LINGUAGGI FORMALI E AUTOMI

(DISPENSE)

ALBERTO BERTONI, BEATRICE PALANO

Capitolo 1: Linguaggi e Grammatiche

1. Monoide delle parole, Linguaggi e operazioni tra linguaggi

In generale, con *linguaggio* si intende la capacità d'uso e l'uso stesso di un qualunque sistema di simboli adatti a comunicare. Questa accezione è del tutto generale: si parla di linguaggio della musica o del cinema, così come di linguaggi di programmazione.

Un esempio importante è il *linguaggio naturale*, scritto o parlato. Va osservata la profonda differenza tra il linguaggio *parlato* e quello *scritto*: nel primo caso, il messaggio è veicolato da un segnale acustico continuo, in cui è difficile rilevare la separazione tra lettere o parole, mentre nel secondo il messaggio è codificato da una sequenza di caratteri tipografici e spazi bianchi. Un "testo" nel linguaggio scritto può quindi essere visto come una sequenza finita di simboli scelti da un insieme finito prefissato di simboli come {a ,.., z , A , .. , Z , . , ; , : , !, …, - (="spazio")}. Naturalmente non ogni sequenza di simboli è riconosciuta essere un "testo" del linguaggio: tutti riconoscono che la sequenza "mi-illumino-di-immenso" è una frase dell'italiano scritto, mentre "miil-lumi-nodiimmen-so" non lo è. Un importante e difficile obbiettivo della linguistica è quello di dare una rigorosa descrizione di un linguaggio scritto, specificando quali frasi sono formate correttamente.

Di particolare interesse sono i *linguaggi artificiali*, importanti strumenti per la comunicazione uomo-macchina. Ad esempio, il linguaggio C è descritto dalle sequenze di caratteri che un compilatore C accetta per tale linguaggio. Facendo astrazione dagli esempi precedenti, possiamo ora introdurre la nozione di *linguaggio formale*.

Si considera per prima cosa un arbitrario insieme finito $\Sigma = \{a_1, a_2 \dots a_n\}$, detto *alfabeto*, i cui elementi sono detti *simboli*.

Una *parola* (o *stringa*) su un alfabeto Σ è una sequenza finita di simboli appartenenti a Σ; ad esempio, *aababbab* e *bbababb* sono due parole sull'alfabeto $\{a,b\}$. E' conveniente considerare la parola non contenente alcun simbolo: essa sarà detta *parola vuota* e indicata con il simbolo ε.

Data una parola w, la *lunghezza* di w (denotata con l(w) oppure |w|) è il numero di simboli che compongono w. La lunghezza della parola vuota è 0.

Dato un alfabeto Σ, l'insieme di tutte le parole che si possono ottenere da Σ viene indicato come Σ^*, mentre l'insieme di tutte le parole che si possono ottenere da Σ tranne la parola vuota viene indicato come Σ^+ .

Date due parole $v = x_1 \dots x_n$ e $w = y_1 \dots y_m$, si dice *prodotto di giustapposizione* di v e w (e si indica come $v \cdot w$) la parola $z = x_1 \dots x_n y_1 \dots y_m$. Si osservi che $l(x \cdot y) = l(x) + l(y)$

Il prodotto di giustapposizione è una operazione binaria su Σ^* , che gode della *proprietà associativa* e in cui ε è l'elemento neutro:

1. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
2. $x \cdot \varepsilon = \varepsilon \cdot x = x$

Grazie a queste due proprietà l'insieme Σ^* con l'operazione di giustapposizione e l'elemento neutro ε forma un *monoide.* Più precisamente, la terna $(\Sigma^*, \cdot, \varepsilon)$ è detta *monoide libero* generato da Σ.

Date le parole x, y diremo che x è *prefisso* di y se $y=x\cdot z$ per qualche z, x è *suffisso* di y se y=zx per qualche z, x è *fattore* di y se $y=z\cdot x\cdot w$ per qualche z e w.

Un *linguaggio* L è un qualunque sottoinsieme (finito o infinito) di Σ^*. Il linguaggio non contenente alcuna parola viene detto *linguaggio vuoto* e indicato con Ø. Si osservi che Ø è diverso dal linguaggio $\{ \varepsilon \}$ contenente solo la parola vuota.

I linguaggi sono sottoinsiemi di Σ^*, quindi si possono applicare ad essi le usuali operazioni booleane:

1. *Unione*: dati linguaggi L_1 e L_2 , il linguaggio $L_1 \cup L_2$ contiene tutte le parole che appartengono s a L_1 oppure a L_2.
2. *Intersezione*: dati linguaggi L_1 e L_2 , il linguaggio $L_1 \cap L_2$contiene tutte le parole che appartengono sia a L_1 che a L_2.
3. *Complemento*: il linguaggio L^c contiene tutte le parole che non stanno in L.

Si osservi che l'unione e intersezione di linguaggi finiti è ancora un linguaggio finito, mentre il complemento di un linguaggio finito è un linguaggio infinito.

Abbiamo introdotto in Σ^* il prodotto di giustapposizione. Due naturali operazioni indotte sui linguaggi grazie alla giustapposizione sono il *prodotto* e la *chiusura di Kleene*:

1. *Prodotto*: dati linguaggi L_1 e L_2 , il loro prodotto è il linguaggio $L_1 \cdot L_2 = \{xy | x \in L_1 \text{ e } y \in L_2$. Poiché il prodotto è associativo, possiamo definire la potenza L^k, dove $L^0 = \{ \varepsilon \}$ $L^{k+1} = L^k \cdot L$.
2. *Chiusura di Kleene*: dato un linguaggio L, la sua chiusura è il linguaggio

$$L^* = L^0 \cup L^1 \cup \ldots \cup L^k \cup \ldots = \bigcup_{k=0}^{\infty} L^k$$

dove $L^0 = \varepsilon$ e L^n è$= L\cdot L^{n-1}$, cioè il prodotto di L per sé stesso n volte.

Poniamo ulteriormente $L^+ = \bigcup_{k=1}^{\infty} L^k$

Si osservi che L* è un sottomonoide di Σ^*, ed in particolare è il più piccolo dei monoidi che contengono L.

Un linguaggio L è un *codice* se ogni parola in L^+ è univocamente decomponibile come prodotto di parole di L.
Ad esempio, L={a, aba, abaab} non è un codice, poichè la parola $abaaba \in L^+$ e può essere scomposta in due modi diversi come prodotto di parole in L: abaaba = aba·aba = abaab·a. E' invece facile osservare che L = ={a, ab} è un codice.

2. Rappresentazione di Linguaggi: generatori e riconoscitori.

Un linguaggio L è un insieme di parole su un dato alfabeto. Come è possibile dare una descrizione di L? Due soluzioni sono:

1. Se L è finito, può essere rappresentato estensivamente elencando tutte le sue parole: $L = \{w_1, w_2 \ldots w_n\}$.
2. Se L è infinito, potremo rappresentarlo solo intensivamente, attraverso cioè una proprietà P(w), descrivibile con una quantità finita di informazione, che risulta vera solo su tutte le parole di L: $L=\{w \mid P(w)=1\}$.

Due importanti metodi per rappresentare linguaggi infiniti sono quello *generativo* e quello *riconoscitivo*.

Dal punto di vista *riconoscitivo*, un linguaggio L è descritto da un algoritmo A che, avendo in ingresso una parola w, dà in uscita 1 se $w \in L$, 0 se $w \notin L$: l'algoritmo A, detto riconoscitore, calcola dunque la funzione caratteristica del linguaggio L, cioè la funzione $\chi_L(w)$ = se $w \in L$ allora 1 altrimenti 0.

Non tutti il linguaggi ammettono un riconoscitore: linguaggi che ammettono riconoscitori sono detti *ricorsivi* o *decidibili*. In questo corso introdurremo riconoscitori come gli *automi a stati finiti* e gli *automi a pila*, per sottoclassi di linguaggi ricorsivi. Osserviamo che la funzione caratteristica del complemento di L è esattamente $1- \chi_L(w)$; se $\chi_L(w)$ è calcolabile mediante un algoritmo, lo è anche $1- \chi_L(w)$, e questo implica che il complemento di un linguaggio ricorsivo è ricorsivo.

Dal punto di vista *generativo*, un linguaggio L è descritto mediante una procedura che genera in modo sistematico tutte le parole del linguaggio. Questa viene usualmente data attraverso un *sistema generativo* (o *sistema formale* o *calcolo logico*), descritto da una relazione V(w,d), con w variabile su Σ^* e d variabile su U^*, dove U è un opportuno alfabeto. Tale relazione deve verificare le seguenti condizioni:

1. Se V(w,d) = 1 allora $w \in L$ (correttezza) e, se $w \in L$, allora esiste $d \in U^*$ per cui V(w,d) = 1 (completezza); se V(w,d) = 1 si suole dire che d è una dimostrazione (o testimone) di w.
2. V è calcolabile da un algoritmo A, esiste cioè un algoritmo A che, avendo in ingresso le parole w e d, dà in uscita 1 se V(w,d) = 1, 0 se V(w,d) = 0.

Osserviamo che $L=\{w \mid \exists d\ V(w,d)=1\}$.

Non tutti il linguaggi ammettono un sistema generativo: linguaggi che ammettono sistemi generativi sono detti *ricorsivamente numerabili* o *semidecidibili*. In questo corso introdurremo sistemi generativi come le *grammatiche*, capaci di generare tutti i linguaggi ricorsivamente numerabili.

Chiaramente, se un linguaggio L ammette un riconoscitore, allora ammette anche un sistema generativo. Questo implica che ogni linguaggio ricorsivo è anche ricorsivamente numerabile.

Se invece L ammette un sistema generativo, in generale è solo possibile costruire una procedura (cioè un algoritmo A che può non terminare) , che avendo in ingresso una parola w, dà in uscita 1 se $w \in L$,

non termina oppure dà in uscita 0 se $w \notin L$. A tal riguardo, si consideri una naturale codifica di interi con parole di U*: diremo u_n la parola che codifica l'intero n.
Se V(w,d) è la relazione stabilita dal calcolo logico per L, la procedura A è la seguente:

Procedura A(w: parola di Σ*)
n=0
while $V(w,u_n)=0$ do n=n+1
return(1)

Se infatti $w \in L$, esiste un n per cui $V(w,u_n)=1$; la procedura A di conseguenza termina e restituisce 1.
Se invece $w \notin L$, per ogni n vale che $V(w,u_n)=0$; in questo caso la procedura A non termina.

Esistono linguaggi L ricorsivamente numerabili il cui complemento non è ricorsivamente numerabile. Tali linguaggi non sono ricorsivi: se infatti L fosse ricorsivo, anche il suo complemento L^c sarebbe ricorsivo e quindi ricorsivamente numerabile. Possiamo concludere che, dato un sistema generativo per L, non necessariamente esiste un algoritmo che termina sempre e che riconosce L.

3. Un esempio di sistema generativo: le grammatiche

Consideriamo il seguente insieme di regole descritte informalmente, che permettono la generazione di alcune frasi dell'italiano:

1. Una <Frase> è la giustapposizione di un <Soggetto>, di un <Predicato>, di un <Complemento>.
2. Un <Soggetto> è la giustapposizione di un <Articolo> e di un <Nome>.
3. Un <Complemento> è la giustapposizione di un <Articolo> e di un <Nome>.
4. Un <Articolo> è "il".
5. Un <Nome> è "cane" oppure "gatto" oppure "topo
6. Un <Predicato> è "mangia" oppure "teme"

Esempi di frasi generate sono "il gatto mangia il topo" oppure "il cane teme il gatto". Osserviamo che tali frasi sono parole sull'alfabeto {il, cane, gatto, topo, mangia, teme}, quindi le regole precedenti denotano un linguaggio $L \subseteq$ { il, cane, gatto, topo, mangia, teme}*, mentre i simboli <Frase>, <Soggetto>, <Complemento>, <Predicato>, <Articolo>, <Nome> sono utilizzati per generare tale linguaggio, ma non fan parte di parole del linguaggio: essi vengono quindi detti *metasimboli* o *simboli non terminali*, mentre {il, cane, gatto, topo, mangia, teme} sono detti *simboli terminali* (o semplicemente simboli).

Le regole 1,2,3,4,5,6 possono essere interpretate come regole di produzione come segue:

<Frase> → <Soggetto> <Predicato> <Complemento>
<Soggetto> → <Articolo> <Nome>
<Articolo> → il
<Nome> → cane / gatto / topo
<Predicato> → mangia / teme

Una regola di produzione è data quindi da una coppia $\alpha \rightarrow \beta$, dove α (parte sinistra) è una parola non vuota di simboli (terminali o non) e β, in modo analogo, una parola di simboli (terminali o non); si osservi che nel nostro caso la parte sinistra di una regola è sempre costituita da un solo metasimbolo.

