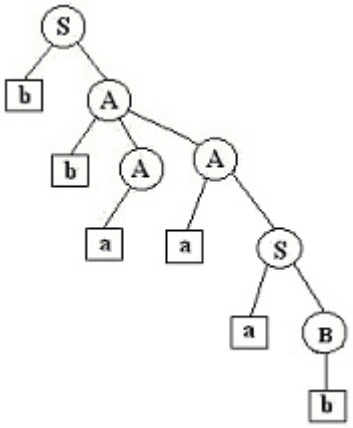
- *I nodi interni sono etichettati da simboli in V,*
- *Le foglie sono etichettate da simboli $\in \Sigma$,*
- *Se un nodo interno è etichettato da $A \in V$ e i suoi figli sono etichettati da $x_1, x_2, \ldots, x_K \in V \cup \Sigma$, nell'ordine stabilito dall'albero, allora la produzione $(A \to x_1, x_2, \ldots, x_K)$ è una produzione in P.*

La parola $x \in \Sigma^+$ derivata dall'albero si ottiene considerando la sequenza delle etichette delle foglie (da sinistra verso destra) rispettando l'ordine dell'albero.

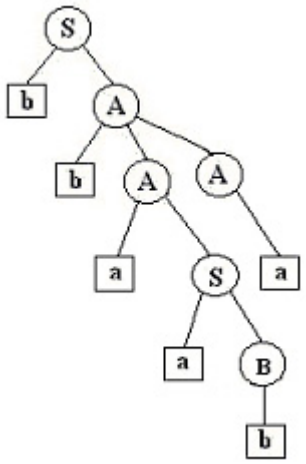
Esempio. Dalla grammatica $G =< \{S, A, B\}, \{a, b\}, S, \{S \to aB, S \to bA, A \to aS, A \to bAA, B \to bS, B \to b, B \to aBB\} >$ che genera il liguaggio $L = \{x \in \{a, b\}^+ : |x|_a = |x|_b\}$.
Rappresentiamo tramite alberi di derivazione le seguenti parole: $x = abbbabaa; x = bbaaab; x = bbaaba$.



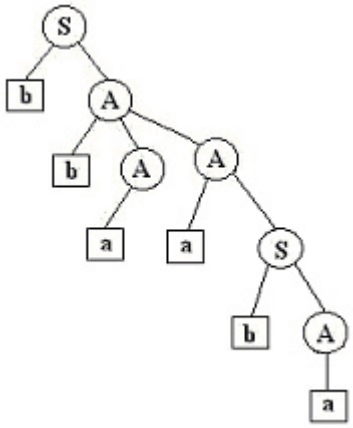
$S \Rightarrow_G aB \Rightarrow GabS \Rightarrow_G abbA \Rightarrow_G abbbAA \Rightarrow_G \Rightarrow_G abbbAa \Rightarrow_G abbbaSa \Rightarrow_G abbbabAa \Rightarrow_G abbbabaa$



$S \Rightarrow_G^\star bbaaab$



$S \Rightarrow_G^\star bbaaba$



2.3.3 Grammatiche regolari

Una grammatica $G =< V, \Sigma, S, P >$ si dice *regolare* se ogni produzione in P è della forma

$$\begin{cases} A \to aB \\ oppure \\ A \to a \end{cases} \tag{11}$$

$$dove \quad A, B \in V \quad a \in \Sigma$$

Le grammatiche dipendenti da contesto vengono anche chiamate di tipo 3.
Si dice *regolare* un linguaggio $L \subseteq \Sigma^+$ per il quale esiste una grammatica G di tipo 3 tale che $L = L(G)$.

Proprietà 2.14 *Ogni linguaggio regolare è libero da contesto e non vale sempre il viceversa.*

Esempio. Il liguaggio $L = \{x \in \{a, b\}^+ : |x|_a \text{ è pari}\}$ è generato dalla grammatica

$$G =< \{S, C\}, \{a, b\}, S, \{S \to bS, S \to aC, S \to b, C \to bC, C \to aS, C \to a\} >$$

2.4 Ricorsività e parola vuota

Teorema 2.15 *Ogni linguaggio dipendente da contesto è ricorsivo.*

Dimostrazione. Sia $L \subseteq \Sigma^\star$ un linguaggio tale che $L = L(G)$ per una particolare grammatica $G =< V, \Sigma, S, P >$ dipendente da contesto : ogni produzione in P è della forma $(\alpha \to \beta)$ con $|\beta| \geq |\alpha|$. Dobbiamo definire un algoritmo (procedura che si ferma sempre) per il seguente problema:

$$INPUT: \quad x \in \Sigma^+ \quad |x| = n$$

$$OUTPUT: \quad \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$$

Definiamo per ogni $k = 1, \ldots, n$ l'insieme T_K tale che

$$T_K = \{\alpha \in (V \cup \Sigma)^+ : |\alpha| \leq k, S \Rightarrow_G^\star \alpha\} \quad \text{se k} > 0 \text{ e con } T_0 = \{S\}$$

Vediamo ora come si definisce l'algoritmo che risolve il nostro problema.

$T_0 = \{S\}$
For $k = 1 \ldots n$ do calcolo T_K a partire da T_{K-1}
If $x \in T_n$ then return 1
else return 0

Definiamo ora la procedura che calcola T_K a partire da T_{K-1}.

$V = T_{K-1}$
Repeat
$U = V$
for $\{\gamma \in U$ AND $(\alpha \to \beta) \in P$ AND $(y, z, w) : \gamma = yzw\}$ do
if $(z = \alpha)$ AND $|y\beta w| \leq k$ then
$V = V \cup \{y\beta w\}$
until $V = U$

return V

Le definizioni precedenti descrivono grammatiche che non generano la parola vuota. E' però possibile estendere le definizioni precedenti considerando anche questo caso. A tale scopo introduciamo il seguente lemma.

Lemma 2.16 *Per ogni grammatica $G =< V, \Sigma, S, P >$ esiste una grammatica $F =< V', \Sigma, S', P' >$ tale che $L(G) = L(F)$ e nella quale S' non compare nella parte destra di alcuna produzione in P'. Inoltre se G è dipendente da contesto (rispettivamente libero da contesto o regolare) allora anche F è dipendente da contesto (rispettivamente libero da contesto o regolare)*

Dimostrazione. Definiamo la grammatica $F =< V', \Sigma, S', P' >$ tale che:

- S' nuova variabile $\notin V$,
- $P' = \{S' \to \alpha : S \to \alpha \in P\} \cup P$
(oppure, più semplicemente: $P' = \{S' \to S\} \cup P$),
- $V' = V \cup \{S'\}$.

Dimostriamo che $L(G) = L(F)$. Proviamo che per ogni $x \in \Sigma^\star$ se $S \Rightarrow_G^\star x$ allora $S' \Rightarrow_F^\star x$. Infatti esiste una catena di derivazione

$$S \Rightarrow_G \quad \alpha_1 \Rightarrow_G \quad \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \quad \alpha_n = x$$
$$\text{inoltre} \quad (S \to \alpha_1) \in P \quad \text{implica}$$
$$(S' \to \alpha_1) \in P' \quad \text{e quindi}$$
$$S' \Rightarrow_F \quad \alpha_1 \Rightarrow_F \quad \alpha_2 \Rightarrow_F \ldots \Rightarrow_F \quad \alpha_n = x \quad \text{che implica}$$
$$S' \Rightarrow_F^\star x.$$

Proviamo che per ogni $x \in \Sigma^\star$ se $S \Rightarrow_F^\star x$ allora $S' \Rightarrow_G^\star x$. Infatti esiste una catena di derivazione

$$\begin{gathered} S' \Rightarrow_F \quad \alpha_1 \Rightarrow_F \quad \alpha_2 \Rightarrow_F \ldots \Rightarrow_F \quad \alpha_n = x \\ \text{inoltre} \quad (S' \to \alpha_1) \in P' \quad \text{implica} \\ (S \to \alpha_1) \in P \quad \text{e quindi} \\ S \Rightarrow_G \quad \alpha_1 \Rightarrow_G \quad \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \quad \alpha_n = x \quad \text{cioè} \\ S' \Rightarrow_G^\star x. \end{gathered}$$

Definizione 2.17 *(Estensione delle precedenti, aggiungendo la possibilità di creare ε)*

1. $G =< V, \Sigma, S, P >$ *è* dipendente da contesto *se :*
   - *ogni produzione* $(\alpha \to \beta) \in P$ *soddisfa*
   $$\begin{cases} |\alpha| \leq |\beta| \\ \textit{oppure} \\ \alpha = S, \beta = \varepsilon \end{cases}$$
   - $(S \to \varepsilon) \in P$, *S non compare in* β *per ogni* $(\alpha \to \beta) \in P$
2. $G =< V, \Sigma, S, P >$ *è* libera da contesto *se :*
   - *ogni produzione* $(\alpha \to \beta) \in P$ *soddisfa*
   $$\begin{cases} \alpha \in V, \quad \beta \neq \varepsilon \\ \textit{oppure} \\ \alpha = S, \quad \beta = \varepsilon \end{cases}$$
   - $(S \to \varepsilon) \in P$, *S non compare in* β *per ogni* $(\alpha \to \beta) \in P$
3. $G =< V, \Sigma, S, P >$ *è* regolare *se :*
   - *ogni produzione* $(\alpha \to \beta) \in P$ *soddisfa*
   $$\begin{cases} \alpha \in V, \quad \beta = aB \quad \text{con } a \in \Sigma, B \in V \\ \textit{oppure} \\ \alpha \in V, \quad \beta \in \Sigma \\ \textit{oppure} \\ \alpha = S, \quad \beta = \varepsilon \end{cases}$$
   - $(S \to \varepsilon) \in P$, *S non compare in* β *per ogni* $(\alpha \to \beta) \in P$

Proposizione 2.18 .

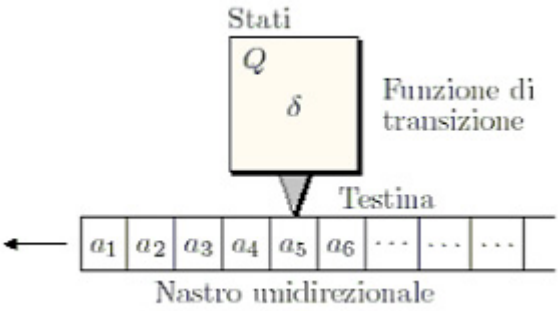
1. *Se* $L \subseteq \Sigma^\star$ *è* <u>*dipendente da contesto*</u>*, allora* $L \cup \{\varepsilon\}$ *e* $L - \{\varepsilon\}$ *sono* dipendenti da contesto.
2. *Se* $L \subseteq \Sigma^\star$ *è* libero da contesto*, allora* $L \cup \{\varepsilon\}$ *e* $L - \{\varepsilon\}$ *sono* <u>*liberi da contesto*</u>.
3. *Se* $L \subseteq \Sigma^\star$ *è* regolare*, allora* $L \cup \{\varepsilon\}$ *e* $L - \{\varepsilon\}$ *sono* regolari.

3 Linguaggi regolari

Tra i vari linguaggi che abbiamo analizzato nel capitolo precedente, quelli regolari (generati dalle grammatiche di tipo 3) sono particolarmente degni di attenzione per diversi motivi. Prima di tutto perché gli elementi sintattici principali di un linguaggio di programmazione (costanti, identificatori, parole chiave, ...) sono definibili generalmente con grammatiche di tipo 3. In secondo luogo, si può mostrare che la classe dei linguaggi regolari gode di molte proprietà algebriche, sebbene le grammatiche che li generano siano definite molto semplicemente.
In questo capitolo definiremo prima di tutto gli automi a stati finiti, che vengono utilizzati per riconoscere i linguaggi generati dalle grammatiche di tipo 3, e successivamente discuteremo le principali proprietà dei linguaggi regolari.

3.1 Automa a stati finiti

Un automa a stati finiti è intuitivamente descritto da un sistema che può assumere un insieme finito di stati, dotato di un nastro di ingresso, suddiviso in celle e di una testina di lettura. Inizialmente una stringa di ingresso formata da n simboli è disposta ordinatamente nelle prime n celle del nastro (un carattere per ogni cella), la testina di lettura è posizionata nella prima cella e il sistema si trova in uno stato particolare detto stato iniziale. Ad ogni passo, a seconda dello stato e del simbolo letto, l'automa cambia stato e muove la testina di una posizione verso destra.
Qui di seguito tratteremo automi a stati finiti deterministici e il caso più generale degli automi a stati finiti non deterministici.
Nel primo caso ad ogni mossa il nuovo stato è determinato univocamente dal carattere letto e dallo stato corrente. Nel caso non deterministico, invece, l'automa sceglie il nuovo stato in un insieme di possibili celle sempre dipendenti dal simbolo letto e dallo stato corrente.



Diamo ora la definizione formale.