La applicazione di una regola del tipo $\alpha \rightarrow \beta$ ad una parola $x\alpha y$ produce la parola $x\beta y$, ottenuta sostituendo il fattore α con β; l'applicazione della regola sarà denotata $x\alpha y \Rightarrow x\beta y$.

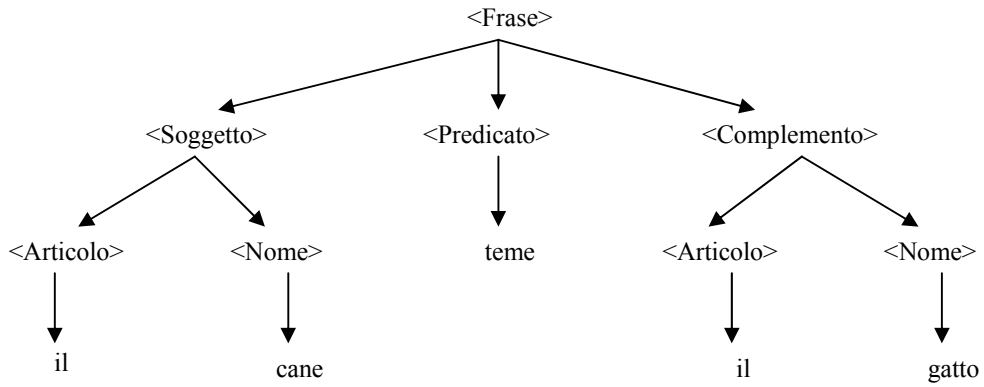
Date due parole w e z, diremo $w \Rightarrow^* z$ se z può essere ottenuto da w applicando un numero finito di regole di produzione; nel nostro caso, ad esempio, possiamo rilevare che:

<Frase>$\Rightarrow^*$ il cane teme il gatto

Infatti:
<Frase> $\Rightarrow$ <Soggetto> <Predicato> <Complemento> $\Rightarrow$
$\Rightarrow$<Articolo> <Nome> <Predicato> <Complemento>$\Rightarrow$ il <Nome> <Predicato> <Complemento>$\Rightarrow$
$\Rightarrow$il cane <Predicato> <Complemento> $\Rightarrow$il cane teme <Complemento>$\Rightarrow$
$\Rightarrow$il cane teme <Articolo> <Nome>$\Rightarrow$il cane teme il <Nome>$\Rightarrow$ il cane teme il gatto

Si noti che nella derivazione precedente le regole di produzione sono state applicate in ogni parola al primo metasimbolo da sinistra. Il linguaggio generato è formato dalle parole $w \in \{$ il, cane, gatto, topo, mangia, teme$\}^*$ tali che <Frase>$\Rightarrow^*$w. Il metasimbolo <Frase> gioca quindi il particolare ruolo di "parola scelta inizialmente" e viene chiamato *assioma*.

Una rappresentazione grafica della precedente derivazione è data dal seguente albero:



Astraendo dall'esempio precedente, siamo pronti ad introdurre in generale il concetto di grammatica G e di linguaggio L(G) generato da G.

Definizione 3.1: una grammatica G è una quadrupla $< \Sigma, Q, P, S >$ dove:
1. Σ e Q sono due alfabeti finiti disgiunti, rispettivamente di simboli terminali e metasimboli (o simboli non terminali).
2. *P* è un insieme finito di regole di produzione; una regola di produzione è una coppia $\alpha \rightarrow \beta$ di parole con $\alpha \in (\Sigma \cup Q)^+$ e $\beta \in (\Sigma \cup Q)^*$.
3. S è un elemento in Q, detto *assioma* o *simbolo di partenza*.

Fissata una grammatica G, date due parole $w, z \in (\Sigma \cup Q)^*$ diremo che z è derivabile in G da w in un passo, scrivendo $w \Rightarrow_G z$, se $w = x\alpha y$, $z = x\beta y$ e $\alpha \rightarrow \beta$ è una regola in *P*.

Una *derivazione* di z da w in G è una sequenza $w \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_m \Rightarrow z$, per cui sia $w \Rightarrow_G w_1$, ..., $w_i \Rightarrow_G w_{i+1}$ .., $w_m \Rightarrow_G z$; diremo infine che z è derivabile in G da w, scrivendo $w \Rightarrow_G^* z$, se w=z oppure se esiste una derivazione di z da w in G.

Il linguaggio L(G) generato dalla grammatica G è l'insieme di parole sull'alfabeto Σ derivabili dall'assioma S, cioè:

$$L(G) = \{w \mid w \in \Sigma^* \text{ e } S \Rightarrow_G^* w\}$$

Due grammatiche G_1 e G_2 sono dette equivalenti se generano lo stesso linguaggio, cioè se $L(G_1)=L(G_2)$

Osserviamo che una derivazione d è una parola in $(\Sigma \cup Q \cup \{\Rightarrow_G\})^*$; consideriamo ora la seguente funzione:

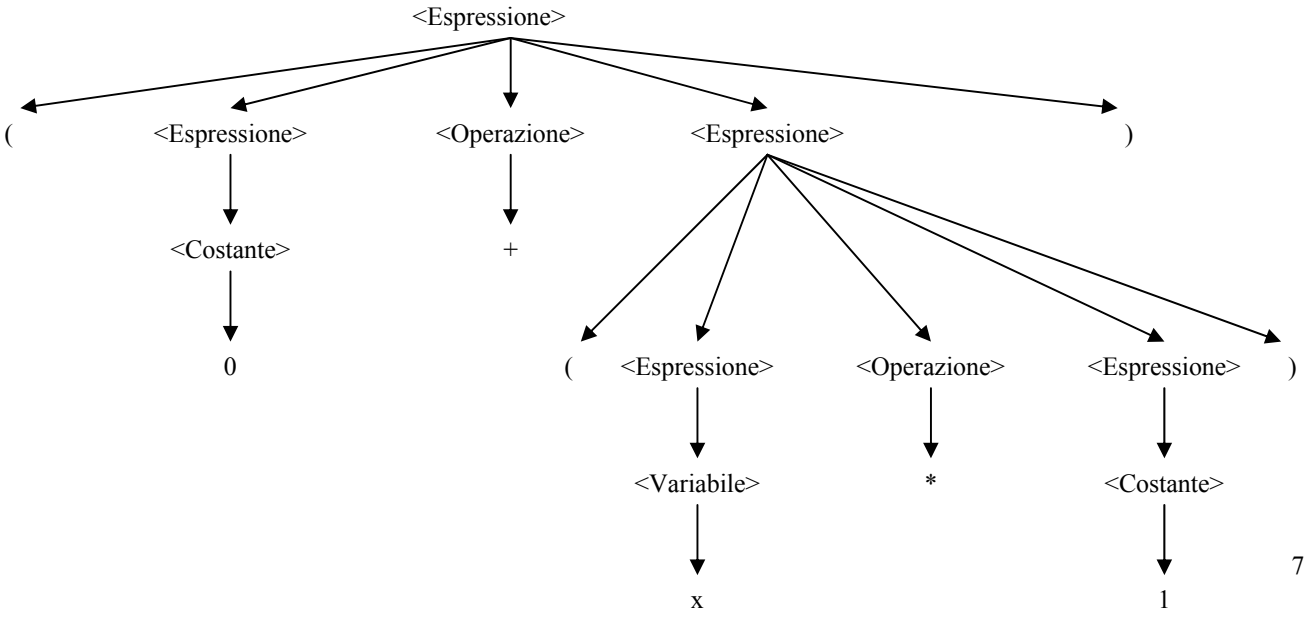
V(w,d) = se d è una derivazione di w da S in G allora 1 altrimenti 0

E' facile disegnare un algoritmo che calcola V, ed inoltre $L=\{w \mid \exists d\ V(w,d)=1\}$. Possiamo concludere che se il linguaggio L è generato da una grammatica G allora L è ricorsivamente numerabile; è possibile mostrare anche la proposizione inversa: se L è ricorsivamente numerabile, allora si può trovare una grammatica G per cui L=L(G).

Esempio 3.1: Sia L il linguaggio formato dalle espressioni parentesizzate ottenute a partire da variabili x,y, da costanti 0,1, contenenti operazioni * , + . E' chiaramente $L \subset \{x , y , 0 , 1 , * , + . (,) \}^*$; per esempio, $((x+1)*y) \in L$ mentre $x)+01) \notin L$.

L può essere generato dalla seguente grammatica:

1. Alfabeto terminale Σ={x , y , 0 , 1 , . , + . (,) }..
2. Alfabeto non terminale Q={<Espressione>,<Variebile>,<Costante>,<Operazione> }
3. Regole di produzione *P*:
   <Espressione> → <Costante> / <Variabile> / (<espressione> <operazione> <espressione>)
   <Costante> → 0 / 1
   <Variabile> → x / y
   <Operazione> → + / *
4. Assioma S = <Espressione>



Il precedente albero identifica una derivazione della parola (0+(x*1)), dimostrando che tale parola è una espressione parentesizzata "corretta".

Esempio 3.2: Sia L il linguaggio $=\{a^n b^n c^n \mid n\geq 1\}$.

L può essere generato dalla seguente grammatica:
1 Alfabeto terminale $\Sigma=\{a, b, c\}$..
2 Alfabeto non terminale $Q=\{S, B, C\}$
3 Regole di produzione *P*:
$S \rightarrow aSBC$, $S \rightarrow aBC$, $CB \rightarrow BC$, $aB \rightarrow ab$, $bB \rightarrow bb$, $bC \rightarrow bc$, $cC \rightarrow cc$
4. Assioma: S

La parola $a^n b^n c^n$ può essere derivata come segue:

$S \Rightarrow_G^* a^{n-1} S(BC)^{n-1}$	applicazione iterata della regola $S \rightarrow aSBC$
$a^{n-1} S(BC)^{n-1} \Rightarrow_G a^n (BC)^n$	applicazione iterata della regola $S \rightarrow aBC$
$a^n (BC)^n \Rightarrow_G^* a^n B^n C^n$	applicazione iterata della regola $CB \rightarrow BC$
$a^n B^n C^n \Rightarrow_G a^n bB^{n-1} C^n$	applicazione della regola $aB \rightarrow ab$
$a^n bB^{n-1} C^n \Rightarrow_G^* a^n b^n C^n$	applicazione iterata della regola $bB \rightarrow bb$
$a^n b^n C^n \Rightarrow_G a^n b^n cC^{n-1}$	applicazione della regola $bC \rightarrow bc$
$a^n b^n cC^{n-} \Rightarrow_G^* a^n b^n c^n$	applicazione iterata della regola $cC \rightarrow cc$

Le parole che non sono del tipo $a^n b^n c^n$ non possono invece essere derivate.

4. Classificazione delle grammatiche e gerarchia di Chomsky

E' possibile classificare le grammatiche in funzione del tipo di regola di produzione. Una interessante classificazione è quella proposta da Chomsky in base a considerazioni di carattere linguistico e di semplicità di trattazione matematica. Viene considerata una gerarchia decrescente di quattro tipi di grammatiche:

1. *Grammatiche di tipo 0*: le regole di produzione sono arbitrarie.

2. *Grammatiche di tipo 1*: ogni regola di produzione $\alpha \rightarrow \beta$ della grammatica deve essere tale che $l(\beta)\geq l(\alpha)$; è permessa la regola $S \rightarrow \varepsilon$, se S è l'assioma, a patto che S non compaia nella parte destra di nessuna altra regola. La grammatica presentata in Esempio 3.2 è di tipo 1.

3. *Grammatiche di tipo 2*: ogni regola di produzione $\alpha \rightarrow \beta$ della grammatica è tale che α è un metasimbolo. La grammatica presentata in Esempio 3.1 è di tipo 2. Un altro esempio è la seguente grammatica G, che genera il linguaggio di Dyck formato dalle parentesizzazioni corrette:
$$G = < \{(\ ,\ )\}, \{S\}\ , \{S \rightarrow SS,\ S \rightarrow (S)\ ,\ S \rightarrow \varepsilon\ \},\ S >$$

4. *Grammatiche di tipo 3*: ogni regola di produzione della grammatica è del tipo $A \rightarrow \sigma B$, $A \rightarrow \sigma$ o $A \rightarrow \varepsilon$, dove A, B sono arbitrari metasimboli e σ un arbitrario simbolo terminale. La seguente grammatica G è di tipo 3 e genera il linguaggio $\{a^{2n} \mid n>0\}$:
$$G = < \{a\}, \{S,A\}\ , \{S \rightarrow aA\ ,\ A \rightarrow aS\ ,\ A \rightarrow a\ \},\ S >$$

Ogni grammatica G genera un linguaggio L(G). Una classificazione delle grammatiche porta ad una naturale classificazione dei linguaggi.

Definizione 4.1: un linguaggio L è detto di *tipo k* (k=0,1,2,3) se esiste una grammatica G di tipo k tale che L = L(G). Indichiamo con R_k la classe dei linguaggi di tipo k (k=0,1,2,3).

Chiaramente le grammatiche di tipo k sono un sottoinsieme proprio delle grammatiche di tipo j, se k<j. Questo implica la seguente relazione di inclusione per le classi R_k :

$$R_3 \subseteq R_2 \subseteq R_1 \subseteq R_0$$

E' possibile mostrare che tale inclusione è propria:

Proposizione 4.1: $R_3 \subset R_2 \subset R_1 \subset R_0$
Dim.:
- $R_3 \subset R_2$: come mostreremo in seguito, il linguaggio $\{a^n b^n \mid n\geq 1\} \notin R_3$; per contro $\{a^n b^n \mid n\geq 1\} \in R_2$ in quanto generabile dalla grammatica G = < {a,b}, {S} , {S→aSb , S→ab}, S>.
- $R_2 \subset R_1$: come mostreremo in seguito, il linguaggio $\{a^n b^n b^n \mid n\geq 1\} \notin R_2$; per contro tale linguaggio è generato da una grammatica di tipo 1 (si veda Esempio 3.2)
- $R_1 \subset R_0$: sappiamo che esistono linguaggi ricorsivamente numerabili che non sono ricorsivi. Basta allora provare che ogni linguaggio $L \in R_1$ è ricorsivo.

A tal riguardo, dato L generato da una grammatica G= < Σ,Q,*P*,S> di tipo 1, fissata una parola $w \in \Sigma^*$, si consideri il grafo orientato finito GR(w) che ha come insieme di vertici le parole $z \in (\Sigma \cup Q)^*$ con $l(w) \geq l(z)$ e come archi le coppie (x,y) per cui $x \Rightarrow_G y$.
Osserviamo che $w \in L$ se e solo se c'è una derivazione $S \Rightarrow_G w_1$, ..., $w_i \Rightarrow_G w_{i+1}$ .., $w_m \Rightarrow_G w$ con $l(w) \geq l(w_m) \geq \ldots \geq l(w_1) \geq l(S)=1$, poiché se $\alpha \rightarrow \beta$ è una regola di produzione della grammatica vale $l(\beta) \geq l(\alpha)$.
Possiamo concludere che $w \in L$ se e solo se in GR(w) c'è un cammino da S a w.
Un algoritmo riconoscitore per L è allora il seguente:
Input (w)
1. Costruisci GR(w)
2. If " in GR(w) c'è un cammino da S a w" then return(1) else return(0) □

I linguaggi di tipo 0 coincidono, come abbiamo visto, coi linguaggi ricorsivamente numerabili.

I linguaggi di tipo 1 sono anche detti *dipendenti dal contesto* o *contestuali*, mentre quelli di tipo 2 sono detti anche *liberi dal contesto* o *non contestuali* . Questa terminologia deriva dal fatto che una regola $A \rightarrow \beta$, con A metasimbolo e β parola non vuota, è detta *non contestuale*, mentre la regola $xAy \rightarrow x\beta y$, con $xy \in (\Sigma \cup Q)^+$, è detta *contestuale*. La prima può essere applicata alla parola gAh, producendo la parola gβh, per qualsivoglia g,h , mentre la seconda può essere applicata alla parola gAh, producendo anche in questo caso la parola gβh, ma solo in certi contesti, e precisamente se x è suffisso di g e y è prefisso di h.
Ogni grammatica con regole di produzione contestuali è detta *contestuale* e un linguaggio generato da una grammatica contestuale è detto *dipendente dal contesto*. Poiché ogni grammatica contestuale è chiaramente di tipo 1 ed è possibile provare che per ogni grammatica G di tipo 1 esiste una grammatica G' contestuale che genera lo stesso linguaggio, i linguaggi di tipo 1 sono esattamente i linguaggi dipendenti dal contesto.

I linguaggi di tipo 3, infine, sono detti anche *linguaggi regolari*.

5. Linguaggi regolari e liberi dal contesto.

In questo corso l'interesse è dedicato ai linguaggi di tipo 2 o 3; introduciamo qui alcune nozioni e risultati specifici per queste classi.

Cominciamo col richiamare che un linguaggio è di tipo 3 se esiste una grammatica di tipo 3 che lo genera. E' tuttavia possibile che una grammatica G non di tipo 3 generi un linguaggio di tipo 3; in questo caso deve naturalmente esistere una grammatica G' di tipo 3 equivalente a G. Esistono quindi classi di grammatiche un po' più generali di quelle di tipo 3 che tuttavia generano linguaggi di tipo 3. Come esempio, consideriamo le grammatiche lineari a destra: una grammatica è detta *lineare a destra* se le sue produzioni sono del tipo $A \rightarrow xB$ o $A \rightarrow y$, con A,B metasimboli e x,y parole di simboli terminali (eventualmente ε). Chiaramente una grammatica di tipo 3 è lineare a destra, mentre in generale non è vero il viceversa. Vale:

Proposizione 5.1: se L è generato da una grammatica lineare a destra, allora è di tipo 3.
Dim.
Il metodo dimostrativo consiste nel trasformare la grammatica lineare in grammatiche equivalenti, fino ad ottenere una grammatica equivalente di tipo 3. Data una grammatica lineare G:

- per ogni regola del tipo $A \rightarrow \sigma_1 \ldots \sigma_m B$ (m>1), si elimina tale regola dopo aver introdotto nuovi metasimboli e regole $A \rightarrow \sigma_1 M_1$, $M_1 \rightarrow \sigma_2 M_2$, …, $M_{m-1} \rightarrow \sigma_m B$;
- stesso discorso per ogni regola del tipo $A \rightarrow \sigma_1 \ldots \sigma_m$ (m>1). Si ottiene in tal modo una grammatica equivalente che contiene regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \sigma$, $A \rightarrow \varepsilon$, $A \rightarrow B$. Questa grammatica non è di tipo 3 solo per la presenza di regole del tipo $A \rightarrow B$.
- Allo scopo di eliminare regole di questo tipo, si considerino tutte le derivazioni del tipo $F \Rightarrow_G^* H$, dove F e H sono metasimboli. Per ogni regola $A \rightarrow \sigma B$ si aggiungono ora le regole $F \rightarrow \sigma H$ se $F \Rightarrow_G^* A$ e $B \Rightarrow_G^* H$; infatti risulta possibile nella grammatica la seguente derivazione:
$$F \Rightarrow_G^* A \Rightarrow_G \sigma B \Rightarrow_G^* \sigma H ,$$
che equivale a riscrivere F come σH .
- Analogamente per ogni regola $A \rightarrow \sigma$ (o $A \rightarrow \varepsilon$) si aggiungono le regole $F \rightarrow \sigma$ (o $F \rightarrow \varepsilon$) se $F \Rightarrow_G^* A$.
- Possiamo a questo punto eliminare tutte le regole del tipo $A \rightarrow B$, ottenendo una grammatica equivalente di tipo 3. □

Si osservi inoltre che ogni grammatica di tipo 3 può essere trasformata in una equivalente contenente solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \varepsilon$. Basta introdurre un nuovo metasimbolo M e la regola $M \rightarrow \varepsilon$, sostituendo poi ogni regola del tipo $A \rightarrow \sigma$ con la regola $A \rightarrow \sigma M$. Vale dunque:

Proposizione 5.2: se L è di tipo 3, allora è generato da una grammatica di tipo 3 contenenti solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \varepsilon$.

E' possibile trasformare ogni grammatica di tipo 3 in una equivalente contenente solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \sigma$, eliminando le regole del tipo $A \rightarrow \varepsilon$? In generale no, perché se L è generato da una grammatica contenente solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \sigma$, allora $\varepsilon \notin L$. Vale tuttavia:

Proposizione 5.3: se L è di tipo 3 con $\varepsilon \in L$, allora $L=L' \cup \{\varepsilon\}$, dove L' è generato da una grammatica con regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \sigma$.
Dim.
Sia G la grammatica di tipo 3 che genera L; per ogni regola in G di tipo $A \rightarrow \sigma B$ aggiungi regole del tipo $A \rightarrow \sigma$ se $B \Rightarrow_G^* \varepsilon$; elimina poi tutte le regole del tipo $A \rightarrow \varepsilon$. La grammatica ottenuta genera L'.

Una proposizione analoga, che diamo senza dimostrazione, vale per linguaggi di tipo 2:

Proposizione 5.4: se L è di tipo 2 con $\varepsilon \in L$, allora $L=L' \cup \{\varepsilon\}$, dove L' è generato da una grammatica con regole del tipo $A \rightarrow x$, con $x \in (\Sigma \cup Q)^+$.

Capitolo 2: Automi a stati finiti

1. Sistemi a stati

Vogliamo introdurre un sistema che modelli un semplice meccanismo di interazione con una "scatola nera", dotata di un ingresso e di una uscita. Supponiamo che tale sistema possa ricevere in ingresso messaggi, dati in un alfabeto finito $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$, e produrre in uscita un valore in $\{0,1\}$che può essere osservato. Un *esperimento* sul sistema consiste nei due passi:

1. Viene presentata una sequenza di messaggi, descritta da una parola $w \in \Sigma^*$.
2. Al termine, viene effettuata una osservazione: se il risultato è 1 la parola viene accettata, altrimenti respinta.

Poiché possiamo accedere al sistema solo attraverso esperimenti, il *comportamento* del sistema è descritto dall'insieme di parole accettate; questo sistema è quindi visto come *riconoscitore* di linguaggi. Esso può essere modellato attribuendo al sistema un insieme di possibili stati interni Q, con le seguenti richieste:
1. Ad ogni istante il sistema si trova in un preciso stato $q \in Q$ e questo stato può essere modificato solo attraverso un messaggio $\sigma \in \Sigma$ inviato in ingresso. La legge che descrive la modifica di stato interno causata da un messaggio è data dalla *funzione di transizione* (o di *stato prossimo*) $\delta: \Sigma x Q \rightarrow Q$. Se il sistema si trova nello stato q ed arriva il messaggio σ, il sistema passa nello stato $\delta(q, \sigma)$.
2. Prima dell'arrivo di ogni messaggio, il sistema si trova in uno stato iniziale $q_o \in Q$.
3. L'osservazione sul sistema è descritta da una *funzione di uscita* $\lambda: Q \rightarrow \{0,1\}$: se il sistema è nello stato q, il risultato dell'osservazione è $\lambda(q)$. Si osservi che la funzione λ è univocamente individuata assegnando l'insieme di stati $F = \{q \mid \lambda(q) = 1\}$; tali stati sono detti *stati finali*.

Formalmente:
Definizione 1.1: Un *automa a stati* A è un sistema $A = < Q , \Sigma , \delta , q_o , F >$, dove Q è *un insieme di stati*, Σ è un alfabeto finito, $\delta: \Sigma xQ \to Q$ è la *funzione di transizione*, $q_o \in Q$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali che definisce una funzione $\lambda: Q \to \{0,1\}$, dove:

$$\lambda(q) = \text{se } q\in F \text{ allora 1 altrimenti 0}$$

Se l'insieme Q è finito, l'automa è detto *a stati finiti*. La funzione $\delta: \Sigma xQ \to Q$ può essere estesa a $\delta: \Sigma^* xQ \to Q$: se la parola $w\in\Sigma^*$ identifica la sequenza di messaggi inviati, $\delta(q, w\sigma)$ è lo stato in cui si trova il sistema, inizializzato con q, dopo aver ricevuto w. La funzione $\delta: \Sigma^* xQ \to Q$ è definita induttivamente da:

1. $\delta(q, \varepsilon)=q$
2. $\delta(q, w\sigma)= \delta(\delta(w,q), \sigma)$ per ogni $w\in\Sigma^*$.

Il linguaggio L(A) riconosciuto dall'automa A è dato da:

$$L(A) = \{w \mid w\in\Sigma^* \text{ e } \delta(q_o,w)\in F \} = \{w \mid w\in\Sigma^* \text{ e } \lambda(\delta(q_o, w))=1 \}$$

Un automa a stati finiti può essere ulteriormente rappresentato come *diagramma degli stati,* cioè come un grafo orientato in cui i vertici (indicati in circonferenze) rappresentano gli stati e i lati (etichettati con simboli di Σ) le possibili transizioni tra stati. Lo stato iniziale viene indicato con una freccia in ingresso mentre gli stati finali vengono indicati con una doppia circonferenza (vedere esempio 2.1).