Definizione 3.1 *Un **automa a stati finiti deterministico** sull'alfabeto Σ è una quartupla così definita: $A = < Q, q_0, \delta, F >$ dove:*

- *Q è un insieme finito di stati,*
- *$q_0 \in Q$ è lo stato iniziale,*
- *δ è la funzione di transizione tale che $\forall q \in Q, \forall a \in \Sigma, \delta(q, a) \in Q : Q \times \Sigma \to Q$,*

- $F \subseteq Q$ *è l'insieme degli stati finali.*

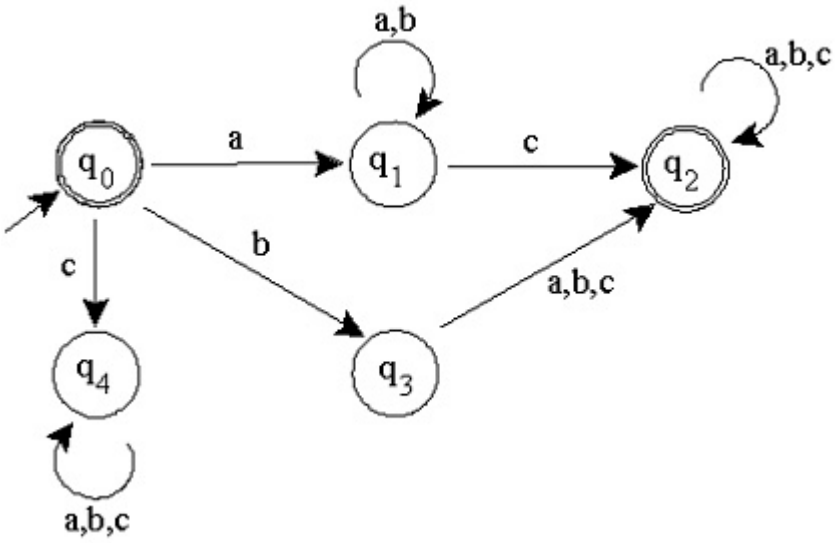
Estendiamo la definizione della funzione di transizione δ, ponendo

- $\widehat{\delta} : Q \times \Sigma^\star \to Q$ dove
- $\widehat{\delta}(q, \varepsilon) = q \quad \forall q \in Q,$
- $\widehat{\delta}(q, xa) = \delta((q, x), a) \quad \forall x \in \Sigma, \forall a \in \Sigma$.

Quindi il linguaggio riconosciuto da A è l'insieme $L(A) = \{x \in \Sigma^\star : \widehat{\delta}(q_{0,}x) \in F\}$

3.1.1 Rappresentazione grafica di un automa

L'automa $A =< Q, q_0, \delta, F >$ può essere rappresentato da un grafo orientato con etichetta sui lati, nel quale: Q è l'insieme dei nodi e per ogni $q, p \in Q$ e ogni $a \in \Sigma$, esiste un lato da q a p etichettato con a se $\delta(q, a) = p$. Denotiamo inoltre con una freccia entrante lo stato iniziale e con un doppio cerchio gli stati finali.
Esempio. Dato l'automa qui di seguito rappresentato, definito sull'alfabeto $\Sigma = \{a, b, c\}$



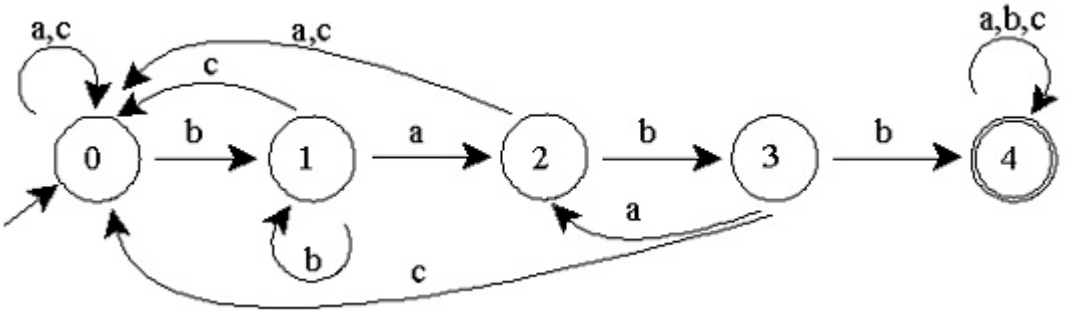
Il linguaggio riconosciuto dall'automa è così definito:

$$\{\varepsilon\} \cup \{b\} \cdot \Sigma^+ \cup \{a\} \cdot \{a, b\}^\star \cdot \{c\} \cdot \Sigma^\star$$

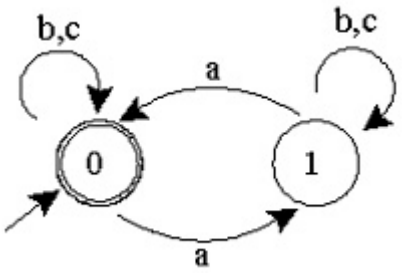
Esempio. Fissato l'alfabeto $\Sigma =$\{a, b, c\}, consideriamo il linguaggio

$$L = \{x \in \{a, b, c\}^\star : x = ybabbz, y, z \in \Sigma^\star\} \quad (quindi \quad L = \Sigma^\star babb\Sigma^\star).$$

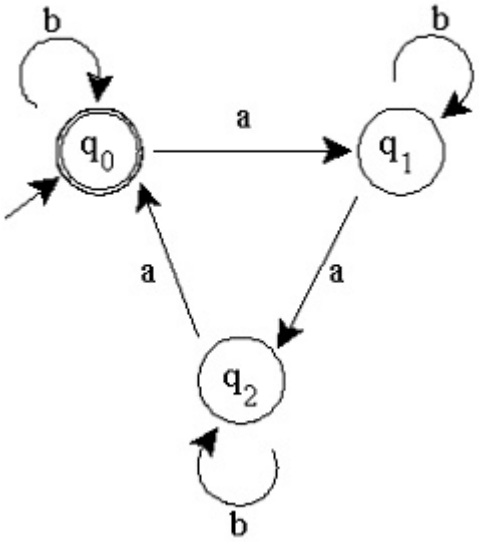
Rappresentiamo graficamente l'automa che lo riconosce:



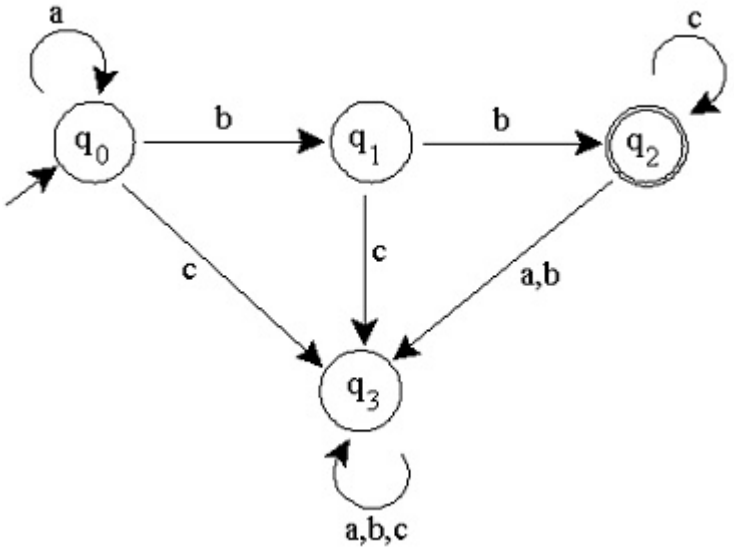
Esempio. Dato il liguaggio $L = \{x \in \{a,b,c\}^\star : |x|_a$ è pari $\}$. L'automa che riconosce il linguaggio è:



Esempio. Dato il linguaggio $L = \{x \in \{a,b\}^\star : |x|_a = 3k \quad \forall k \in \mathrm{N}\}$. L'automa che lo riconosce è:



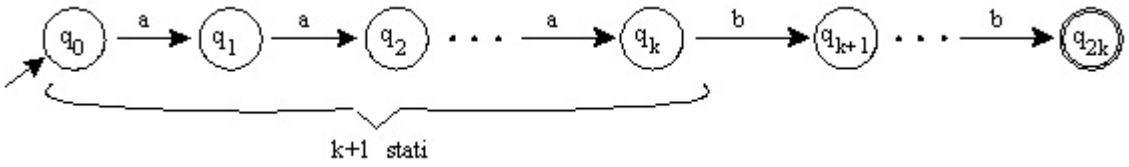
Esempio. Dato il linguaggio $L(A) = \{a\}^\star.\{bb\}.\{c\}^\star$ definito sull'alfabeto $\Sigma = \{a,b,c\}$, l'automa che lo riconosce è:



Definizione 3.2 *Si dice che un linguaggio $L \subseteq \Sigma^\star$ è riconoscibile se esiste un automa a stati finiti (deterministico) $A =< Q, q_0, \delta, F >$ su Σ tale che con $L = L(A)$.*

3.2 Lemma di iterazione

Il linguaggio $L = \{a^n b^n : n \in N\}$ non è riconoscibile tramite un automa a stati finiti, perchè il suo funzionamento richiederebbe una quantità infinita di memoria (per ricordare il numero di a letti durante il calcolo).
E' abbastanza facile intuire che il $L = \{a^n b^n : n \in N\}$ non è riconoscibile: per assurdo esista $A =< Q, q_0, \delta, F >$ che riconosce L e sia k= #Q (numero di stati). Consideriamo la stringa $a^k b^k$. La computazione di A su tale input può essere descritta dal seguente diagramma:



Essendoci solo k stati si deve necessariamente formare un ciclo. Quindi si riconoscono parole non appartenenti al linguaggio non essendoci un limite al numero di volte che un ciclo può essere ripetuto. Di conseguenza A non può riconoscere L. Altri esempi di linguaggi non riconoscibili:

$$L_1 = \{x \in \{a, b\}^\star : |x|_a = |x|_b\} \tag{12}$$

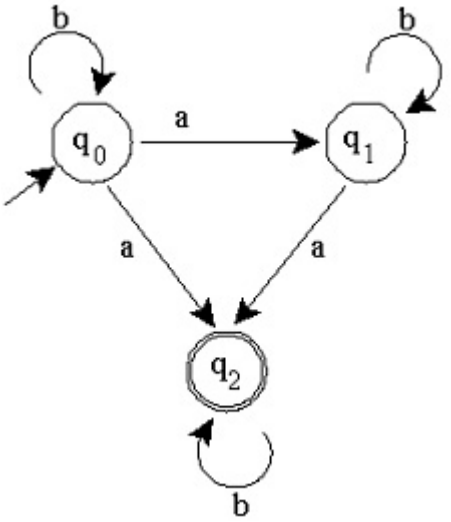
$$L_2 = \{x \in \{(,)\}^\star : x \text{ è una sequenza di parentesi correttamente innestate}\} \tag{13}$$

Il lemma di iterazione sostanzialmente dice che ogni stringa sufficientemente lunga appartenente ad un linguaggio regolare, ha una struttura che mostra delle regolarità, o meglio, contiene una sottoparola che può essere ripetuta quanto si vuole, ottenendo sempre stringhe del linguaggio.

Teorema 3.3 ***(lemma d'iterazione per i linguaggi regolari).***
Per ogni linguaggio riconoscibile $L \subseteq \Sigma^\star$, esiste un $n > 0$ tale che $\forall x \in L$ se $|x| \geq n$, esistono $u, v, z \in \Sigma^\star$ tale che:

- $x = uvz$,
- $|uv| \leq n$,
- $|v| \geq 1$,
- $\forall k \in N \quad uv^k z \in L$.

Esempio. $L = \{x \in \{a,b\}^\star : |x|_a = 2 \mod (3)\}$



$x = \underset{u}{\underline{\mathbf{ab}}} \ \underset{v}{\underline{\mathbf{b}}} \ \underset{z}{\underline{\mathbf{aaaba}}}$
$x \in$ ad L, anche $x_k = ab(b)^k aaaba, (\forall k \in N)$ appartiene ad L.

3.3 Automi non deterministici

Definizione 3.4 *Un* ***automa a stati finiti non deterministico*** *sull'alfabeto Σ è una quartupla $A = < Q, q_0, \delta, F >$ dove:*

- *Q è un insieme finito di stati,*
- *$q_0 \in Q$ è lo stato iniziale,*
- *δ è la funzione di transizione così definita $\delta : Q \times \Sigma \to P(Q)$,*
- *$F \subseteq Q$ sottoinsieme degli stati di Q, chiamato insieme degli stati finali.*

Estendiamo la definizione della funzione δ:

- $\widehat{\delta} : Q \times \Sigma^\star \to P(Q)$,
- $\forall q \in Q, \widehat{\delta}(q, \varepsilon) = \{q\}$,
- $\forall x \in \Sigma^\star, \forall a \in \Sigma \quad \widehat{\delta}(q, xa) = \bigcup \delta(p, a) \quad p \in \widehat{\delta}(q, x)$.