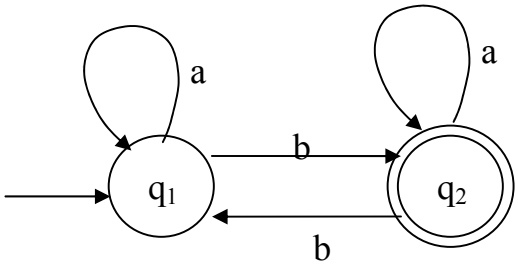
Esempio 2.1: si consideri l'automa $A= < \Sigma, Q, \delta, q_0, F>$ dove:

$\Sigma=\{a,b\}$
$Q=\{q_1, q_2\}$
$\delta: \{a,b\}x\{ q_1, q_2\} \to \{ q_1, q_2\}$ data dalla tabella :

δ	A	b
Q_1	Q_1	q_2
Q_2	Q_2	q_1

Stato iniziale: q_1
$F=\{ q_2\}$

Il suo diagramma a stati è il seguente:



Si osservi che ogni parola $w \in \Sigma^*$ induce nel diagramma degli stati un cammino, dallo stato iniziale q_o ad uno stato q, così che il linguaggio accettato dall'automa corrisponda ai cammini che portano dallo stato iniziale in uno stato finale.

Nel corso di questo capitolo studiamo gli automi come riconoscitori di linguaggi. In particolare affrontiamo problemi di sintesi (dato un linguaggio, costruire un automa che lo riconosce) e di sintesi ottima (dato un linguaggio, costruire il "più piccolo" automa che lo riconosce). Diamo poi due caratterizzazioni della classe dei linguaggi riconosciuta da automi a stati finiti.

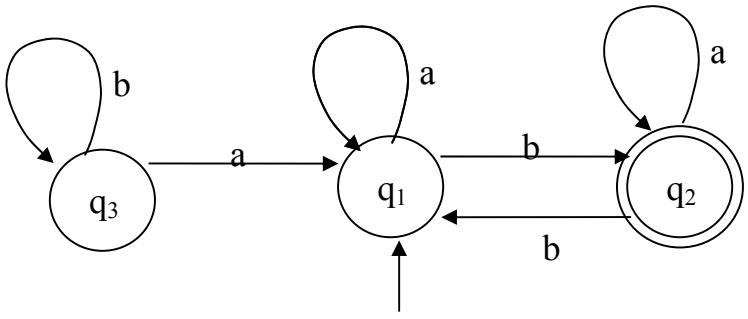
2. Osservabilità e indistinguibilità di stati.

Dato un automa a stati $A = < Q , \Sigma , \delta , q_o , F >$, il comportamento di A è il linguaggio

$$L(A) = \{w \mid w \in \Sigma^* \text{ e } \delta(q_o, w) \in F \} = \{w \mid w \in \Sigma^* \text{ e } \lambda(\delta(q_o, w))=1 \}$$

Essenzialmente il comportamento è ottenuto dai risultati degli esperimenti sull'automa, poiché $\lambda(\delta(q_o, w))$ è il risultato di una osservazione dopo che l'automa ha processato la sequenza di messaggi descritta dalla parola w. Uno stato $q \in Q$ è detto *osservabile* se esiste una parola $w \in \Sigma^*$ per cui $q = \delta(q_o, w)$; un automa A è detto *osservabile* se tutti i suoi stati sono osservabili.

Esempio 2.2: nel seguente automa lo stato q_3 è irraggiungibile o non osservabile, mentre q_1 e q_2 sono osservabili:



Gli stati non osservabili in un automa sono irrilevanti per quanto riguarda il riconoscimento e possono essere tranquillamente soppressi rendendo l'automa osservabile: per questo motivo in seguito supponiamo di trattare solo automi osservabili.

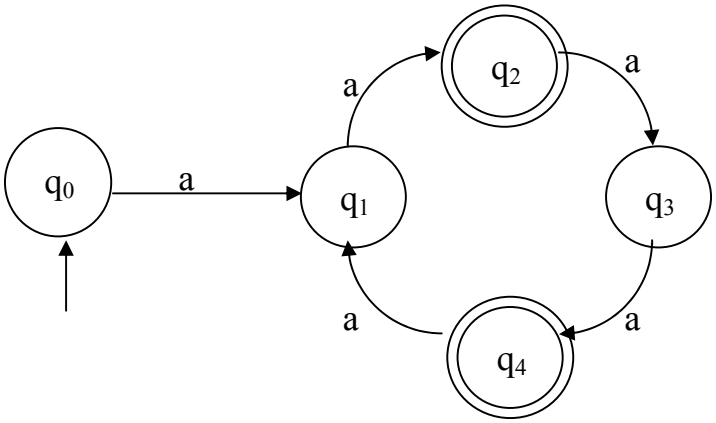
Dato un automa $A = < Q , \Sigma , \delta , q_o , F >$, due stati $q , q' \in Q$ sono *indistinguibili* se, per ogni parola $w \in \Sigma^*$ vale che $\lambda(\delta(q, w)) = \lambda(\delta(q', w))$: non è cioè possibile distinguere con esperimenti il comportamento dell'automa inizializzato con q da quello inizializzato con q'. Se q e q' sono indistinguibili, porremo $q \approx q'$, e diremo *distinguibili* due stati non indistinguibili.

Proposizione 2.1: $\approx$ è una relazione di equivalenza, verifica cioè le proprietà:
1. Riflessiva: $\forall q \ (q \approx q)$
2. Simmetrica: $\forall q,q' \ (q \approx q' \Rightarrow q' \approx q)$
3. Transitiva: $\forall q,q',q'' \ (q \approx q' \text{ e } q' \approx q'' \Rightarrow q \approx q'')$.

$\approx$ verifica inoltre la proprietà:
4. Se $q \approx q'$, allora $\delta(q, z) \approx \delta(q', z)$ per ogni $z \in \Sigma^*$.

Dim.
E' immediato dalla definizione mostrare che $\approx$ è una relazione di equivalenza. Verifichiamo quindi solo la proprietà 4. Supponiamo che $q \approx q'$. Per parole $w,x \in \Sigma^*$ vale che $\delta(\delta(x, q), w) = \delta(q, wx)$ e $\delta(\delta(x, q',w)) = \delta(q', wx)$; ne segue che per ogni parola $w \in \Sigma^*$ vale $\lambda(\delta(\delta(q,x),w)) = \lambda(\delta(q, wx)) = \lambda(\delta(q', wx)) = \lambda(\delta(\delta(q',x),w))$, poiché $q \approx q'$. Questo prova che $\delta(q,z)) \approx (\delta(q',z)$.

Esempio 2.3: si consideri il seguente automa, rappresentato dal diagramma degli stati:



In tale automa q_1 e q_3 sono indistinguibili, q_2 e q_4 sono indistinguibili mentre sono tutti gli stati q_0 , q_1 e q_2 sono distinguibili tra loro.

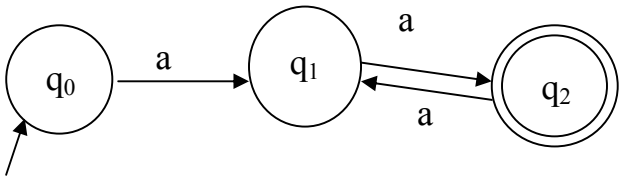
La presenza in un automa A di stati indistinguibili ci permette di costruire un nuovo automa $A_\approx$, ottenuto da A identificando gli stati indistinguibili. $A_\approx$ ha "meno" stati di A pur riconoscendo lo stesso linguaggio.

Più precisamente, osserviamo che la relazione $\approx$ partiziona l'insieme degli stati Q in classi di equivalenza; indichiamo con $[q]_\approx$ la classe di equivalenza contenente q, cioè l'insieme di stati q' con $q \approx q'$. Possiamo ora costruire un nuovo automa $A_\approx$ che ha come stati le classi di equivalenza $[q]_\approx$, come funzione di transizione $\delta([q]_\approx, \sigma) = [\delta(q, \sigma)]_\approx$, come stato iniziale la classe di equivalenza $[q_o]_\approx$, come funzione λ la funzione $\lambda([q]_\approx) = \lambda(q)$.

Si osservi che, a causa della Proposizione 2.1, tali definizioni sono ben poste e che il nuovo automa riconosce lo stesso linguaggio del precedente: $L(A)=L(A_{\approx})$.

Esempio 2.4: riprendendo **Esempio 2.3**, si ottiene l'automa $A_{\approx}$ dove:

osservabili:



3. Sintesi di automi

Affrontiamo in questa sezione il problema di sintesi di automi: dato un linguaggio $L \subseteq \Sigma^*$, costruire un automa che riconosca L. Considereremo in particolare *il più grande* automa osservabile che riconosce L e il *minimo* automa che riconosce L.

Osserviamo che un automa A associa ad ogni parola $w \in \Sigma^*$ lo stato $\delta(q_o, w)$, definendo dunque una funzione $f: \Sigma^* \to Q$ dove $f(q)= \delta(q_o, w)$. In un automa osservabile, per ogni stato q esiste una parola $w \in \Sigma^*$ tale che $q= \delta(q_o, w)$: la funzione f risulta quindi essere suriettiva.

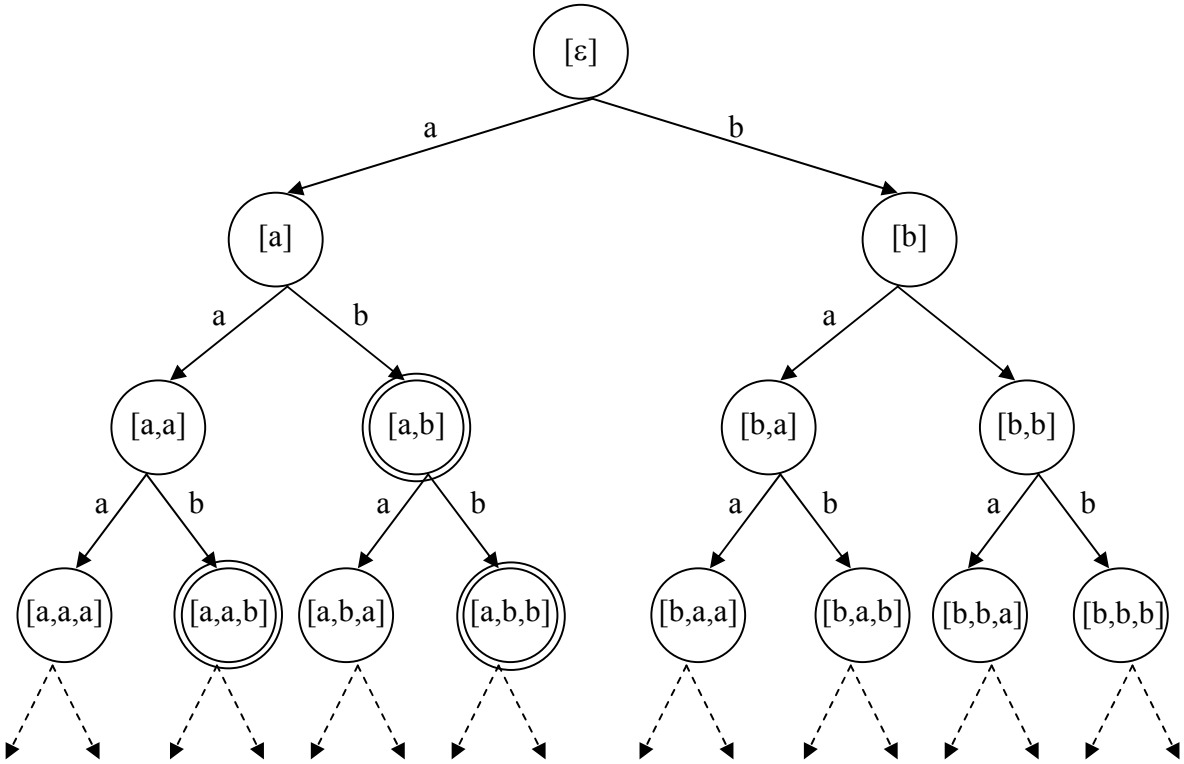
Nel *più grande* automa osservabile per un linguaggio L parole diverse corrispondono a stati diversi, cioè se $w \neq w'$ deve essere $\delta(q_o, w) \neq \delta(q_o, w')$. La funzione $f: \Sigma^* \to Q$ risulta allora una corrispondenza biunivoca: indicheremo con [w] lo stato corrispondente alla parola w.

Dato un linguaggio $L \subseteq \Sigma^*$, il *più grande* automa osservabile per L è l'automa $G_L = < Q , \Sigma , \delta , q_o , F >$ dove:

1. $Q=\{[x] \mid x \in \Sigma^*\}$
2. $\delta([x], \sigma) = [x\sigma]$
3. $q_o = [\varepsilon]$
4. $F = \{[y] \mid y \in L\}$

Si osservi che gli stati di G_L sono essenzialmente le parole in Σ^*, quindi per qualsiasi L l'automa G_L ha sempre infiniti stati.

Esempio 3.1
Il grafo degli stati G_L per il linguaggio $L=\{a^n b^m \mid m,n>0\}$ è il seguente:



Tornando al caso generale, osserviamo che due stati [x], [y] in G_L sono indistinguibili se, per ogni w, $xw \in L \Leftrightarrow yw \in L$. L'automa minimo M_L per L è ottenuto identificando gli stati indistinguibili in G_L, con la costruzione presentata in Sezione 2. Come ci si può aspettare, l'automa minimo è, fra tutti gli automi che riconoscono lo stesso linguaggio, quello che ha il minimo numero di stati.

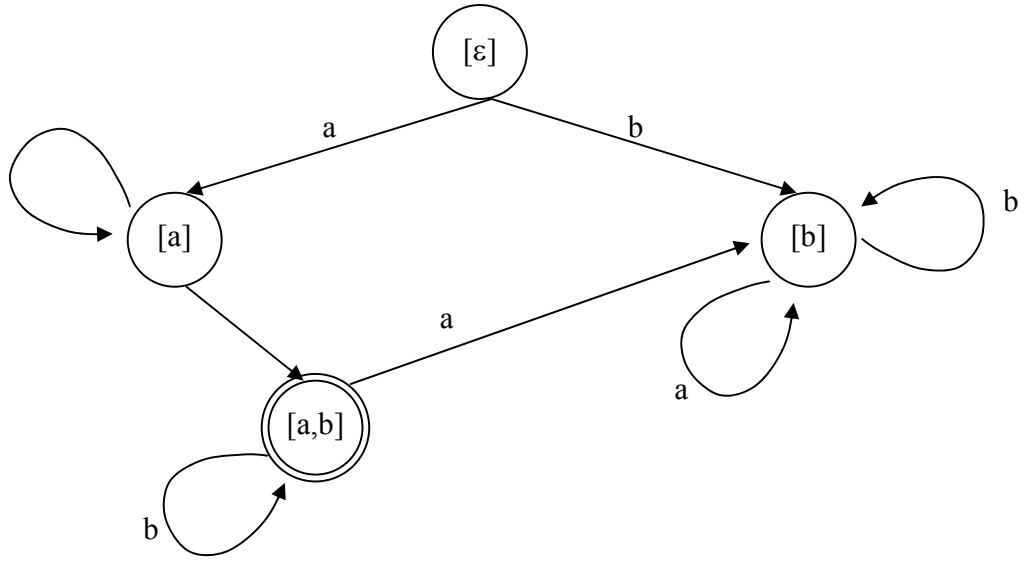
Proposizione 3.1: Se $A = < Q , \Sigma , \delta , q_o , F >$ è un automa che riconosce L, il numero di stati di M_L non può superare quello di A.
Dim. Siano [x], [y] due stati distinguibili in G_L, e quindi per una opportuna parola w vale ad esempio che $xw \in L$ ma $yw \notin L$. Allora gli stati $\delta(q_o, x)$ e $\delta(q_o, y)$ sono stati di A distinti, poiché $\lambda(\delta(\delta(q_o, x), w) = \lambda(\delta(q_o, xw)) = 1$, ma $\lambda(\delta(\delta(q_o, y), w) = \lambda(\delta(q_o, yw)) = 0$.
Siano ora $[x_1]_{\approx}, [x_2]_{\approx}, [x_3]_{\approx}, \ldots$ tutti gli stati di M_L ; poiché $[x_1], [x_2], [x_3], \ldots$ sono stati distinguibili in G_L , per quanto detto sopra $\delta(q_o, x_1)$, $\delta(q_o, x_2)$, $\delta(q_o, x_3)$, … sono stati distinti in A. A possiede almeno tanti stati quanto M_L , da cui la tesi.
L'automa minimo per un linguaggio L può essere dunque determinato dal seguente procedimento:

1. Considera l'automa G_L (descritto da un albero con infiniti nodi)
2. Partendo dalla radice [ε], visita l'albero in ampiezza e modificalo, fino a quando ci sono nuovi nodi da visitare. Visitando il nodo [wσ], cui arriva un arco etichettato σ dal nodo [w], se [wσ] è indistinguibile da un nodo [x] già precedentemente visitato allora collega [w] a [x] con un arco etichettato σ e cancella il sottoalbero di radice [wσ].

3. Il nodo iniziale del grafo degli stati di è [ε], i nodi finali sono quelli del tipo [x] con $x \in L$.

Esempio 3.2:
Sia $L=\{a^n b^m \mid m,n>0\}$.

Per costruire M_L si parta a visitare in ampiezza G_L dalla radice [ε].
1. [a] è distinguibile da [ε], poiché $\varepsilon b \notin L$ ma $ab \in L$.
2. [b] è distinguibile sia da [ε] che da [a], poiché $bab \notin L$ ma $\varepsilon ab \in L$ e $bb \notin L$ ma $ab \in L$.
3. [aa] è indistinguibile da [a], come si verifica facilmente. Collega allora [a] a se stesso con un arco etichettato "a" e sopprimi il sottoalbero di radice [aa].
4. [ab] è distinguibile da [ε], da [a] e da [b], come si verifica facilmente (per esempio, [ab] è distinguibile da [b] perché $bb \notin L$ ma $abb \in L$).
5. [ba] è indistinguibile da [b], come si verifica facilmente. Collega allora [b] a se stesso con un arco etichettato "a" e sopprimi il sottoalbero di radice [ba].
6. [bb] è indistinguibile da [b], come si verifica facilmente. Collega allora [b] a se stesso con un arco etichettato "b" e sopprimi il sottoalbero di radice [bb].
7. [aba] è indistinguibile da [b], come si verifica facilmente. Collega allora [ab] a [b] con un arco etichettato "a" e sopprimi il sottoalbero di radice [aba].
8. [abb] è indistinguibile da [ab], come si verifica facilmente. Collega allora [ab] a se stesso con un arco etichettato "b" e sopprimi il sottoalbero di radice [aba].
9. Non esistendo nuovi nodi da visitare, la costruzione è terminata. L'automa minimo per $L=\{a^n b^m \mid m,n>0\}$ è allora descritto dal seguente grafo degli stati:



Esempio 3.3:
Sia $L=\{a^n b^n \mid n>0\}$. Osserviamo che, in G_L , per $k,n>0$ e se $k \neq n$ vale che $[a^k]$ è distinguibile da $[a^n]$, poichè $a^n b^n \in L$ ma $a^k b^n \notin L$. Ne segue che in M_L gli stati $[a]_{\approx}$, $[a^2]_{\approx}$, … , $[a^k]_{\approx}$, … sono distinti: l'automa minimo contiene dunque infiniti stati, quindi il linguaggio $L=\{a^n b^n \mid n>0\}$ non può essere riconosciuto da automi a stati finiti.

4. Automi a stati finiti e grammatiche di tipo 3

In questa sezione proviamo che i linguaggi regolari (di tipo 3) sono esattamente quelli riconosciuti da automi a stati finiti; gli automi a stati finiti risultano dunque i riconoscitori corrispondenti a quei sistemi generativi che sono le grammatiche di tipo 3.

Proposizione 4.1: Per ogni linguaggio L riconosciuto da un automa a stati finiti esiste una grammatica G di tipo 3 tale che $L=L_G$.

Dim. Sia $A = < Q , \Sigma , \delta , q_o , F >$ un automa a stati finiti che riconosce L. Dato A, costruiamo la seguente grammatica $G=< \Sigma,Q,P, q_o >$ dove:

1. L'insieme dei metasimboli coincide con gli stati dell'automa.
2. L'assioma è lo stato iniziale.
3. $q_k \rightarrow \sigma q_j$ è una regola di produzione di G se $q_j = \delta(q_k, \sigma)$.
4. $q_k \rightarrow \varepsilon$ è una regola di produzione di G se $q_k \in F$.

Per induzione sulla lunghezza della parola, si prova che:

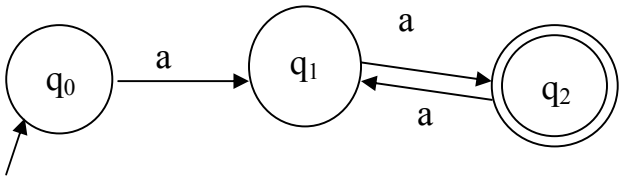
(1) $q_j = \delta(q_0, w)$ se e solo se $q_0 \Rightarrow_G^*$ w q_j per ogni parola w

Se $l(w)=0$, cioè $w=\varepsilon$, la proprietà (1) è verificata. Supponiamo che la proprietà sia verificata per tutte le parole di lunghezza al più n, e consideriamo una parola $w\sigma$ di lunghezza n+1. Supponiamo che $q_j=\delta(q_0, w)$ e che $q= \delta(q_0, w\sigma)$, così che $q= \delta(q_j, \sigma)$ poiché $\delta(q_0, w\sigma)= \delta(\delta(q_0, w), \sigma)$,Poiché $l(w)=n$, per ipotesi di induzione esiste un unico q_j tale che $q_0 \Rightarrow_G^*$ w q_j ; per la regola di costruzione della grammatica numero 3, l'unica regola in G che permette di riscrivere q_j con una parola iniziante per q è $q_j \rightarrow \sigma q$. Ne segue:

$q = \delta(q_0, w\sigma)$ se e solo se $q_0 \Rightarrow_G^*$ $w\sigma q$

Se q_j è stato finale, allora osserviamo ulteriormente che $q_0 \Rightarrow_G^*$ w se e solo se $q_0 \Rightarrow_G^*$ wq_j e q_j è finale (per la regola di costruzione della grammatica numero 4). Concludiamo che $q_0 \Rightarrow_G^*$ w se e solo se $\delta(q_0, w) \in F$, che equivale a dire: se e solo se $w \in L$.

Esempio 4.1:
Sia dato il seguente automa, descritto dal diagramma degli stati:



Il linguaggio accettato dall'automa è generabile dalla grammatica con assioma q_0 e dalle seguenti regole di produzione:
$q_0 \rightarrow aq_1$, $q_2 \rightarrow aq_1$, $q_1 \rightarrow aq_2$,
$q_2 \rightarrow \varepsilon$

Abbiamo visto che, per ogni automa a stati finiti è possibile costruire una grammatica di tipo 3 che genera il linguaggio riconosciuto dall'automa. Poniamoci il problema inverso: data una grammatica di tipo 3, costruire una automa che riconosce il linguaggio generato dalla grammatica.

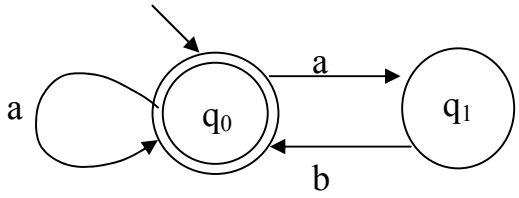
Si consideri, a questo riguardo, una grammatica $G =< \Sigma,Q,P,S>$ di tipo 3; per la **Proposizione 5.2** in **Cap.1** possiamo supporre, senza perdita di generalità, che la grammatica contenga solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \varepsilon$. Cercando di invertire la costruzione data in **Proposizione 4.1,** possiamo costruire un grafo con archi etichettati Σ come segue:

1. Vertici del grafo sono i metasimboli in Q.
2. Nel grafo c'è un arco da A a B etichettato σ se $A \rightarrow \sigma B$ è una produzione di G.
3. Nodo iniziale è S
4. A è un nodo finale se $A \rightarrow \varepsilon$ è una regola di G.

Esempio 4.2:
Data la grammatica $G =< \Sigma,Q,P,S>$ con regole di produzione

$q_0 \rightarrow aq_0$
$q_0 \rightarrow aq_1$
$q_1 \rightarrow bq_0$
$q_0 \rightarrow \varepsilon$

il grafo con archi etichettati associato è il seguente:



Interpretando i metasimboli come stati, osserviamo che il grafo precedente non è il grafo degli stati di un automa. In un automa, infatti, dato uno stato q e un simbolo σ esiste *un unico stato prossimo* $\delta(\sigma,q)$; nel grafo precedente, ci sono invece due transizioni etichettate con σ che portano da q_0 a due distinti stati q_0 e q_1. Possiamo interpretare questo fatto come una specie di *non determinismo*: se il sistema si trova in un dato stato (ad esempio q_0), l'arrivo di un messaggio non porta necessariamente ad uno stato prossimo univocamente individuato dallo stato presente e dal messaggio, bensì porta in uno tra un insieme di stati *possibili* (nel nostro esempio q_0 o q_1). Questo porta alla seguente:

Definizione 4.1: Un *automa a stati finiti non deterministico* A è un sistema $A = < Q , \Sigma , R , q_o , s_0 >$, dove Q è *un insieme finito di stati*, Σ è un alfabeto finito, $R \subseteq Q x \Sigma x Q \rightarrow Q$ è la *relazione di transizione* (R si può anche indicare come $R:Q x \Sigma \rightarrow P(Q)$ dove P(Q) è l'insieme delle parti di Q), $q_o \in Q$ è lo *stato iniziale*, $F \subseteq Q$ è l'insieme degli *stati finali*.