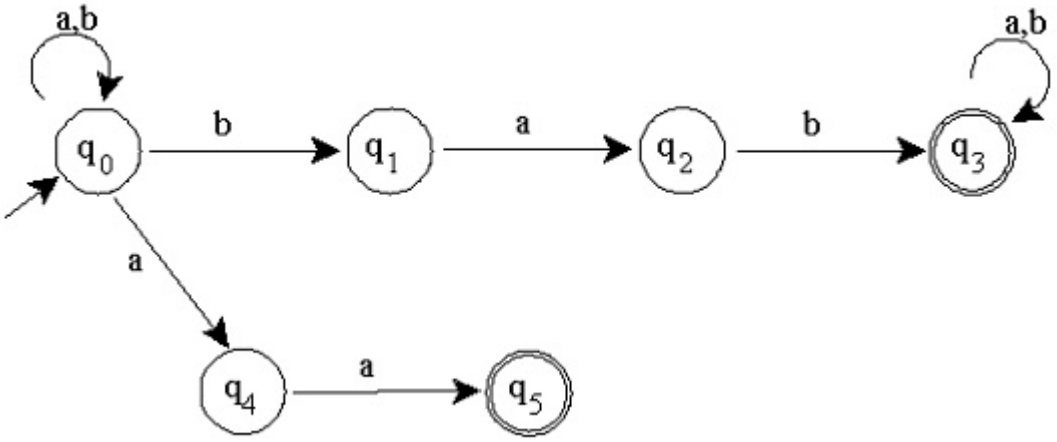
Allora il linguaggio riconosciuto dall'automa è l'insieme:

$$L(A) = \{x \in \Sigma^\star : \widehat{\delta}(q_0, x) \cap F \neq \emptyset\}.$$

Osservazione: Poichè l'automa è non deterministico, la macchina ha possibilità di scelta, quindi una parola viene accettata se tra le varie scelte, ne esiste almeno una che lo porti in uno stato finale.
Esempio. Dato il linguaggio

$$L = \{x \in \{a, b\}^\star : x \text{ contiene il fattore } bab \text{ oppure } x \text{ termina con } aa\}$$

l'automa che lo riconosce è:



Si osservi che:

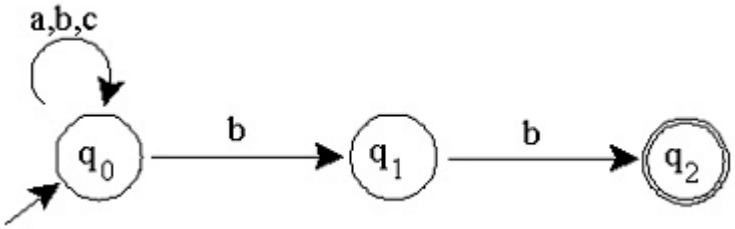
$\widehat{\delta}(q_0, abbabb) = \{q_0, q_3\}$,

$\widehat{\delta}(q_0, babaa) = \{q_0, q_3, q_5\}$,

$\widehat{\delta}(q_0, ba) = \{q_0, q_2, q_4\}$,

$\widehat{\delta}(q_{5,}a) = \emptyset$.

Esempio. Un automa non deterministico che riconosce $\{a, b, c\}^\star\{bb\}$ è definito dal seguente diagramma:



Osserviamo che $\delta(q_1, a) = \emptyset$.

I due tipi di automi che abbiamo definito in questo capitolo (deterministico e non deterministico), sembrebbero a prima vista riconoscere classi di linguaggi differenti, ma dimostriamo con il seguente teorema che questo non è vero.

Teorema 3.5 *Un linguaggio L è riconosciuto da un automa a stati finiti deterministico se e solo se L è riconosciuto da un automa a stati finiti non deterministico.*

Dimostrazione.

Se L è riconosciuto da $A =< Q, q_0, \delta, F >$ deterministico allora L è riconosciuto da $B =< Q, q_0, \delta_b, F >$ non deterministico, ponendo $\delta_b(q, a) = \{\delta(q, a)\}$ per ogni $q \in Q$ e per ogni $a \in \Sigma$. E' chiaro che i due automi si comportano in modo equivalente.

Se L è riconosciuto da $B =< Q, q_0, \delta_b, F >$ non deterministico, allora definiamo $A =< P(Q), \{q_0\}, \delta_a, F_a >$ deterministico tale che $P(Q)$ è l'insieme delle parti di Q, la funzione transizione δ_a è definita da

$$\delta_a(T, a) = \bigcup_{p \in T} \delta_b(p, a) \quad \forall T \in P(Q), \forall a \in \Sigma \quad e \quad F_a = \{T \in P(Q) : T \cap F \neq \emptyset\}$$

3.4 Automi e linguaggi regolari

Il seguente teorema dimostra che la classe dei linguaggi regolari coincide con la classe dei linguaggi riconoscibili.

Teorema 3.6 *Un linguaggio $L \subseteq \Sigma^\star$ è riconoscibile se e solo se L è regolare.*

Dimostrazione. Sia per ipotesi $L \subseteq \Sigma^\star$ riconoscibile, quindi esiste un automa $A =< Q, q_0, \delta, F >$ deterministico che riconosce L.
Trattiamo il caso in cui $q_0 \notin F$ (e quindi la parola vuota non appartiene al linguaggio riconosciuto dall'automa A). Consideriamo una grammatica $G =< V, \Sigma, S, P >$ nella quale poniamo:

- l'insieme delle variabili V corrispondente all'insime degli stati Q di A,
- il simbolo iniziale S uguale allo stato iniziale q_0,
- l'insieme delle produzioni $P = \{q \to ap : p = \delta(q, a)\} \cup \{q \to a : \delta(q, a) \in F\}$

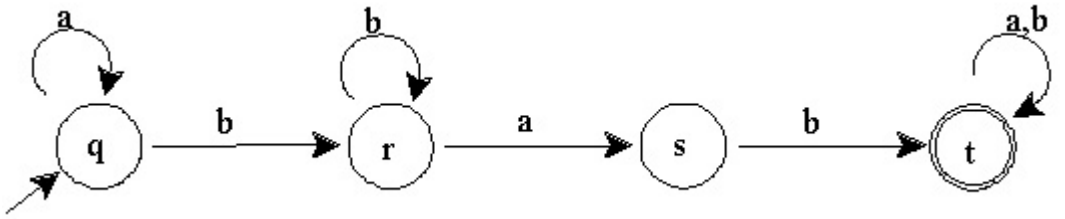
In questo modo abbiamo definito tutte le produzioni $q \to ap$ dove q e p rappresentano lo stato prima e dopo aver letto a, insieme alle produzioni della forma $q \to a$ nel caso in cui la transizione porti in uno stato finale. Così, se una parola $x = a_1 a_2 \cdots a_n$ appartiene al linguaggio L, esisteranno gli stati $q_1 q_2 \ldots q_n$ con $q_n \in F$ tali che $\delta(q_0, a_1) = q_1, \ldots, \delta(q_{n-1}, a_n) = q_n$, ed esisteranno quindi le produzioni $q_0 \to a_1 q_1, q_1 \to a_2 q_2, ..., q_{n-1} \to a_n$ con le quali possiamo generare la parola x. Abbiamo così dimostrato che se x appartiene ad L allora x appartiene anche al linguaggio generato dalla grammatica G, definita come sopra, che è proprio una grammatica regolare (di tipo 3).
Consideriamo invece ora il caso in cui $q_0 \in F$ (il linguaggio rionosciuto dall'automa A contiene la parola vuota). Costruiamo così la grammatica $G_1 =< V', \Sigma, S', P' >$ nella quale di nuovo ci sono:

- S' che è un simbolo non appartenente a Q,

- $V' = Q \cup \{S'\}$,
- l'insieme delle produzioni $P' = P \cup \{S' \to \varepsilon\} \cup \{S' \to \alpha : q_0 \to \alpha \in P\}$

Definendo la grammatica in questo modo si può anche produrre la parola vuota che è contenuta in L.

Esempio.



Partendo dall'automa sopra rappresentato, costruiamo le produzioni della grammatica G (regolare) che genera il linguaggio L riconosciuto dall'automa:

$$P = \{q \to aq, r \to br, s \to bt, t \to at, q \to br, r \to as, s \to b, t \to bt, t \to a, t \to b\}.$$

Assumiamo ora invece per ipotesi che L sia un linguaggio regolare, e quindi esiste una grammatica regolare $G =< V, \Sigma, S, P >$ che lo genera. Trattiamo ancora il caso in cui $\varepsilon \notin L$ (la parola vuota non appartine al linguaggio). Costruiamo un automa $A =< Q, q_0, \delta, F >$ nel quale:

- l'insieme degli stati Q è $V \cup \{f\}$,
- l'insieme degli stati finale F è $\{f\}$,
- lo stato iniziale q_0 coincide con S
- definiamo la funzione transizione δ come segue:

$$\forall A \in V, \quad \forall a \in \Sigma \quad \delta(A, a) = \begin{cases} \{B \in V : A \to aB \in P\} & \text{se } (A \to a) \notin P \\ \{f\} \cup \{B \in V : A \to aB \in P\} & \text{se } (A \to a) \in P \end{cases}$$

$$\delta(f, a) = \emptyset$$

Osserviamo che l'automa definito è non deterministico.
Possiamo ora affermare che se x è una parola appartenente al linguaggio generato dalla grammatica regolare G, allora x appartiene anche al linguaggio riconosciuto dall'automa A da noi costruito. Questo perchè se $x = a_1 a_2 \cdots a_n$, allora esisteranno $A_1, A_2, \ldots, A_{n-1} \in V$ e una derivazione della forma:

$$S \to a_1 A_1 \to a_1 a_2 A_2 \to a_1 a_2 a_3 A_3 \to \ldots \to a_1 a_2 \ldots a_{n-1} A_{n-1} \to a_1 a_2 \ldots a_{n-1} a_n$$

Ma questo implica che:

$$A_1 \in \delta(S, a_1)$$
$$A_2 \in \delta(A_1, a_2)$$
$$\dots$$
$$A_{n-1} \in \delta(A_{n-2}, a_{n-1})$$
$$f \in \delta(A_{n-1}, a_n)$$

e quindi $f \in \widehat{\delta}(S, x)$; di conseguenza x è riconosciuta dall'automa.

3.5 Automa minimo

Un linguaggio può essere riconosciuto da automi diversi. Uno di questi è chiamato automa minimo perchè riconosce il linguaggio con un numero minimo di stati. Per potere mostrare quanto detto introduciamo le seguenti definizioni e proposizioni.

Definizione 3.7 *Relazione di equivalenza R su un insieme S: è una relazione binaria su S che soddisfa le seguenti proprietà:*

1. *per ogni $x \in S \quad xRx$ (proprietà riflessiva),*
2. *per ogni $x, y \in S \quad xRy$ implica che yRx (proprietà simmetrica),*
3. *per ogni $x, y, z \in S \quad xRy, yRz$ implica che xRz (proprietà transitiva).*

Ricordiamo che, data una relazione di equivalenza R su un insieme S, per ogni $x \in S$ si può considerare la classe di equivalenza rappresentata da x, cioè l'insieme

$$[x]_R = \{y \in S : xRy\}$$

che contiene tutti gli elementi di S equivalenti ad x. E' evidente che l'insieme delle classi di equivalenza $\{[x]_R : x \in S\}$ rappresenta una partizione di S in sottoinsiemi disgiunti.

Definizione 3.8 *Invariante sinistro: è una relazione R su S, tale che R è di equivalenza e per ogni $x, y, z \in S$, xRy implica $zxRzy$.*

Definizione 3.9 *Invariante destro: è una relazione R su S, tale che R è di equivalenza e per ogni $x, y, z \in S, xRy$ implica $xzRyz$.*

Fissata una relazione di equivalenza R su $\Sigma^\star$, possiamo ripartire $\Sigma^\star$ in classi di equivalenza. Sia

$$\frac{\Sigma^\star}{R} = \{[x]_R : x \in \Sigma^\star\}$$

Se R è invariante destro e sinistro di $\Sigma^\star$ allora si definisce l'operazione $\odot$ tale che per ogni $x, y \in \Sigma^\star \quad [x]_R \odot [y]_R = [xy]_R$. L'operatore $\odot$ su $\frac{\Sigma^\star}{R}$ è ben definita perchè non dipende dai rappresentanti, infatti se xRx' e yRy', allora

$$[x']_R \odot [y]_R = [x'y]_R = [xy]_R = x'yRxy \quad e \quad [x]_R \odot [y']_R = [xy']_R = [xy]_R = xy'Rxy$$

perché R è invariante destro e sinistro. Per ogni relazione R invariante destra e sinistra di $\Sigma^\star$, si può dimostrare che $< \frac{\Sigma^\star}{R}, \odot, [\epsilon]_R >$ è un monoide. Esso chiamato monoide quozionte di $\Sigma^\star$ rispetto ad R. Si dice che R è di indice finito se $\frac{\Sigma^\star}{R}$ è un insieme finito. Più in generale, l'indice di R è il numero di classi di equivalenza $[x']_R$ con $x \in S^\star$. Le proprietà degli automi a stati finiti sono legate alle relazioni sintattiche definite dai linguaggi su Σ. Per ogni linguaggio $L \subseteq \Sigma^\star$ definiamo la relazione R_L tale che per ogni $x, y \in \Sigma, xRy$ se e solo se

$$\forall z \in \Sigma^\star \quad xz \in L \Leftrightarrow yz \in L$$

Proposizione 3.10 *Per ogni $L \subseteq \Sigma^\star$ la relazione R_L è un invariante destro.*

Dimostrazione. Si verificano le seguenti proprietà:

- R_L è riflessiva;
- R_L è simmetrica (perchè la definizione è simmetrica tra x e y);
- R_L è transitiva perchè se xR_Ly, yR_Lw allora $\forall z, xz \in L$ se e solo se $wz \in L$ e quindi x R_Lw;
- R_L è invariante destro infatti, per ogni $x, y, z \in \Sigma^\star$ se xR_Ly allora $\forall w \in \Sigma^\star xzw \in L \Leftrightarrow yzw \in L$ e quindi xzR_Lyz

Notiamo che ogni automa a stati finiti produce una relazione di equivalenza che è anche invariante a destra. Questo risultato viene formalizzato nel seguente teorema.