Una parola $w= x_1 \ldots x_m$ è riconosciuta dall'automa non deterministico A se essa induce almeno un cammino $s_0 , s_1, \ldots, s_m$ dallo stato iniziale $s_0 = q_o$ a uno stato finale $s_m \in F$, cioè se:

1. $s_0 = q_o$
2. $R(s_0, x_1 , s_1)= \ldots = R(s_k, x_{k+1} , s_{k+1}) = \ldots = R(s_{m-1}, x_m , s_m)=1$
3. $s_m \in F$

Il linguaggio L(A) riconosciuto dall'automa non deterministico A è l'insieme delle parole riconosciute.

Osserviamo che un automa a stati finiti $A = < Q , \Sigma , \delta , q_o , F >$ è un particolare automa non deterministico $A = < Q , \Sigma , R , q_o , F >$, in cui $R(q, \sigma,q')=1$ se e solo se $q'=\delta(q, \sigma)$. La relazione R è equivalente in tal caso ad una funzione $\delta: \Sigma xQ \rightarrow Q$, ed ogni parola $w= x_1 \dots x_m$ induce un unico cammino partente dallo stato iniziale.

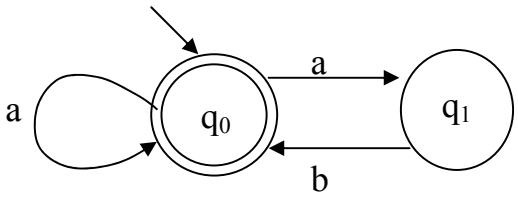
Ovviamente esistono automi a stati finiti non deterministici che non sono automi a stati (si veda **Esempio 4.2**). Ciò nonostante, i linguaggi riconosciuti da automi non deterministici coincidono con quelli riconosciuti da automi deterministici:

Proposizione 4.2: Per ogni linguaggio L riconosciuto da un automa a stati finiti non deterministico finiti esiste un automa a stati finiti che lo riconosce.
Dim. Sia $A = < Q , \Sigma , R , q_o , F >$ un automa a stati finiti non deterministico che riconosce L. Possiamo costruire il seguente automa deterministico $A' = < Q' , \Sigma , \delta , q_{o'} , F' >$ che riconosce lo stesso linguaggio come segue:

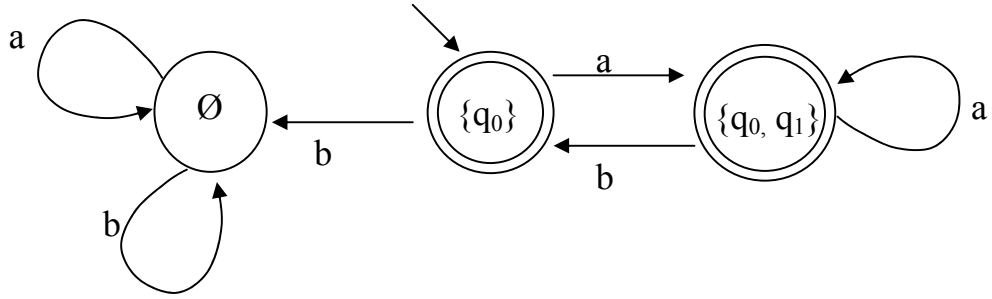
1. $Q' = 2^Q$, cioè gli stati del nuovo automa A' sono i sottoinsiemi degli stati di A.
2. Se $X \subseteq Q$, allora $\delta(X, \sigma)=\{q' | R(q,\sigma,q')=1 \text{ e } q \in X\}$, cioè $\delta(X, \sigma)$ è l'insieme di stati accessibili da X con un arco etichettato con σ.
3. $q_{o'} = \{ q_o\}$
4. $F' = \{X | X \subseteq Q \text{ e } X \cap F \neq \emptyset\}$, cioè F' è formato dai sottoinsiemi di Q che contengono almeno un stato in F.

Per induzione sulla lunghezza della parola, a causa delle condizioni 2. e 3. di dimostra facilmente che:
$$\delta(q_{o'}, x_1 \dots x_m,) = \{q | x_1 \dots x_m \text{ induce un cammino } q_o, s_1, \dots, s_m \text{ da } q_o \text{ allo stato } s_m = q\}$$
Per la 4. segue infine che $x_1 \dots x_m$ è riconosciuta da A se e solo se è riconosciuta da A'.

Esempio 4.2:
Sia L il linguaggio riconosciuto dall'automa non deterministico descritto dal seguente grafo:



Un automa deterministico equivalente, ottenuto con la costruzione mostrata in **Proposizione 4.2**, è il seguente:



Nota bene: nel grafo così ottenuto non viene visualizzato lo stato $\{q_1\}$ in quanto tale stato non è osservabile.

Possiamo a questo punto concludere:

Teorema 4.1. Le due seguenti affermazioni sono equivalenti:
(1) L è generato da una grammatica di tipo 3.
(2) L è riconosciuto da un automa a stati finiti
Dim.
(1) implica (2).
E' il contenuto di **Proposizione 4.1.**
(2) implica (1).
Data una grammatica G di tipo 3 che genera L, si costruisce il grafo etichettato associato, che può essere interpretato come automa a stati finiti non deterministico che riconosce L. Si costruisce infine come in **Proposizione 4.2** un automa a stati finiti che riconosce L.

In **Proposizione 4.2** abbiamo mostrato che per ogni automa non deterministico a stati finiti esiste un automa deterministico equivalente. Va per contro osservato che, se l'automa non deterministico ha M stati, il nuovo automa deterministico può avere fino a 2^M stati, cosa che rende inutilizzabile la costruzione in molte applicazioni.

5. Automi a stati finiti e espressioni regolari

In questa sezione introduciamo una classe di espressioni (le espressioni regolari) ed associamo in modo naturale ad ogni espressione un linguaggio. Mostriamo poi che la classe di linguaggi denotati da espressioni regolari è la classe di linguaggi riconosciuti da automi a stati finiti (Teorema di Kleene).

Definizione 5.1: Dato un alfabeto Σ, le *espressioni regolari* su Σ sono definite induttivamente:
1. Ø, ε, σ (per $\sigma\in\Sigma$) sono espressioni regolari
2. se p, q sono espressioni regolari, allora (p+q), (p.q), (p*) sono espressioni regolari

Richiamiamo ora che, dati linguaggi L_1, L_2 e L sull'alfabeto Σ:
1. Unione di L_1 e L_2 è il linguaggio $L_1 \cup L_2 = \{x | x\in L_1 \text{ o } x\in L_2\}$
2. Prodotto di L_1 e L_2 è il linguaggio $L_1.\ L_2 = \{xy | x\in L_1 \text{ e } y\in L_2\}$
3. Chiusura (di Kleene) di L è il linguaggio $L^* = L^0 \cup L^1 \cup \ldots \cup L^k \cup \ldots = \bigcup_{k=0}^{\infty} L^k$

Ad ogni espressione regolare p associamo un linguaggio L (e diremo che p *denota* L) come segue:
1. Ø denota il linguaggio vuoto, ε denota il linguaggio $\{\varepsilon\}$, σ denota il linguaggio $\{\sigma\}$
2. se p,q,m denotano rispettivamente L_1, L_2 e L, allora (p+q), (pq), (m*) denotano rispettivamente $L_1\cup L_2$, $L_1.\ L_2$, L^*.

Esempio 5.1:
a*b*a*+(ab)* è una espressione regolare che denota il linguaggio L dove:
$L = \{w \mid w=a^j b^k a^l \ (j,k,l\geq 0) \text{ o } w=(ab)^k \ (k\geq 0)\}$.
Il linguaggio $\{w \mid w=a^{2k+1} \ (k\geq 0)\}$ è denotato dall'espressione regolare a.(aa)* , oppure da (aa)*a o (aa)*a(aa)* (o da altre ancora).

Osserviamo che le espressioni regolari denotano linguaggi in modo *composizionale*: una data espressione indica le operazioni di unione, prodotto e chiusura che, applicate ai linguaggi-base Ø, {ε}, {σ}, permettono di ottenere il linguaggio denotato. Questo permette l'uso di tecniche induttive per mostrare proprietà dei linguaggi denotati da espressioni regolari. Ad esempio, per mostrare che una proprietà P vale per tutti i linguaggi denotati da espressioni regolari, basta provare che:

1. Il linguaggi base Ø, {ε}, {σ} verificano la proprietà P
2. Se i linguaggi A,B verificano la proprietà P, allora A∪B, AB, A* verificano la proprietà P.

Esempio 5.2:

Data una parola $w=x_1x_2..\ x_m$, la sua *trasposta* w^R è la parola $w^R = x_mx_{m-1}..\ x_1$; dato un linguaggio L, il suo *trasposto* L^R è il linguaggio $\{w \mid w^R \in L\}$. Vogliamo provare che se L è denotato da una espressione regolare, allora anche L^R lo è. A tal riguardo, basta osservare:

1. $Ø^R = Ø$, $\{ε\}^R = \{ε\}$, $\{σ\}^R = \{σ\}$

2. $(A∪B)^R = A^R ∪ B^R$, $(AB)^R = B^RA^R$, $(A^*)^R = (A^R)^*$

Le regole precedenti permettono, conoscendo l'espressione che denota L, di costruire l'espressione che denota L^R. Per esempio, se (a+b)(ac+ba)* denota L, allora (ca+ab)*(a+b) denota L^R.

I linguaggi denotati da espressioni regolari sono tutti e soli quelli riconosciuti da automi a stati finiti, come enunciato nel seguente:

Teorema (di Kleene) 5.1: Le due seguenti affermazioni sono equivalenti:
(1) L è denotato da una espressione regolare
(2) L è riconosciuto da un automa a stati finiti

Dim.
(1) implica (2).

Basta provare che:

1. Ø, {ε}, {σ} sono riconosciuti da automi a stati finiti
2. Se A, B sono riconosciuti da automi a stati finiti, allora A∪B, AB, A* lo sono.

Per il punto 1., basta osservare che:

sono automi che riconoscono rispettivamente Ø, {ε}, {σ}.
Per il punto 2., supponiamo che A,B siano riconosciuti da automi a stati finiti.
Per la **Proposizione 4.1**, A è generato da una grammatica $G'=< Σ,Q',P',S' >$ di tipo 3 e B è generato da una grammatica $G''=< Σ,Q'',P'',S'' >$ di tipo 3. Senza perdita di generalità, possiamo supporre che le regole siano del tipo $q_k \rightarrow σq_j$ oppure $q_k \rightarrow ε$, ed inoltre che Q' e Q'' siano insiemi disgiunti.
Si verifica facilmente che:

a. A∪B è generato dalla grammatica $G_1=< Σ,Q_1,P_1,S_1 >$, dove Q_1 contiene i metasimboli in Q', i metasimboli in Q'' e un nuovo metasimbolo S_1, mentre P_1 contiene le regole in *P*', le regole in *P*'' e le nuove regole $S_1 \rightarrow S'$, $S_1 \rightarrow S''$.

b. AB è generato dalla grammatica $G_2=< \Sigma,Q_2,P_2,S' >$, dove Q_2 contiene i metasimboli in Q' e in Q", mentre P_1 contiene le regole in *P*', ad esclusione di quelle del tipo q'→ε, tutte le regole in *P*" e le nuove regole q'→S" per ogni metasimbolo q' in Q' per cui q'→ε è una regola in *P'*.
c. A* è generato dalla grammatica $G_3=< \Sigma,Q', P_3, S' >$, dove P_3 contiene le regole in P' più le nuove regole q'→S' per ogni metasimbolo q' in Q' per cui q'→ε è una regola in *P'*.

(2) implica (1)

Sia L riconosciuto dall'automa a stati finiti $A = < Q , \Sigma , \delta , q_o , F >$; vogliamo esibire una espressione regolare che denota L.
A tal riguardo, per ogni stato q_k consideriamo l'automa $A_k = < Q , \Sigma , \delta , q_k , F >$ che differisce da A solo per lo stato iniziale, che è q_k anziché q_0. Sia X_k il linguaggio riconosciuto da A_k, così che $L=X_0$. Osserviamo che:

1. $\varepsilon \in X_k$ se e solo se q_k è stato finale.
2. $\sigma w \in X_k$ se e solo se $\delta(q_k,\sigma)=q_j$ e $w \in X_j$.