Teorema 3.11 *Le seguenti proprietà sono equivalenti:*

1. *$L \subseteq \Sigma^\star$ è un linguaggio regolare,*
2. *Esiste una relazione R' invariante a destra di indice finito tale che L è unione di alcune classi di equivalenza di R',*
3. *R_L è di indice finito.*

Dimostrazione.

1. $\Rightarrow$ 2) Se L è un linguaggio regolare, allora esiste un automa a stati finiti deterministico $A =< Q_A, q_0^A, \delta_A, F_A >$ che lo riconosce. Definiamo una relazione R' dove:

   $$\forall x, y \in \Sigma^\star \quad xR'y \Leftrightarrow \delta(q_0^A, x) = \delta(q_0^A, y).$$

   E' facile verificare che R' è una relazione di equivalenza. Inoltre R' è anche un invariante a destra perché:

   $$\forall x, y, z \in \Sigma^\star \quad \delta(q_0^A, x) = \delta(q_0^A, y) \Rightarrow \delta(q_0^A, xz) = \delta(q_0^A, yz).$$

   Infine R' è di indice finito perché l'indice è al più il numero di stati dell'automa. Si verifica ora che L è l'unione delle classi di equivalenza che includono un elemento x tale che $\delta(q_0^A, x)$ appartenga a F_A.

2. $\Rightarrow$ 3) Sappiamo che $xR'y$. Essendo R' invariante destro, per ogni $z \in \Sigma, xzR'yz$, e così yz appartiene a L se e solo se xz vi appartiene. Quindi xR_Ly. Abbiamo così dimostrato che $xR'y$ implica xR_Ly. Quindi l'indice di R' è maggiore o uguale all'indice di R_L. Essendo R' di indice finito, anche R_L lo è.

3. $\Rightarrow$ 1) Consideriamo xR_Ly. Ora prendiamo l'insieme delle classi di equivalenza di R_L e chiamiamolo Q_M; sia $[x]$ l'elemento di Q_M che contiene x. Definiamo $\delta_M([x], a) = [xa]$. Questa definizione è ben posta perché R_L è invariante destro. Consideriamo $q_0^M = [\varepsilon]$ e $F_M = \{[x] \mid x \in L\}$. L'automa a stati finiti $M =< Q_M, q_0^M, \delta_M, F_M >$ riconosce L perché $\delta_M(q_0^M, x) = [x]$, e così x appartiene al linguaggio riconosciuto da M se e solo se $[x]$ sta in F_M.

Osservazione. Se $x \in L$ e xR_Ly allora $y \in L$.

Teorema 3.12 ***Teorema dell'automa minimo***. *Per ogni linguaggio regolare L esiste (a meno di modificare il nome degli stati) un solo automa a stati finiti deterministico che riconosce L con il minimo numero di stati.*

Dimostrazione. Proviamo che l'automa M definito nella dimostrazione precedente è proprio il minimo per il linguaggio L che riconosce.
Sia $A =< Q, q_0, \delta, F >$ un qualsiasi automa deterministico che riconosce L tale che per ogni $q \in Q$, esiste $w \in \Sigma^\star$ tale che $\delta(q_0, w) = q$. Vogliamo dimostrare che esiste una funzione suriettiva $f : Q \to Q_M$ e vogliamo provare che:

$$\forall q \in Q, \quad \forall a \in \Sigma \quad f(\delta(q, a)) = \delta_M(f(q), a)$$

Per ogni $q \in Q$ scegliamo $w \in \Sigma$ tale che $\delta(q_0, w) = q$ e definiamo $f(q) = [w]R_L$. La definizione è ben posta. Infatti se esiste $y \neq w$ tale che $\delta(q_0, y) = q$ allora per ogni $z \in \Sigma^\star \quad \delta(q_0, wz) = \delta(q_0, yz)$ e quindi:

$$wz \in L \Leftrightarrow yz \in L \Rightarrow yR_Lw \Rightarrow [w]R_L = [y]R_L$$

Inoltre la funzione f è suriettiva, infatti:

$$\forall [w]R_L \in Q_M \quad \exists q = \delta(q_0, w) \in Q \quad t.c. \quad f(q) = [w]R_L$$

Infine f conserva le transizioni: per ogni $a \in \Sigma$ e per ogni $q \in Q$, se $f(q) = [w]R$ allora:

$$f(\delta(q, a)) = [wa]R_L = [w]R_L \odot [a]R_L = \delta_M(f(q), a)$$

3.6 Proprietà di chiusura (rispetto alle operazioni sui linguaggi)

1. Se $L \subseteq \Sigma^\star$ è un linguaggio regolare, allora anche L^c è un linguaggio regolare. Se L è regolare, allora esiste un automa deterministico $A =< Q, q_0, \delta, F >$ che riconosce L e quindi $B =< Q, q_0, \delta, Q - F >$ è un automa deterministico che riconosce L^c.

2. Siano $I \subseteq \Sigma^\star$ e $J \subseteq \Sigma^\star$ due linguaggi regolari. Proviamo che:

   (a) $K = I \cup J$ è regolare,

   (b) $T = I \cap J$ è regolare,

   (c) $H = I \cdot J$ è regolare.

   Sia $A =< Q_I, q_0^I, \delta_I, F_I >$ un automa deterministico che riconosce I e $B =< Q_J, q_0^J, \delta_J, F_J >$ un automa deterministico che riconosce J e $B =< Q_J, q_0^J, \delta_J, F_J >$ un automa deterministico che riconosce J

(a) Costruiamo un nuovo automa $C =< Q_J \times Q_I, (q_0^J, q_0^I), \delta_C, F_C >$ tale che:

$\forall (q,p) \in Q_J \times Q_I,$
$\forall a \in \Sigma \quad \delta_c((q,p),a) = (\delta_I(q,a), \delta_J(p,a)),$
$F_C = \{(q,p) \in Q_J \times Q_I \mid q \in F_I \text{ oppure } p \in F_J\}.$

(b) Costruiamo un nuovo automa $D =< Q_J \times Q_I, (q_0^J, q_0^I), \delta_D, F_D >$ tale che:

$\forall (q,p) \in Q_J \times Q_I \ ,$
$\forall a \in \Sigma \quad \delta_D((q,p),a) = (\delta_I(q,a), \delta_J(p,a)),$
$F_D = \{(q,p) \in Q_J \times Q_I \mid q \in F_I \text{ e } p \in F_J\}.$

Si verifica che D riconosce T e quindi anche T è un linguaggio regolare.

(c) $H = \{w \in \Sigma^\star \mid \exists x \in I \text{ e } y \in J : w = xy\}$
Costruiamo un nuovo automa a stati finiti non deterministico $E =< Q_J \cup Q_I, q_0^I, \delta_E, F_E >$ che riconosca H.

$$\delta_E(q,a) = \begin{cases} \{\delta_I(q,a)\} & se \quad q \in Q_I - F_I \qquad (i) \\ \{\delta_I(q,a), \delta_J(q_0^J,a)\} & se \quad q \in F_I \qquad (ii) \\ \{\delta_J(q,a)\} & se \quad q \in Q_J \qquad (iii) \end{cases}$$

i. Permette ad E di agire come A per un segmento iniziale dell'input.
ii. Permette ad E di continuare la simulazione di A oppure permette di individuare il simbolo di inizio di una parola appartenente al linguaggio J, assicurandosi che il simbolo precedente completi una parola del linguaggio I.
iii. Permette solo la simulazione di B dopo che E ha verificato che la parola del linguaggio J è iniziata.

$$F_E = \begin{cases} F_J & se \quad q_0^J \notin F_J \\ F_I \cup F_J & altrimenti \end{cases}$$

Si nota che E riconosce H e quindi H è regolare.

3. Se $L \subseteq \Sigma^\star$ è regolare allora $L^\star$ è regolare.
Sia $A =< Q, q_0, \delta, F >$ un automa deterministico che riconosce L. Definiamo l'automa non deterministico $B =< Q_B, q_0^B, \delta_B, F_B >$:

$$Q_B = Q \cup \{q_0^B\} \qquad F_B = F \cup \{q_0^B\}$$
$$\forall q \in Q, \forall a \in \Sigma \qquad \delta(q_0^B, a) = \delta(q_0, a)$$
$$\delta_B(q,a) = \begin{cases} \delta(q,a) & se \quad q \notin F \\ \{\delta(q,a), \delta(q_0,a)\} & se \quad q \in F \end{cases}$$

Si verifica che B riconosce $L^\star$.

3.7 Teorema di Kleene

Il teorema di Kleene mette in relazione i linguaggi regolari con le operazioni razionali di $\Sigma^\star$. Ricordiamo che le operazioni razionali sui sottoinsiemi di $\Sigma^\star$ sono le operazioni di unione, prodotto e $\star$ (detta anche chiusura di Kleene) definite nel primo capitolo. Queste operazioni permettono di definire particolari espressioni chiamate espressioni regolari su un dato alfabeto Σ. Queste sono definite formalmente nel modo seguente:

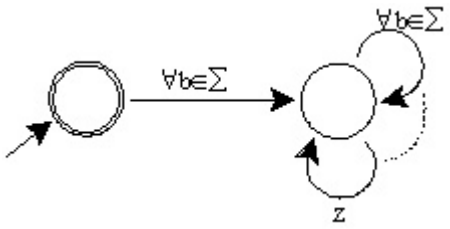
1. La parola vuota ε è un'espressione regolare,
2. Un qualsiasi simbolo $\sigma \in \Sigma$ è un'espressione regolare,
3. Se α e β sono espressioni regolari allora anche $(\alpha \cup \beta), (\alpha \cdot \beta), \alpha^\star$ sono espressioni regolari.

Data un'espressione regolare E su Σ esiste un solo linguaggio $\|E\| \subseteq \Sigma^\star$ rappresentato da E in modo ovvio. Il teorema di Kleene afferma che un linguaggio $L \subseteq \Sigma^\star$ è regolare se e solo se L rappresentabile da un'espressione regolare su Σ. In altre parole un linguaggio $L \subseteq \Sigma^\star$ è regolare se e solo se L si può ottenere dagli insiemi finiti in $\Sigma^\star$ usando le operazioni unione, prodotto e $\star$. Questo significa che la classe dei linguaggi regolari su Σ è la più piccola classe di linguaggi su Σ contenente i linguaggi finiti e chiusa rispetto alle operazioni razionali.

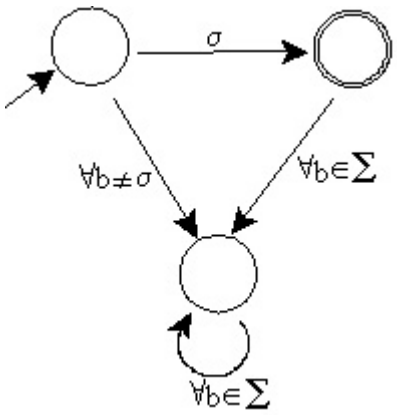
Teorema 3.13 *Per ogni linguaggio $L \subseteq \Sigma^\star$, L è regolare se e solo se L è rappresentabile da un'espressione regolare su Σ.*

Dimostrazione.

1. Dobbiamo dimostrare che se E è un'espressione regolare allora $\|E\|$ è regolare.
   Se $E = \varepsilon$ allora l'automa definito nella seguente figura riconosce il linguaggio $\|E\|$.



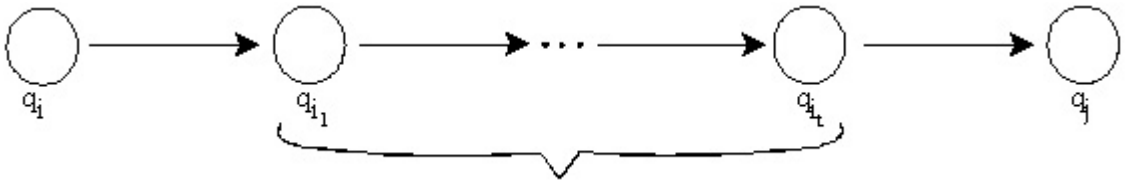
   Se $E = \sigma$ con $\sigma \in \Sigma$, il seguente automa riconosce il proprio $\{\sigma\}$



In fine, se E_1, E_2 sono espressioni regolari e $A1, A2$ sono gli automi a stati finiti che riconoscono $\|E_1\|, \|E_2\|$ allora, per un teorema dimostrato nella sezione precedente, esiste B automa a stati finiti che riconosce $\|E_1\| \cup \|E_2\|$. Quindi $E = (E_1 \cup E_2)$ rappresenta un linguaggio $\|E\|$ riconosciuto da B e di conseguenza regolare. Inoltre esistono gli automi C e D che riconoscono rispettivamente $\|E_2\| \cdot \|E_2\|$ e $\|E_1\|^\star$ e quindi $F = (E_1 \cdot E_2)$ e $G = E_1^\star$ rappresentano linguaggi regolari.

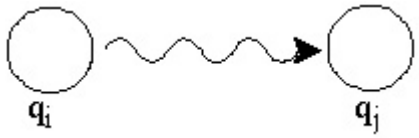
2. Se $L \subseteq \Sigma^\star$ è riconosciuto da un automa a stati finiti (deterministico) allora esiste un'espressione regolare E tale che $L = \|E\|$. Consideriamo $A =< Q, q_1, \delta, F >$ automa a stati finiti deterministico che riconosce il linguaggio L. L'insieme Q sia formato dagli stati $q_1, q_2, \ldots, q_m$. Ora per ogni $i, j, k = 1, 2, \ldots, m$ definiamo:

$$R_{ij}^k = \{x \in \Sigma \mid \delta(q_i, x) = q_j \text{ con stati intermedi } q_r \text{ tali che } r \leq k\}.$$



$$i_1, i_2, \ldots, i_k \leq k$$

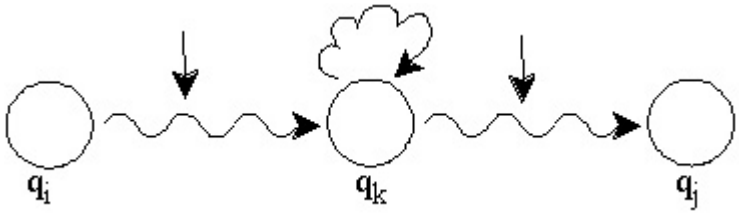
Ad esempio $R_{ij}^0 = \{a \in \Sigma^\star \mid \delta(q_i, a) = q_j\}$ rappresenta il passaggio diretto da q_i a q_j, senza passare per stati intermedi.



Quindi $L = \bigcup_{q_j \in F} R_{ij}^m$.

Ora dobbiamo provare che R_{ij}^k è rappresentato da un'espressione regolare per ogni i, j, k. Ragioniamo per induzione su $k = 0, 1, \ldots, m$; con $k = 0$ la proprietà è banalmente vera per tutti gli insiemi R_{ij}^0. Supponiamo $k \geq 1$ e assumiamo che R_{ij}^r sia rappresentata da un'espressione regolare per ogni $r = 0, 1, \ldots, k-1$. Si verifica che

$$R_{ij}^k = R_{ij}^{k-1} \cup (R_{ik}^{k-1} \cdot R_{kk}^{k-1} \cdot R_{kj}^{k-1}).$$



Di conseguenza anche R_{ij}^k è rappresentabile da un'espressione regolare e quindi anche il linguaggio

$$L = \bigcup_{q_j \in F} R_{ij}^m.$$

4 Linguaggi liberi da contesto

Una *Grammatica libera da contesto* è una quartupla $G =< V, \Sigma, S, P >$ dove V è un insieme finito di variabili (simboli non terminali), Σ è un insieme finito di simboli terminali (alfabeto finito), $S \in V$ è chiamato simbolo iniziale della grammatica e P è un insieme finito di produzioni della forma $A \to \beta$ con $A \in V, \beta \in (\Sigma \cup V^+)$ e con eventualmente la produzione $S \to \varepsilon$ (se succede questo, S non compare mai nella parte destra di alcuna produzione in P).
Esempio. Definiamo la grammatica $G =< \{S\}, \{(,)\}, S, \{S \to \varepsilon, S \to (S)S\} >$ tale che il corrispondente linguaggio generato sia l'insieme di tutte le parole formate da parentesi correttamente innestate e proviamo ad esempio a derivare la parola ( )(( )). Possiamo ottenerla con i seguenti passaggi:

$$S \Rightarrow_G (S)S \Rightarrow_G ()S \Rightarrow_G ()(S)S \Rightarrow_G ()((S)S)S \Rightarrow_G ()(()).$$

Trasformiamo ora la grammatica G in una grammatica $G1$ libera da contesto. Per far ciò dobbiamo modificare le produzioni di G affinché S non compaia mai nelle parte destra. Per generare tutte le possibili parole del linguaggio possiamo ad esempio usare le seguenti produzioni:

$$S \to (A)A, S \to (), S \to ()A, S \to (A), S \to \varepsilon, A \to (A)A, A \to (), A \to ()A, A \to (A)$$
$$\text{e quindi} \quad G_1 =< \{S, A\}, \{(,)\}, S, P >$$
$$P = \{S \to (A)A, S \to (), S \to ()A, S \to (A), S \to \varepsilon, A \to (A)A, A \to (), A \to ()A, A \to (A)\}$$

4.1 Forma normale di Chomsky

Definizione 4.1 *Una grammatica libera da contesto $G =< V, \Sigma, S, P >$ è in forma normale di Chomsky se ogni produzione in P è del tipo $A \to a$ oppure $A \to BC$ con $a \in \Sigma \quad A, B, C \in V$.*

Vogliamo dimostrare che per ogni grammatica libera da contesto G esiste una grammatica(libero da contesto) in forma normale di Chomsky che genera lo stesso linguaggio.
Esempio. Consideriamo $L = \{x \in \{a, b\}^+ : |x|_a = |x|_b\}$ e sia

$$G =< \{A, B, S\}, \{a, b\}, S, \{S \to aB, S \to bA, A \to a, A \to aS, A \to bAA, B \to b, B \to bS, B \to aBB\} >$$

Tale grammatica non è una grammatica in forma normale di Chomsky perché solo le produzioni $A \to a, B \to b$ soddisfano la definizione data.
Mostriamo come si può ottenere la un'altra grammatica in forma normale di Chomsky che genera lo stesso linguaggio L. Aggiungiamo due nuove variabili C, D e consideriamo l'insieme di produzioni

$$\{C \to a, D \to b, S \to CB, S \to DA, A \to CS, B \to DS, A \to a, B \to b\}$$

Tale insieme simula le produzioni

$$S \to aB, S \to bA, A \to a, A \to aS, A \to bAA, B \to b, B \to bS, B \to aBB$$

Dobbiamo ora trovare il modo di simulare le produzioni $A \to bAA, B \to aBB$. A tale scopo introduciamo le variabili F e G e aggiungiamo all'insieme precedente le produzioni

$$E \to DA, F \to CB, A \to EA, B \to FB$$

Quindi la nuova grammatica G_1 in forma normale di Chomsky sarà:

$$G_1 =< \{S, A, B, C, D, E, F\}, \{a, b\}, S, P_1 > \quad \text{dove}$$
$$P_1 = \{A \to a, B \to b, C \to a, D \to b, S \to CB, S \to DA, A \to CS,$$
$$A \to EA, B \to DS, B \to FB, E \to DA, F \to CB\}$$

Cerchiamo ora di estendere questo esempio al caso generale. Supponiamo che $G =< V, \Sigma, S, P >$ sia libera da contesto e che $L(G) \subseteq \Sigma^+$. Possiamo costruire una grammatica equivalente in forma normale di Chomsky attraverso i seguenti passi.

1. Eliminare tutte le variabili inutili e le relative produzioni,cioè tutte le variabili che non compaiono nella derivazione di qualche parola ω a partire da S,

2. Eliminare tutte le produzioni unitarie, cioè quelle della forma $A \to B, \quad A, B \in V$. Esiste un algoritmo per calcolare su input $A \in V$, l'insieme delle variabili $\{B \in V \mid A \Rightarrow_G^\star B\}$

   begin
   V = {A}
   repeat
   U = V
   for $B \in U$ do
   for $B \to C \in P$ do
   if $C \notin V$ then V = V ∪ {C}
   until V = U
   end

   Mettiamo in $R\{A \to \alpha \in P \mid \alpha \notin V\}$

   repeat
   T = R
   for $A, B \in V (A \neq B)$ do
   if $A \Rightarrow_G^\star B$ then
   for $B \to \beta \in T$ do
   $R = R \cup \{A \to B\}$
   until T = R

3. Modificare la grammatica in modo da avere solo produzioni della forma $A \to a$ e $A \to \alpha$ con $A \in V, a \in \Sigma$ e $\alpha \in V^\star$ con $|\alpha| \geq 2$.
   Inseriamo nella grammatica tutte le produzione del tipo $R_a \to a$. Sostituiamo tutte le produzioni del tipo $A \to \alpha (|\alpha| \geq 2)$ con $A \to \alpha'$.

4. Eliminare le produzioni della forma $A \to \alpha$ con $A \in V, \alpha \in V^\star \quad |\alpha| \geq 3$.

5. $A \to B_1 B_2 \dots B_k$ con $k \geq 3, A_1, B_1, \dots B_k$ variabili definiamo $D_1 D_2 \dots D_{k-2}$ e le produzioni $A \to B_1 D_1, D_1 \to B_2 D_2, D_2 \to B_3 D_3, \dots, D_{k-2} \to B_{k-1} B_k$. Sostituiamo ogni produzione $A \to B_1 B_2 \dots B_k$ con $k \geq 3$ e $A_1, B_1, \dots B_k$ variabili con le produzioni $A \to B_1 D_1, D_1 \to B_2 D_2, D_2 \to B_3 D_3, \dots, D_{k-2} \to B_{k-1} B_k$.

Esempio. La produzione $A \to BCEF$ diventa $A \to BD_1 \quad D_1 \to CD_2 \quad D_2 \to EF$

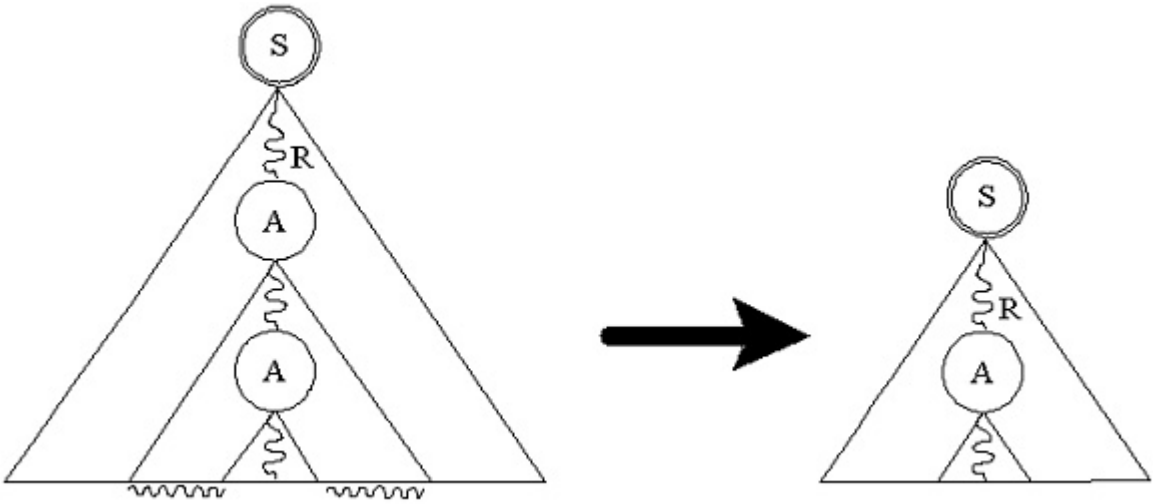
4.1.1 Eliminazione delle variabili inutili

Questo passo è basato sul seguente risulatato.

Lemma 4.2 *Esiste un algoritmo per il seguente problema:*

$$INPUT: G =< V, \Sigma, S, P >$$
$$\textit{(grammatica libera da contesto)}$$
$$OUTPUT : \begin{cases} 1 & \textit{se } L(G) \neq \emptyset \\ 0 & \textit{altrimenti} \end{cases}$$

Dimostrazione. L'algoritmo che descriviamo è basato sulla seguente osservazione:
Supponiamo che esista $\omega \in \Sigma^\star$ tale che S $\Rightarrow_G^\star w$ e consideriamo un albero di derivazione per ω. Se tale albero possiede due nodi etichettati con la stessa variabile $A \in V$ lungo un cammino dalla radice a una foglia allora si può modificare l'albero di derivazione nel modo indicato in figura, cioè sostituendo il sottoalbero più grande di radice A con quello più piccolo.



Possiamo ripetere questa operazione per tutte le coppie di nodi che hanno la stessa etichetta e si trovano in un cammino dalla radice alla foglia. Otteniamo così un albero di derivazione di altezza al più pari al numero di variabili. Di conseguenza l'algoritmo per risolvere il problema, esegue i seguenti passi:

1. Genera tutti gli alberi di derivazione in G che hanno per altezza il numero di variabili #V,
2. Controlla se tra questi ne esiste uno le cui foglie sono tutte etichettate da simboli terminali. In caso affermativo restituisci 1 altrimenti restituisci 0.