Per ogni stato finale q_k , possiamo quindi scrivere l'equazione:

$$X_k = \{\varepsilon\} \cup (\cup_{\sigma \in \Sigma, \delta(qk\ ,\ \sigma)=qj}\ \sigma X_j)$$

Analogamente, per ogni stato non finale q_s possiamo scrivere l'equazione:

$$X_s = (\cup_{\sigma \in \Sigma, \delta(\sigma,qs)=qj}\ \sigma X_j)$$

Abbiamo ora un sistema in cui le incognite sono linguaggi; il numero di incognite è inoltre pari al numero delle equazioni. Va segnalato inoltre che il sistema è "lineare a destra" (la parte sinistra di ogni equazione è una variabile, mentre la parte destra è una unione di prodotti, ed in ogni prodotto l'incognita compare al più una volta ed in ultima posizione). Allo scopo di risolvere tale sistema, cominciamo con lo studio dell'equazione

$$(\#) \quad X=AX \cup B$$

dove A,B sono linguaggi e supponiamo che $\varepsilon \notin A$. Tale equazione ammette una sola soluzione che è

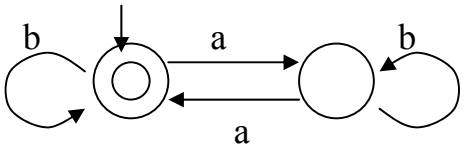
$$X=A^*B$$

- A^*B è una soluzione, come si verifica immediatamente:
  $A(A^*B)\cup B=A(\{\varepsilon\}\cup A\cup A^2...\cup A^k..)B\cup B=(A\cup A^2...\cup A^k..)B\cup B=(A\cup A^2...\cup A^k..\cup\{\varepsilon\})B=A^*B$
- Tale soluzione è inoltre la sola. Supponiamo infatti che ci siano due soluzioni distinte X,Y tali che $X=AX\cup B$ e $Y=AY\cup B$. Sia h la lunghezza della più corta parola che distingue X da Y (si trova in un linguaggio e non nell'altro) e sia inoltre s la lunghezza della più corta parola in A ($s>0$ poiché per ipotesi $\varepsilon \notin A$). La più corta parola che distingue $AX\cup B$ da $AY\cup B$ risulta allora di lunghezza $h+s>h$: assurdo, poiché $X=AX\cup B$ e $Y=AY\cup B$.

Il sistema può essere risolto per sostituzione: si fissa una equazione e una incognita in essa contenuta, la si risolve come (#) e si sostituisce il risultato in ogni altra equazione, applicando poi opportunamente la proprietà distributiva. Si ottiene un sistema dello stesso tipo, con una equazione in meno ed una incognita in meno.

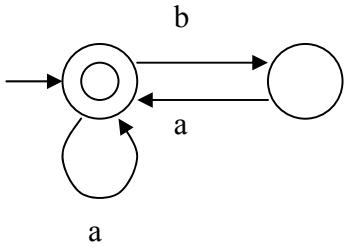
Al termine ogni linguaggio X_k, e quindi in particolare L= X_0, verrà ottenuto applicando a Ø, {ε}, {σ} un numero finito di volte le operazioni di unione, prodotto e chiusura.

Esempio 5.3:
Dato il seguente automa che riconosce il linguaggio L, trovare l'espressione regolare che denota L.



Detto q_0 lo stato iniziale e q_1 l'altro stato, otteniamo il sistema (utilizzando il simbolo + invece di ∪):
X_0=ε+ aX_1+bX_0
X_1= aX_0+bX_1
Risolvendo la seconda equazione:
X_1= b*aX_0
Sostituendo la soluzione data dalla seconda equazione nella prima e raccogliendo:
X_0 = ε +a(b*aX_0)+ bX_0= ε + ab*aX_0+ bX_0= ε + X_0 (ab*a + b)
Risolvendo:
X_0 = (ab*aX_0 +b)* ε=(ab*aX_0 +b)*=L

Esempio 5.4:
Il linguaggio L={w | w∈{a,b}* e bb non è fattore di w} è riconosciuto dal seguente automa non deterministico:



Trovare l'espressione regolare che determina lo stesso linguaggio.

Detto q_0 lo stato iniziale e q_1 l'altro stato, otteniamo il sistema (utilizzando il simbolo + invece di ∪):
X_0 = aX_0+bX_1+ε
X_1 = aX_0
Sostituendo la soluzione data dalla seconda equazione nella prima e raccogliendo:
X_0 = (a+ba)X_0+ε
Risolvendo:
X_0 = (a+ba)*ε
Da cui L= X_0 = (a+ba)*, ottenendo una espressione regolare che denota L.

Capitolo 3: Linguaggi liberi dal contesto

1. Albero di derivazione e derivazione "più a sinistra"

Ricordiamo che un linguaggio L è *acontestuale* (o *libero dal contesto*) se è generato da una grammatica di tipo 2, in cui cioè ogni regola di produzione è della forma $\alpha \rightarrow \beta$ dove α è un metasimbolo. Questa particolarità permette di dare una semplice descrizione visiva di una "derivazione " mediante il cosiddetto *albero di derivazione*.

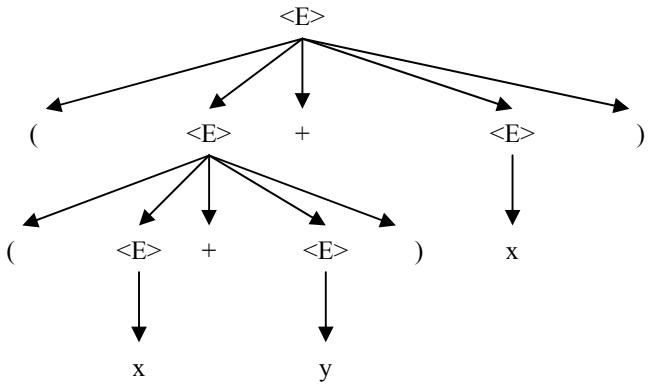
Data una grammatica $G = < \Sigma, Q, P, S >$ di tipo 2 che genera il linguaggio L(G), un *albero di derivazione* della parola $w \in L(G)$ in G è un albero ordinato i cui nodi sono etichettati con simboli in $\Sigma \cup Q$, così che:

- ogni foglia è etichettata con un simbolo terminale, mentre ogni nodo interno è etichettato con un metasimbolo.
- la radice è etichettata con l'assioma S
- se un nodo interno è etichettato col metasimbolo B e i suoi figli sono etichettati, in ordine, coi metasimboli $B_1 \ldots B_m$, allora $B \rightarrow B_1 \ldots B_m$ è una regola di produzione di G
- leggendo le foglie in ordine prefisso, la sequenza di etichette forma la parola w.

Esempio 1.1:
Data la grammatica G =< {(,),+,x.,y}},{E},{E→(E+E)/x/y},E> , si consideri la seguente derivazione della parola ((x+y)+x):

$$E \Rightarrow_G (E+E) \Rightarrow_G ((E+E)+E) \Rightarrow_G ((x+E)+E) \Rightarrow_G ((x+y)+E) \Rightarrow_G ((x+y)+x)$$

L'albero di derivazione corrispondente è:



Si osservi che diverse derivazioni possono avere associato lo stesso albero. Per esempio, riferendoci alla grammatica dell'esempio precedente, la seguente derivazione, diversa da quella presentata sopra, genera lo stesso albero:

$$E \Rightarrow_G (E+E) \Rightarrow_G (E+x) \Rightarrow_G ((E+E)+x) \Rightarrow_G ((E+y)+E) \Rightarrow_G ((x+y)+x)$$

Data una grammatica G, diremo che due derivazioni sono *equivalenti* se hanno associato lo stesso albero. Può essere interessante ottenere un elemento rappresentativo nella classe delle derivazioni equivalenti, compatibili quindi con lo stesso albero di derivazione. Una soluzione è quella di considerare la cosiddetta "*derivazione più a sinistra*", corrispondente alla visita dell'albero in profondità. Data una parola w contenente almeno un metasimbolo, è infatti univocamente individuato il

metasimbolo “più a sinistra” nella parola. Ad esempio, nella parola abAbbCcCB il metasimbolo più a sinistra è A. Questo porta alla seguente:

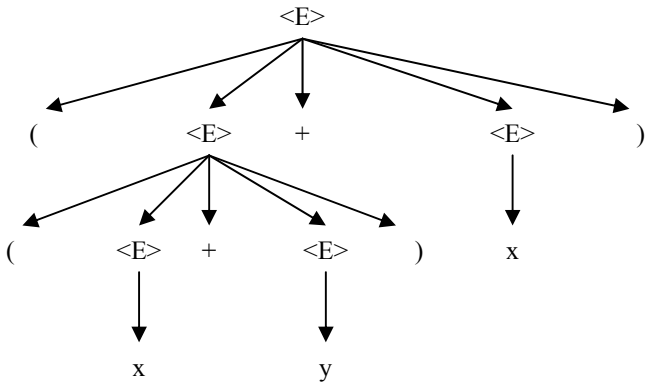
Definizione 1.1: data una grammatica G = < Σ,Q,*P*,S> di tipo 2, sia w∈Σ* una parola derivabile dall’assioma S. La derivazione $S \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_m \Rightarrow_G w$ è detta “derivazione più a sinistra” se, per ogni j ($1 \le j \le m$), w_{j+1} è ottenuta applicando una regola di produzione al metasimbolo più a sinistra in w_j .

Ovviamente, ogni albero di derivazione in G ha associata un derivazione “più a sinistra” e viceversa: le derivazioni “più a sinistra” risultano allora in corrispondenza biunivoca con gli alberi di derivazione.

2. Grammatiche di tipo 2 ambigue e non ambigue

In vari contesti, data una grammatica G di tipo 2 che genera il linguaggio L(G) ed una parola w∈L(G), risulta ragionevole interpretare un albero di derivazione di w in G come “*significato*” della parola w.

Consideriamo, a scopo esemplificativo, la grammatica G =< {(,),+,x.,y}},{E},{E→(E+E)/x/y},E> : possiamo interpretare le parole del linguaggio generato L(G) come particolari *espressioni aritmetiche* in un linguaggio di programmazione. Assegnando ai simboli x,y un valore numerico e interpretando + come operazione binaria, ad esempio la usuale somma, deve essere possibile attribuire un preciso valore ad ogni parola del linguaggio: in questo caso, l’albero di derivazione della parola stabilisce l’ordine di applicazione delle operazioni così da individuare univocamente il valore dell’espressione. Ad esempio, la parola ((x+y)+x) ha associato l’albero di derivazione:

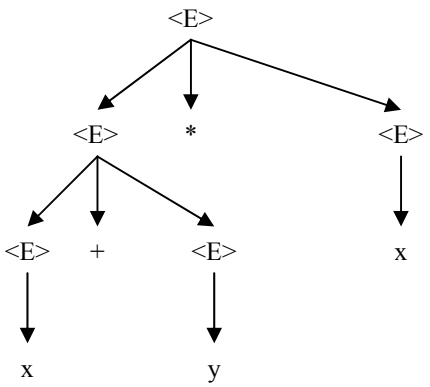


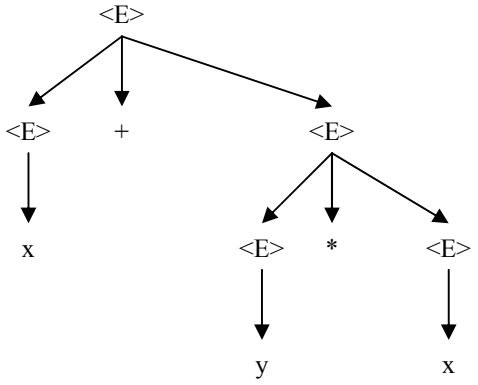
Se x=3 e y=5, l’albero di derivazione permette di calcolare il valore ((3+5)+3)=11 dell’espressione.

Una richiesta importante è che ogni parola in L(G) abbia un unico significato.
Attribuendo un albero di derivazione di w in G come “*significato*” della parola w, può succedere che la stessa parola abbia diversi alberi di derivazione, e quindi significati diversi, risultando *ambigua*. Questo porta alla seguente definizione:

Definizione 2.1: Una grammatica G = < Σ,Q,*P*,S> di tipo 2 è detta *ambigua* se esiste una parola w∈L(G) che ammette due diversi alberi di derivazione; una grammatica G = < Σ,Q,*P*,S> di tipo 2 è detta *non ambigua* se ogni parola w∈L(G) ammette un unico albero di derivazione.

Esempio 2.1:
Data la grammatica G =< {+,*,x.,y}},{E},{E→E+E/E*E/x/y},E> , si consideri la parola x+y*x, Essa ammette i seguenti due distinti alberi di derivazione:





Tale grammatica risulta dunque ambigua.