Proposizione 4.3 *Per ogni grammatica $G =< V, \Sigma, S, P >$ libera da contesto tale che $L(G) \neq \emptyset$, esiste una grammatica F libera da contesto e priva di simboli inutili tale che $L(F) = L(G)$.*

Dimostrazione. Usando il lemma precedente possiamo calcolare l'insieme

$$R = \{A \in V \mid \exists w \in \Sigma^\star : A \Rightarrow_G^\star w\}$$

Di conseguenza definiamo l'insieme

$$T = \{A \to \alpha \in P \mid A \in R, \quad \alpha \text{ contiene solo variabili in R e simboli in } \Sigma \}$$

e le grammatica $H =< R, \Sigma, S, T >$. Si verifica facilmente che $L(G) = L(H)$.
Calcolo l'insieme

$$U = \{A \in R \mid S \Rightarrow_H^\star \alpha A \beta \quad \text{per qualche } \alpha, \beta\}$$

Tale insieme è ottenuto dalla seguente procedura:

```
begin
    U ={S}
    repeat
        V = U
        for A ∈ U do
            for A → α ∈ T do
                for B ∈ R in α do
                    V = V ∪ {B}
    until V = U
    return U
end
```

Definiamo quindi l'insieme

$$W = \{A \to \alpha \in T \mid A \in U, \quad \alpha \text{ contiene solo variabili in } U\}$$

e la grammatica $F =< U, \Sigma, S, W >$. Si verifica di nuovo che $L(F) = L(H)$ e inoltre F possiede solo variabili utili.

4.1.2 Eliminazione delle produzioni unitarie

Questo passo è basato sulla seguente proprietà.

Proposizione 4.4 *Per ogni grammatica libera da contesto $G =< V, \Sigma, S, P >$ esiste una grammatica libera da contesto G' che genera lo stesso linguaggio privo di produzioni della forma $A \to B$ dove A e B sono 2 variabili.*

Dimostrazione. Esiste un algoritmo per calcolare su input $A \in V$, l'insieme delle produzioni $\{B \in V : A \Rightarrow_G^\star B\}$

```
begin
    V = {A}
    repeat
        U = V
        for B ∈ U do
            for B→C ∈ P do
                if C ∉ V then V = V ∪ {C}
    until V = U
    return U
end
```

Possiamo così definire la seguente procedura:

repeat
$R = \{A \to \alpha \in P : \alpha \notin V\}$
T = R
for $A, B \in V (A \neq B)$ do
if $A \Rightarrow_G^\star B$ then
for $B \to \beta \in T$ do
$R = R \cup \{A \to B\}$
until T = R
return R

4.1.3 Costruzione della grammatica in forma normale di Chomsky

Gli ultimi due passi del procedimento di costruzione della forma normale di Chomsky, possono essere descritte nel modo seguente.
Supponiamo di avere una grammatica libera da contesto $G =< V, \Sigma, S, P >$, priva di variabili inutili e di produzioni unitarie (ottenuta così attraverso i due passi precedenti). Allora per ogni $a \in \Sigma$ definiamo una nuova variabile R_a e una nuova produzione $R_a \to a$. Per ogni produzione $A \to \alpha$ (con $\alpha \in (V \cup \Sigma)^+ : |\alpha| \geq 2$) definiamo α' ottenuto da α sostituendo tutte le occorrenze di un simbolo terminale $a \in \Sigma$ con la variabile R_a. Ad esempio la produzione $A \to aBbCCDd$ diventa $A \to R_aBR_bCCDR_d$. Inseriamo nella grammatica tutte le produzione del tipo $R_a \to a$. Sostituiamo tutte le produzioni del tipo $A \to \alpha(|\alpha| \geq 2)$ con $A \to \alpha'$. Abbiamo così ottenuto una nuova grammatica G' equivalente a G che possiede solo produzioni del tipo $A \to a$ e $A \to \alpha$ con α stringa di variabili di lunghezza ≥ 2 . Eliminiamo ora le produzioni $A \to \alpha$ con $|\alpha| \geq 3$. Consideriamo una produzione del tipo $A \to B_1B_2 \dots B_k$ con $k \geq 3, A_1, B_1, \dots B_k$ variabili. Definiamo le nuove variabili $D_1D_2 \dots D_{k-2}$ e le produzioni $A \to B_1D_1, D_1 \to B_2D_2, D_2 \to B_3D_3, \dots, D_{k-2} \to B_{k-1}B_k$. Sostituiamo la produzione $A \to B_1B_2 \dots B_k$ con le produzioni $A \to B_1D_1, D_1 \to B_2D_2, D_2 \to B_3D_3, \dots, D_{k-2} \to B_{k-1}B_k$. Ad esempio la produzione $A \to BCEF$ può essere simulata dalle produzioni $A \to BD_1, D_1 \to CD_2, D_2 \to EF$.

4.2 Derivazioni leftmost

Una derivazione $S \Rightarrow_G \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \dots \Rightarrow_G \alpha_n$ in una grammatica libera da contesto $G =< V, \Sigma, S, P >$ con $\alpha_i \in (V \cup \Sigma)^\star$ per ogni $i = 1 \dots n$ si dice di tipo **leftmost**, se per ogni $i = 1, \dots, n-1, \quad \alpha_i \Rightarrow_G \alpha_{i+1}$ si ottiene applicando una produzione alla prima variabile a sinistra di α_i .
Esempio. Dato il seguente insieme di produzioni

$$S \to aAB, A \to BBC, B \to BA, B \to bc, A \to BB, C \to c$$

applicando derivazioni di tipo leftmost ottengo

$$S \Rightarrow aAB \Rightarrow aBBCB \Rightarrow abcBCB \Rightarrow abcbcCB \Rightarrow abcbccB \Rightarrow abcbccb$$

Osservazione. Esiste una corrispondenza biunivoca tra alberi di derivazione e derivazioni leftmost; infatti, dato un albero di derivazione, si può costruire una sola derivazione leftmost che rappresenta l'albero.

Proposizione 4.5 *In ogni grammatica libera da contesto* $G =< V, \Sigma, S, P >$ *una parola* $\alpha \in (V \cup \Sigma)^+$ *può essere derivata da* S *se e solo se esiste una derivazione leftmost* $S \Rightarrow_G^\star \alpha$

Dimostrazione.

- proprietà è ovvia perchè una derivazione leftmost è una particolare derivazione,
- è sufficiente riordinare i singoli passi delle derivazioni in modo da ottenere una derivazione di tipo leftmost

4.3 Algoritmo CYK

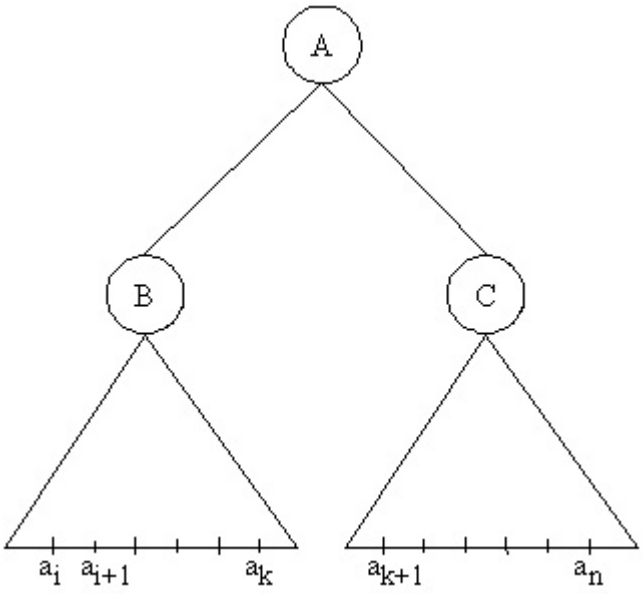
Prende il nome dalle iniziali dei suoi creatori: Cocke, Young, Kasami e risolve il problema di determinare l'appartenenza di una parola ad un linguaggio libero da contesto generato da una grammatica in forma normale di Chomsky.
Ricordiamo che se $L \subseteq \Sigma^\star$ è libero da contesto, esiste una grammatica libera da contesto $G =< V, \Sigma, S, P >$ in forma normale di Chomsky (ossia con produzioni del tipo $A \to a, A \to BC$) che genera L. Consideriamo la grammatica $G =< V, \Sigma, S, P >$ in C.N.F. L'algoritmo CYK risolve il seguente problema:

Problema Appartenenza ($G =< V, \Sigma, S, P >$)

INPUT: $x \in \Sigma^\star$

$$OUTPUT : \begin{cases} 1 & \text{se } S \Rightarrow_G^\star x \\ 0 & \text{altrimenti} \end{cases}$$

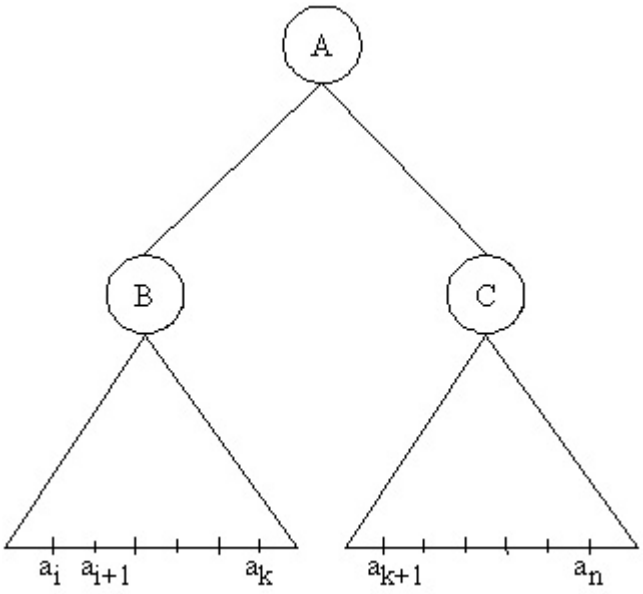
E' un algoritmo basato sul metodo della programmazione dinamica, ossia a fronte di un problema, l'algoritmo lo spezza in più sottoproblemi che però sono dipendenti l'uno dall'altro: per risolverli si sfruttano le dipendenze presenti tra essi e si memorizzano i risultati parziali in una matrice di valori.
Per risolvere il problema devo determinare questo insieme di variabili $U_{1n} = \{A \in V : A \Rightarrow_G^\star x\}$. Definiamo $x = a_1 a_2 \cdots a_n$ dove ogni $a_i \in \Sigma$. L'algoritmo calcola tutti gli insiemi $U_{ij} = \{A \in V : A \Rightarrow_G^\star a_i a_{i+1} \dots aj\}$, con $1 \le i \le j \le n$.
Vediamo per esempio come calcolare U_{1n}. E' chiaro che $A \in U_{1n}$ se e solo se esiste $A \to BC$ in P ed esiste $K \in \{1, \dots, n-1\}$ tale che $B \in U_{1k}$ e $C \in U_{k+1n.}$ Infatti la derivazione $A \Rightarrow_G^\star a_i a_{i+1} \dots aj$ è rappresentata da un albero della forma



Per costruire U_{1n}, una volta noti gli insiemi $U_{11}, U_{2n}, U_{12}, U_{3n}, \ldots, U_{1n-1}, U_{nn}$, posso eseguire il seguente ciclo di istruzioni:

$T = \{\emptyset\}$
for k = 1, ..., n-1 do
    for $A \to BC \in P$ do
        if $B \in U_{1k}$ AND $C \in U_{k+1n}$ then
                $T = T \cup \{A\}$ (se A e' gia' presente non lo aggiungo)
return T

Osserviamo inoltre che gli insiemi $U_{11}, U_{22}, U_{33}, \ldots, U_{nn}$ sono formati solo da variabili che derivano un simbolo terminale e possono essere facilmente calcolati dalla grammatica: per ogni i, j *con* $i < j, U_{ij}$ può essere calcolato da U_{ik} e U_{k+1j} *con* $k = i, \ldots, j-1$ usando un procedimento simile a quello illustrato sopra per calcolare U_{1n}.



L'algoritmo è allora descritto dalla seguente procedura.

begin
for i = 1, ..., n do $U_{ii} = \{A \in V : A \to a_i \in P\}$
for d = 1, ..., n-1 do (creo un ciclo sulla distanza crescente tra i e j)
for i = 1, ..., n-d do
j = i + d
$U_{ij} = \emptyset$
for k = i, i+1, ..., j-1 do
for $A \to BC \in P$ do
if $B \in U_{ik}$ AND $C \in U_{k+1j}$ then
$U_{ij} = U_{ij} \cup \{A\}$
if $S \in U_{1n}$ then return 1
else return 0
end

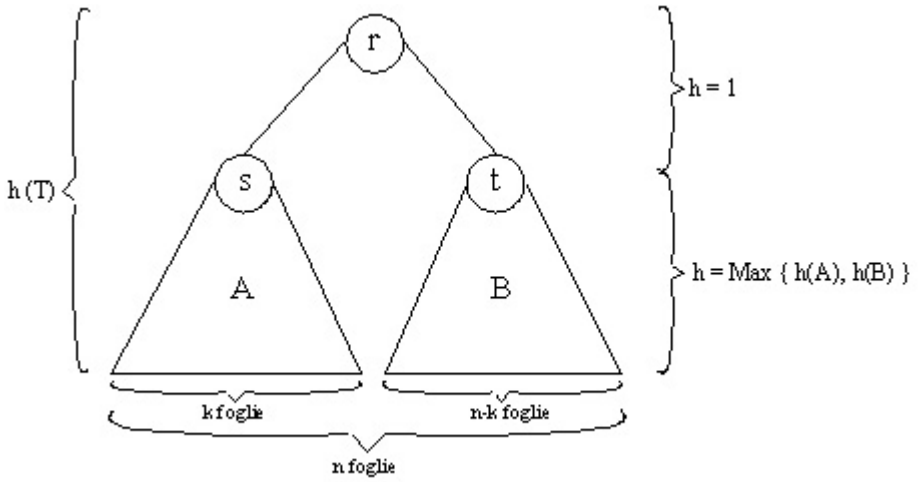
Proposizione 4.6 *Per ogni linguaggio libero da contesto L, l'algoritmo CYK su un input di lunghezza n richiede un tempo di calcolo che è dell'ordine di grandezza di n^3 e uno spazio di memoria dell'ordine di n^2.*

4.4 Lemma d'iterazione.

Prima di enunciarlo nella nuova forma, valida per le grammatiche libere da contesto, introduciamo una proprietà delle altezze degli alberi binari che ci tornerà utile per farne la dimostrazione.

Lemma 4.7 *sull'altezza degli alberi binari. In ogni albero binario T con k foglie, l'altezza dell'albero $h(T) \geq \lceil log_2 k \rceil$*

Dimostrazione. Dimostriamo il lemma per induzione. È facilmente verificato nel caso $k_0 = 1$, caso in cui l'albero è composto da un solo nodo. Supposto vero l'enunciato per $k = 1, 2, \ldots, n-1$ dimostriamolo per $k = n$.



Sia T un albero binario con n foglie e supponiamo che la radice abbia due figli (fig.28). Indipendentemente dal fatto che T sia o meno bilanciato, la sua altezza $h(T)$ è data dalla più grande delle altezze calcolata fra i suoi due sottoalberi A, B incrementata di uno, cioè:

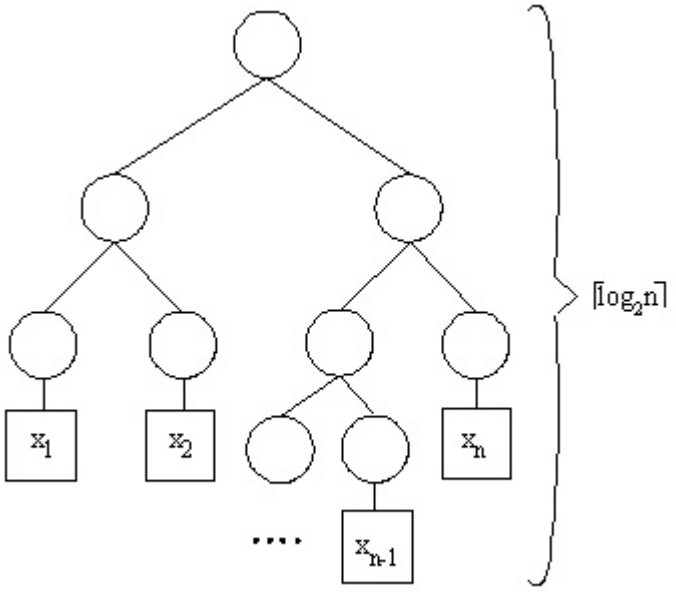
$$h(T) = 1 + Max\{h(A), h(B)\}$$

Per ipotesi induttiva sappiamo che $h(A) \geq \lceil \log_2 K \rceil$ e che $h(B) \geq \lceil \log_2 (n-k) \rceil$. Abbiamo poi n nodi da dividere tra i due sottoalberi A e B; questo significa che o A o B avrà almeno $\frac{n}{2}$ nodi e quindi sostituendo nella precedente formula:

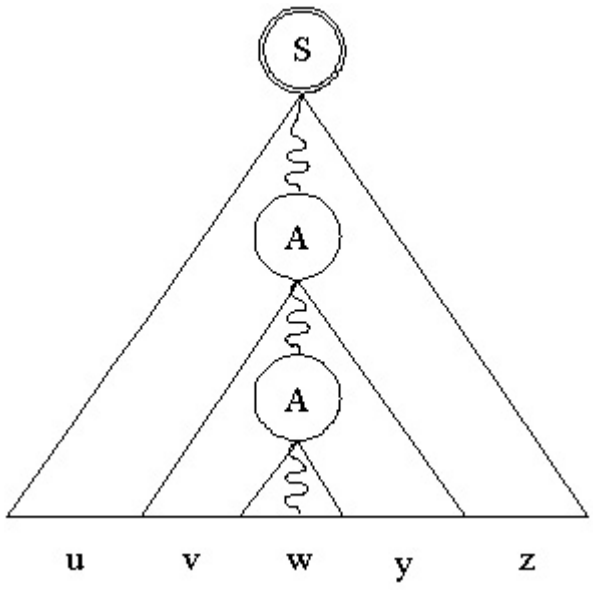
$$h(T) = 1 + Max\{h(A), h(B)\} \geq 1 + \lceil \log_2 \frac{n}{2} \rceil \quad ovvero \quad h(T) \geq \lceil \log_2 n \rceil$$

Abbiamo così dimostrato il lemma sulle altezze degli alberi binari.

Consideriamo ora una grammatica $G =< V, \Sigma, S, P >$ libera da contesto scritta in forma normale di Chomsky e definiamo il linguaggio generato $L = L(G)$ con $L \subseteq \Sigma^\star$. L'albero di derivazione D della parola $x = x_1 x_2 \cdots x_{n-1} x_n$ generata dalla grammatica G in CNF, è un albero binario e quindi verifica la proprietà enunciata nel lemma precedente, ovvero $h(D) \geq \lceil \log_2 n \rceil$.



A questo punto notiamo che se la parola è sufficientemente lunga da avere $\lceil \log_2 |x| \rceil \geq \#V$ (numero di variabili), per il lemma precedente, avremo sicuramente un cammino che unisce una foglia con la radice dell'albero passante per due nodi etichettati con la stessa variabile, $A \in V$. I due nodi sono radici di sotto-alberi uno incluso nell'altro, e quindi la parola x può essere separata in cinque fattori u, v, w, y, z con $x = uvwyz$, definiti dalla seguente figura:



Quindi se nell'albero di derivazione sostituiamo al sotto-albero che genera w, il sotto-albero che genera vwy, produciamo la parola uv^2wy^2z. Se eseguo n successive sostituzioni come la precedente, produco n identiche iterazioni dei fattori v, y all'interno della parola x, nel pieno rispetto delle regole di produzione della grammatica G libera da contesto. Formalizziamo ora quanto detto nel seguente lemma.

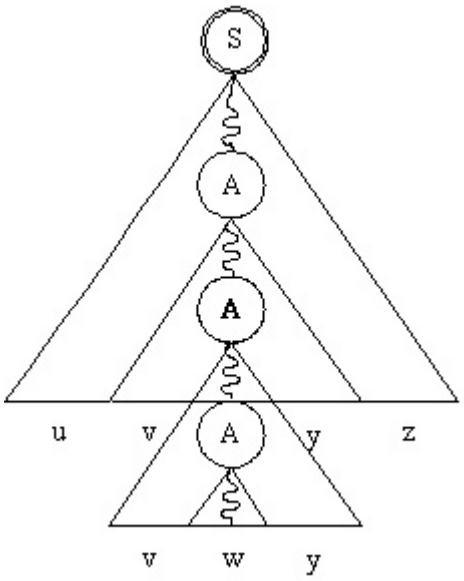
Lemma 4.8 *(lemma di iterazione per i linguaggi liberi da contesto).*

Data una grammatica $G < V, \Sigma, S, P >$ libera da contesto e definito il linguaggio $L = L(G)$, esiste un $N > 0$ intero, tale che per ogni $x \in L$ con $|x| > N$, esistono cinque parole $u, v, w, y, z \in \Sigma^\star$ dotate delle seguenti proprietà:

- $x = u\ v\ w\ y\ z$,
- $|\ v\ w\ y\ | \leq N$,
- $v\ y \neq \varepsilon$,
- *per ogni* $n \in N \quad uv^n wy^n z \in L$.

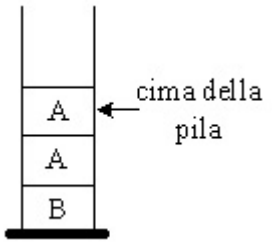
Dimostrazione. Cerchiamo allora un N che soddisfi le quattro proprietà sopra elencate.
Sia $N = 2^{\#V}$. Per il lemma sulle altezze degli alberi binari, l'albero di derivazione che genera una parola di lunghezza N, sarebbe alto $h \geq \#V$, cioè avremmo sicuramente due nodi in un cammino che unisce una foglia alla radice etichettati con la stessa variabile. Tale coppia di nodi spezza la parola x in 5 fattori u, v, w, y, z, come mostrato nella figura precedente. Partendo dal valore di N, dimostriamo le quattro proprietà sopra elencate.

1. Ovvia.
2. Consideriamo un cammino di lunghezza h dalla radice a una foglia e percorriamo il cammino in senso inverso (cioè dalla foglia alla radice). In questo percorso scegliamo i primi due nodi che hanno la stessa etichetta. Quindi l'albero che ha per radice il secondo nodo ha un'altezza pari al più al numero di variabili $\#V$. Di conseguenza le parole generate da tale sottoalbero, sono lunghe al più $2^{\#V} = N$. Poiché tale parola coincide proprio con vwy, la proparietà 2) è dimostrata.
3. Considerando che ci sono due variabili A nel cammino dalla foglia alla radice, e trovandoci in un albero binario (CNF), entrambe le variabili A, produrranno almeno un figlio, quindi le produzioni v e y non potranno essere entrambe nulle.
4. È sufficiente ripetere le produzioni come in figura, per mantenere inalterate le sottoparole u w z e far ciclare v, y dello stesso numero di volte.

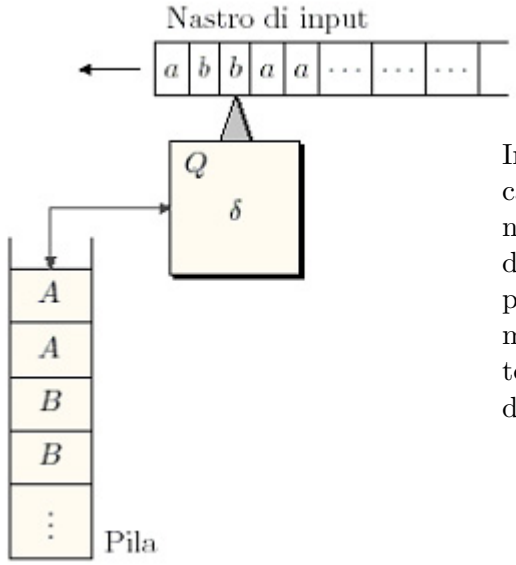


4.5 Automa a pila

Nel terzo capitolo di questa dispensa, abbiamo definito gli automi a stati finiti come particolari sistemi in grado di riconoscere i linguaggi regolari. Quello che ora definiamo, è un particolare automa, detto automa a pila, capace di riconoscere i linguaggi liberi da contesto. Prima di descrivere formalmente il nuovo automa, ricordiamo cos'è una pila.



Una pila è una particolare struttura dati, definita da una lista di record e dalle operazioni di lettura, inserimento e cancellazione Top, Push e Pop. Queste operazioni vengono eseguite solo ad una estremità della lista di record che per comodità viene chiamata cima della pila. Intuitivamente una pila rappresenta una memoria potenzialmente illimitata con vincoli particolari che regolano l'accesso alle informazioni, l'inserimento e la cancellazione dei dati. Per esempio, volendo accedere ad un record B che non si trova sulla cima della pila, devo prima eliminare ad uno ad uno, tutti i record collocati sopra B.



Intuitivamente un automa a pila è un modello di calcolo costituito da un insieme finito di stati, un nastro di ingresso suddiviso in celle, una testina di lettura del nastro e una pila. Su un dato input, la macchina può eseguire una sequenza di mosse ciascuna delle quali dipende dal simbolo letto dalla testina di lettura, dallo stato corrente e dal simbolo che si trova in cima alla pila.

Le mosse che l'automa può fare sono due: mosse tradizionali ed ε-mosse. In una mossa tradizionale, l'automa entra in un nuovo stato, sposta la testina di lettura di una posizione a destra e modifica la cima della pila sostituendo il simbolo corrente con una stringa di simboli (eventualmente vuota). Nella ε-mossa, l'automa non muove la testina di lettura ma può solo modificare la cima della pila e lo stato corrente. Come per l'automa a stati finiti, anche per l'automa a pila possiamo definire una versione deterministica e una non deterministica.

Definizione 4.9 *Un automa a pila su un alfabeto Σ è una sestupla $M =< Q, q_0, \Gamma, Z, \delta, F >$ dove:*

- *Q è un insieme finito di stati,*
- *$q_0 \in Q$ è lo stato iniziale,*

- $F \subseteq Q$ *è l'insieme degli stati finali,*
- Γ *è l'alfabeto dalla pila,*
- $Z \in \Gamma$ *è un simbolo particolare detto simbolo iniziale della pila,*
- δ *è la funzione di transizione che rappresenta le mosse della macchina. Il risultato di una mossa è rappresentato da uno stato e da una stringa di simboli che viene inserita in cima alla pila al posto del simbolo corrente.*

  $$\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \to PF(Q \times \Gamma^\star)$$

  con $PF(Q \times \Gamma^\star)$ *indichiamo l'insieme delle parti finite di* $(Q \times \Gamma^\star)$*. Osserviamo infatti che* $\Gamma^\star$ *contiene infinite stringhe.*

Per ogni $q \in Q$, per ogni $A \in \Gamma$ e ogni $a \in \Sigma$, l'insieme $\delta(q, a, A)$ definisce l'insieme delle possibili mosse tradizionali della macchina quando questa si trova nello stato q, legge a sul nastro di ingresso e il simbolo A si trova in cima alla pila. Se

$$\delta(q, a, A) = \{(q_1, \beta_1), (q_2, \beta_2), \ldots, (q_k, \beta_k)\}$$

per ogni $i = 1, \ldots, k$, l'automa può entrare nello stato q_i e sostituire A con la stringa β_i in cima alla pila, muovendo quindi la testina di lettura di una posizione a destra. Analogamente, per ogni $q \in Q$, e ogni $A \in \Gamma, \delta(q, \varepsilon, A)$ definisce l'insieme delle ε-mosse che la macchina può compiere trovandosi nello stato q e leggendo il simbolo A in cima alla pila. Tali mosse non dipendono dal simbolo letto sul nastro d'ingresso. Se $\delta(q, \varepsilon, A) = \{(P_1, \gamma_1), (P_2, \gamma_2), \ldots, (P_k, \gamma_k)\}$, allora per ogni $i = 1, \ldots, k$, la macchina può entrare nello stato p_i e sostituire il simbolo A con la stinga β_i in cima alla pila.
Esistono due criteri di riconoscimento per gli automi a pila: per stato finale o per svuotamento della pila. Nel primo caso l'automa accetta una parola di input quando al termine della sua lettura la macchina entra in uno tato finale. Nel secondo caso quando, dopo aver letto la parola, l'automa svuota la pila. Per definire questi criteri, dobbiamo formalmente introdurre la nozione di configurazione.

Definizione 4.10 *Configurazione (di una macchina M): è una stringa della forma* $q\beta$*, dove* $q \in Q$ *rappresenta lo stato corrente e* $\beta \in \Gamma^\star$ *rappresenta la parola che si trova sulla pila. Ogni mossa eseguita dalla macchina, può essere rappresentata dalle due configurazioni della macchina prima e dopo l'esecuzione della mossa. La configurazione iniziale è rappresentata da* $q_0 Z$ *.*

Definiamo ora le relazioni di transizione tra configurazioni.

Definizione 4.11 *Relazione di transizione in un passo.*

$$\forall a \in \Sigma, \quad \forall q, p \in Q, \quad \forall B \in \Gamma, \quad \forall \gamma, \beta \in \Gamma^\star \begin{cases} qB\beta \vdash_M^a p\gamma\beta & se \quad (p, \gamma) \in \delta(q, a, B) \\ qB\beta \vdash_M^\varepsilon p\gamma\beta & se \quad (p, \gamma) \in \delta(q, \varepsilon, B) \end{cases}$$

Definizione 4.12 *Relazione di transizione in più passi.*

$$\forall x \in \Sigma^\star, \quad \forall q, p \in Q, \quad \forall \gamma, \beta \in \Gamma^\star \quad qB\beta \vdash_M^{x\,\star} p\gamma\beta \text{ se esistono}$$

$a_1, a_2, \ldots, a_n \in \Sigma \cup \{\varepsilon\}$, *una catena di stati* $P_1, P_2, \ldots, P_n \in Q$ *e* $\alpha_1, \alpha_2, \ldots, \alpha_n \in \Gamma^\star$ *tali che*

$x = a_1 a_2 \cdots a_n$ *e* $p_i\alpha_i \vdash_M^{a_i} p_{i+1}\alpha_{i+1}$ *per ogni* $i = 1, \ldots, n-1$ *con* $q = p_1, p = p_n, \beta = \alpha_1, \gamma = \alpha_n$

Siamo in grado di definire allora il linguaggio riconosciuto dall'automa a pila in due modi:

1. $L(M) = \{x \in \Sigma^\star / \exists p \in F, \exists \gamma \in \Gamma^\star : q_0 Z \vdash_M^x {}^\star p\gamma\}$
   $L(M)$ *rappresenta il linguaggio accettato o riconosciuto da* M *mediante stato finale;*

2. $N(M) = \{x \in \Sigma^\star / \exists p \in Q : q_0 Z \vdash_M^x {}^\star p\varepsilon\}$
   $N(M)$ *rappresenta il linguaggio accettato da* M *mediante svuotamento della pila.*

Esempio. Sia dato il linguaggio $L = \{x \in \{a,b,c\} : x = wcw^R$ dove $w \in \{a,b\}^\star\}$.
Qui denotiamo con w^R la parola inversa della stringa w; per esempio se $w = abbab$ allora $w^R = babba$.
Definiamo l'automa M ponendo $Q = \{q_1, q_2\}, \Gamma = \{A, B, Z\}$ e assumiamo che q_1 e Z siano gli elementi iniziali dell'automa. Definiamo:

$$\begin{array}{lll}
\delta(q_1,a,Z) = \{(q_1,AZ)\}, & \delta(q_1,b,Z) = \{(q_1,BZ)\}, & \delta(q_1,c,Z) = \{(q_1,\varepsilon)\},\\
\delta(q_1,a,A) = \{(q_1,AA)\}, & \delta(q_1,a,B) = \{(q_1,AB)\}, & \delta(q_1,b,A) = \{(q_1,BA)\},\\
\delta(q_1,b,B) = \{(q_1,BA)\}, & \delta(q_1,c,A) = \{(q_2,A)\}, & \delta(q_1,c,B) = \{(q_2,B)\},\\
\delta(q_2,a,A) = \{(q_2,\varepsilon)\}, & \delta(q_2,b,B) = \{(q_2,\varepsilon)\}, & \delta(q_2,\varepsilon,Z) = \{(q_2,\varepsilon)\}.
\end{array}$$

Ogni altro insieme $\delta(q, \Sigma, X)$ è vuoto.
Consideriamo l'input $x = babacabab$ e analizziamo le transizioni dell'automa:

$$q_1 Z \vdash_A^b \quad q_1 BZ \vdash_A^a \quad q_1 ABZ \vdash_A^b \quad q_1 BABZ \vdash_A^a \quad q_1 ABABZ \vdash_A^c \quad q_2 ABABZ \vdash_A^a$$
$$q_2 BABZ \vdash_A^b \quad q_2 ABZ \vdash_A^a \quad q_2 BZ \vdash_A^b \quad q_2 Z \vdash_A^\varepsilon \quad q_2$$

Di conseguenza x è accettato dalla macchina per pila vuota. Si può verificare che $L = L(M)$.

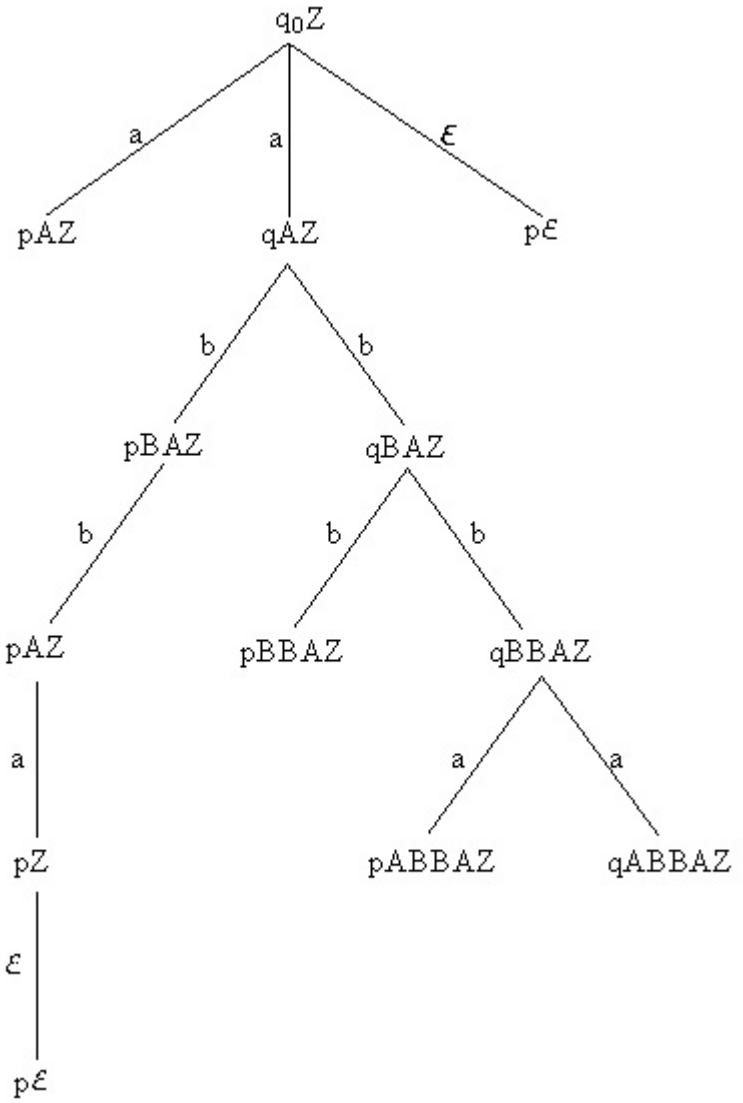
Esempio. Sia dato il linguaggio $L = \{x \in \{a,b\} : x = ww^R$ dove $w \in \{a,b\}^\star\}$

$$M = < \{q,p\}, q, \{A,B,Z\}, Z, \delta, \emptyset >$$

Transizioni dallo stato iniziale

$$\overbrace{\delta(q,a,Z) = \{(q,AZ),(p,AZ)\} \quad \delta(q,b,Z) = \{(q,BZ),(p,BZ)\} \quad \delta(q,\varepsilon,Z) = \{(p,\varepsilon)\}}$$
$$\delta(q,a,A) = \{(q,AA),(p,AA)\} \quad \delta(q,a,B) = \{(q,AB),(p,AB)\} \quad \delta(q,b,A) = \{(q,BA),(p,BA)\}$$
$$\delta(q,b,B) = \{(q,BB),(p,BB)\} \quad \delta(q,a,A) = \{(p,\varepsilon)\} \quad \delta(q,b,B) = \{(p,\varepsilon)\} \quad \delta(q,a,Z) = \{(p,\varepsilon)\}$$

Mostriamo come funziona la macchina, su una stringa particolare $x = abba$. Rappresentiamo il calcolo tramite l'albero di computazione.



Osservazione. Negli esempi appena visti, si nota che l'automa che riconosce il linguaggio $L = \{x \in \{a,b,c\} : x = wcw^R \text{ dove } w \in \{a,b\}^\star\}$ si comporta in modo deterministico, in quanto la macchina inizia a svuotare la pila quando legge il simbolo c, viceversa l'automa che riconosce il linguaggio $L = \{x \in \{a,b\} : x = ww^R \text{ dove } w \in \{a,b\}^\star\}$ non si comporta in modo deterministico proprio perché non è in grado di decidere quando deve iniziare a svuotare la pila.

4.5.1 Automi a pila deterministici

L'automa a pila $M =< Q, q_0, \Gamma, Z, \delta, F >$ è deterministico se:

1. $\forall a \in \Sigma, \forall q \in Q, \forall B \in \Gamma$
   $\delta(q, \varepsilon, B) \neq \emptyset \quad$ implica $\quad \delta(q, a, B) = \emptyset$,
2. $\forall b \in \Sigma \cup \{\varepsilon\}, \forall q \in Q, \forall A \in \Gamma$
   $\delta(q, a, B)$ contiene al più un elemento di $Q \times \Gamma^\star$.

Esempio. Sia dato il linguaggio

$D = \{x \in \{(,),\$\}^\star : x = y\$,\ \text{con } y \in \{(,)\}^\star \text{ che è una stringa di parentesi correttamente innestate}\}$.

Definiamo le transizioni:

- $\delta(q, \$, Z) = \{(p, Z)\}$,
- $\delta(q, (, Z) = \{(q, A, Z)\}$,
- $\delta(q, (, A) = \{(q, AA)\}$,
- $\delta(q,), A) = \{(q, \varepsilon)\}$.

E' facile verificare che il linguaggio D è riconosciuto da un automa a pila deterministico mediante stato finale.

Proposizione 4.13 *La classe dei linguaggi*

$$D = \{L \subseteq \Sigma^\star : L = L(M),\ M \text{ automa a pila deterministico}\}$$

è propriamente contenuta nella classe

$$N = \{L \subseteq \Sigma^\star : L = L(M),\ M \text{ automa a pila non deterministico}\}$$

ovvero D è strettamente contenuto N.

Proposizione 4.14 *Se un linguaggio $L \subseteq \Sigma^\star$ verifica la condizione $L = L(M)$, allora esiste un automa a pila V tale che $L = N(V)$. Viceversa, se $L = N(M)$ per qualche automa a pila M, allora esiste un altro automa a pila V tale che $L = L(V)$.*

Proposizione 4.15 *Un linguaggio $L \subseteq \Sigma^\star$ è libero da contesto se e solo se esiste un automa a pila M (non deterministico) tale che $L = N(V)$.*

La dimostrazione di questo teorema è basata sulla forma normale di Greibach.