Esempio 2.2:
Data la grammatica G =< {(,),+,*,x.,y}},{E},{E→(E+E)/(E*E)/x/y},E> , è possibile mostrare che ogni parola del linguaggio generato ammette esattamente un albero di derivazione (o, equivalentemente, una derivazione più a sinistra). Dimostriamolo per induzione sulla lunghezza della parola derivata.
Le parole di lunghezza 1 in L(G) sono x e y, che hanno un'unica derivazione (e quindi un unico albero di derivazione).
Sia $w \in L(G)$ una parola di lunghezza n>1. Essa è della forma $(w_1+ w_2)$ o alternativamente $(w_1* w_2)$, dove $|(w_1|,| w_2)|<n$. Supponiamo senza perdere di generalità che $w=(w_1+ w_2)$. Allora w è generata da una derivazione "più a sinistra" del tipo $E \Rightarrow_G (E+E) \Rightarrow_G *(w_1+E) \Rightarrow_G * (w_1+ w_2)$, Per ipotesi di induzione, esiste un'unica derivazione "più a sinistra" del tipo $E \Rightarrow_G * w_1$ e del tipo $E \Rightarrow_G * w_2$, quindi esiste un'unica derivazione "più a sinistra" di w.

Se L è generato da una grammatica ambigua G di tipo 3, è sempre possibile costruire una grammatica non ambigua G' di tipo 3 che genera L. Basta infatti costruire l'automa deterministico che riconosce L: la relativa grammatica non è ambigua.

Esistono invece linguaggi acontestuali che possono essere generati solo da grammatiche di tipo 2 ambigue: tali linguaggi sono detti *inerentemente ambigui*.

Esempio 2.2:
Si consideri il linguaggio

$$L = \{a^j b^s c^k \mid j=s \text{ oppure } k=s \}.$$

Esso è acontesuale, poiché è generato dalla grammatica con assioma S e produzioni del tipo:

S → XC / AY , Y → aYb / ε , X → bXc / ε , A → aA / ε , C → cC / ε

Tale grammatica è ambigua poichè le parole del tipo $a^j b^j c^j$ ammettono due distinti alberi di derivazione. E' inoltre possibile mostrare che, per ogni grammatica G di tipo 2 che genera L, esistono infinite parole del tipo $a^j b^j c^j$ che ammettono due distinte derivazioni.

3. Forme normali di Chomsky e di Greibach.

Sappiamo che se L è di tipo 2 con $\varepsilon \in L$, allora $L=L' \cup \{\varepsilon\}$, dove L' è generato da una grammatica con regole del tipo $A \rightarrow x$, con $x \in (\Sigma \cup Q)^+$. Pertanto in questo paragrafo prenderemo in considerazione solo linguaggi liberi dal contesto non contenenti la parola vuota.

Due importanti sottoclassi di grammatiche di tipo due sono le seguenti:

Definizione 3.1: Una grammatica $G = < \Sigma, Q, P, S >$ di tipo 2 è detta in forma normale di Chomsky se le sue regole di produzione sono del tipo $A \rightarrow BC$ oppure $A \rightarrow x$, dove A,B,C sono metasimboli e x è un simbolo terminale.

Definizione 3.2: Una grammatica $G = < \Sigma, Q, P, S >$ di tipo 2 è detta in forma normale di Greibach se le sue regole di produzione sono del tipo $A \rightarrow xW$, dove x è un simbolo terminale e W una parola (eventualmente vuota) di metasimboli.

E' possibile mostrare la seguente:

Proposizione 3.1: Per ogni linguaggio L libero dal contesto non contenente la parola vuota, esiste una grammatica G' in forma normale di Chomsky che genera L ed esiste una grammatica G'' in forma normale di Greibach che genera L.

Esempio 3.1:
Si consideri la grammatica con assioma E e regole di produzione:

$$E \rightarrow (E+E) \,/\, x \,/\, y$$

Osserviamo che tale grammatica non è in forma normale di Chomsky, per via della regola $E \rightarrow (E+E)$.
Tale regola può tuttavia essere sostituita dalle regole:

$$E \rightarrow AEBEC \;,\; A \rightarrow (\;,\; B \rightarrow + \;,\; C \rightarrow)$$

ottenendo una grammatica equivalente. A sua volta, la regola $E \rightarrow AEBEC$ può essere sostituita dalle regole $E \rightarrow DBEC$, $D \rightarrow AE$, e, infine, la regola $E \rightarrow DBEC$ può essere sostituita dalle regole $E \rightarrow FG$, $F \rightarrow DB$, $G \rightarrow EC$, ottenendo una grammatica equivalente. In sostanza, si ottiene la grammatica equivalente con assioma E e con regole:

$$E \rightarrow FG,\; F \rightarrow DB,\; G \rightarrow EC,\; D \rightarrow AE,\; A \rightarrow (\,,\; B \rightarrow +\,,\; C \rightarrow)\;,\; E \rightarrow x\;,\; E \rightarrow y$$

Tale grammatica è in forma normale di Chomsky.

Esempio 3.2:
Si consideri la grammatica con assioma E e regole di produzione:

$$E \rightarrow (E+E) \,/\, x \,/\, y$$

Osserviamo che tale grammatica non è in forma normale di Greibach, per via della regola $E \rightarrow (E+E)$.
Tale regola può tuttavia essere sostituita dalle regole:

$$E \rightarrow (EAEB \;,\; A \rightarrow + \;,\; B \rightarrow)$$

ottenendo una grammatica equivalente. Otteniamo allora una grammatica in forma normale di Greibach, equivalente alla precedente. Tale grammatica ha assioma E e regole di produzione:

$$E \rightarrow (EAEB \;,\; A \rightarrow + \;,\; B \rightarrow)\;,\; E \rightarrow x\;,\; E \rightarrow y$$

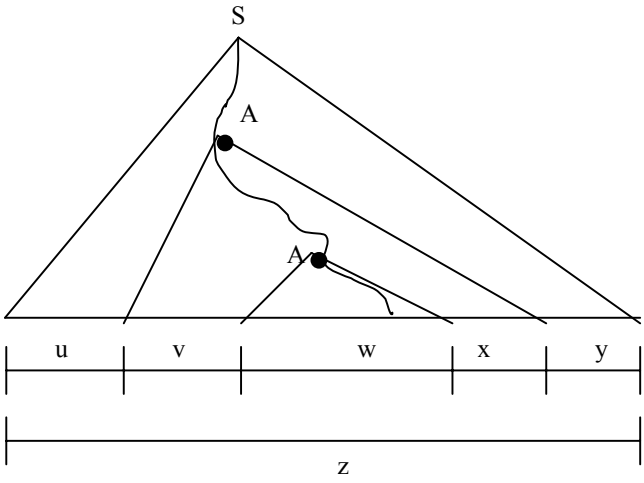
4.“Pumping lemma” e linguaggi non liberi dal contesto.

Osserviamo che ogni albero di derivazione di una grammatica in forma normale di Chomsky è un albero binario. Poiché un albero binario con n foglie ha altezza almeno $\log_2 n$, si ottiene una condizione (pumping lemma) necessaria affinché un linguaggio sia acontestuale, che può essere utilizzata per provare che certi linguaggi non sono liberi dal contesto:

Proposizione 4.1 (pumping lemma): Per ogni linguaggio libero dal contesto L esiste una costante H tale che ogni parola $z \in L$ con lunghezza $|z|>H$ può essere decomposta nella forma z=uvwxy tale che:

1. $|vx| \geq 1$ (almeno una parola tra x e v è diversa da ε)
2. $|vwx| \leq H$
3. per ogni $k \geq 0$ vale che $uv^k wx^k y \in L$

Dim. Sappiamo che esiste una grammatica libera dal contesto G in forma normale di Chomsky che genera $L-\{\varepsilon\}$; sia h il numero di metasimboli di G e sia S l’assioma di G. Posto $H=2^{h+1}$, sia z una parola in L con $|z|>H$. Poiché $z \in L$, esiste un albero di derivazione di z con H foglie e, quindi, con altezza almeno $\log_2 H$; consideriamo ora il più lungo cammino C dalla radice a una foglia, avente quindi lunghezza almeno $h+1 = \log_2 H$. Consideriamo ora un sottocammino di C composto dagli ultimi h+1 nodi consecutivi prima della foglia; poiché G ha h metasimboli, nel sottocammino ci sarà un metasimbolo (diciamo A) ripetuto in due nodi differenti. Chiamiamo w la parola ottenuta dalle foglie del sottoalbero che ha come radice la seconda ripetizione di A; analogamente chiamiamo vwx la parola ottenuta dalle foglie del sottoalbero che ha come radice la prima ripetizione di A. Risulta quindi che z=uvwxy per opportune parole u ed y.



Risulta inoltre:

1. $|vx| \geq 1$. Infatti almeno una parola tra x e v è diversa da ε, altrimenti i due nodi (distinti) etichettati con A sarebbero coincidenti.
2. $|vwx| \leq H$. Infatti l’albero che ha come radice la prima ripetizione di A ha altezza al più h+1, e quindi possiede al più $H=2^{h+1}$ foglie.

3. per ogni $k\geq 0$ vale che $uv^kwx^ky\in L$. Infatti dall'analisi dell'albero di derivazione si deduce che: $S\Rightarrow_G^* uAy$, $A\Rightarrow_G^* vAx$, $A\Rightarrow_G^* w$. Allora, per ogni $k\geq 0$ vale che $S\Rightarrow_G^* uAy \Rightarrow_G^* u\,vAx\,y \Rightarrow_G^* u\,v^2\,Ax^2\,y \Rightarrow_G^* \ldots \Rightarrow_G^* u\,v^k\,Ax^k\,y \Rightarrow_G^* u\,v^k\,w\,x^k\,y$

Il risultato precedente viene usato per dimostrare che certi linguaggi non sono liberi dal contesto; a tale scopo, basta infatti provare che un dato linguaggio non verifica il "pumping lemma".

Esempio 4.2:
Consideriamo il linguaggio $L = \{a^n\,b^n\,c^n \mid n\geq 1\}$ e dimostriamo che questo linguaggio non è acontestuale, come affermato in Proposizione 4.1 di Cap.1. A tale scopo, supponiamo per assurdo che lo sia e che quindi verifichi il "pumpimg lemma"; esiste quindi una costante H tale che la parola $z=a^Hb^Hc^H \in L$ con lunghezza $|z|= 3H>H$ può essere decomposta nella forma $z=uvwxy$ tale che $|vwx|\leq H$, $|vx|\geq 1$ e $uv^2wx^2y\in L$. La parola vwx non può contenere sia a che c infatti, se così fosse, dovrebbe contenere anche tutti i b, che sono H (per l'ipotesi $|z|=3H$) e ciò sarebbe in contrasto con l'ipotesti $|vwx|\leq H$. Supponiamo quindi che c non sia contenuto in vwx. Il numero di simboli c contenuti in uv^2wx^2y è allora uguale al numero di simboli c contenuti in $z = uvwxy$, cioè H. Sappiamo che $uv^2wx^2y\in L$ e che la lunghezza di una parola in L è il triplo del numero di c contenuti, quindi $|uv^2wx^2y| = 3H$. Questo è assurdo, poiché $3H=|uv^2wx^2y|=| uvwxy|+|vx|=3H+|vx|>3H$, essendo $|vx|\geq 1$.

5. Riconoscitori per linguaggi liberi dal contesto

Sappiamo che esistono linguaggi acontestuali, ad esempio $L = \{a^n\,b^n \mid n\geq 1\}$, non riconoscibili con automi a stati finiti. Il problema è che un automa a stati finiti possiede una memoria limitata dal numero degli stati, e quindi non è in grado di riconoscere il linguaggio $L = \{a^n\,b^n \mid n\geq 1\}$ per il semplice fatto che non può "contare" il numero di simboli a in una parola per poterne poi operare il confronto col numero di simboli b. Per riconoscere linguaggi liberi dal contesto è allora necessario considerare dispositivi con una memoria aggiuntiva potenzialmente infinita: consideriamo qui il caso della memoria a pila (stack).

Una memoria a pila permette di memorizzare una qualsiasi parola su un alfabeto K. Sulla parola z memorizzata possiamo operare come segue:
1. E' possibile controllare se la parola z è vuota, col predicato ISEMPTY(z) che vale 1 se $z=\varepsilon$ e vale 0 se $z\neq\varepsilon$.
2. E' possibile leggere il primo simbolo in z con l'operazione TOP(z). Se $z=\varepsilon$ allora TOP(z) è indefinita, altrimenti se z è della forma ax con $a\in K$, e vale TOP(ax)=a
3. E' possibile cancellare il primo simbolo in z con l'operazione POP(z). Se $z=\varepsilon$ allora POP(z) è indefinita, altrimenti se z è della forma ax con $a\in K$, e vale POP(ax)=x
4. E' possibile aggiungere un simbolo $a\in K$ in testa a z con l'operazione PUSH(a,z) = az.

La possibilità di costruire riconoscitori per linguaggi liberi dal contesto utilizzando una memoria a pila viene ben evidenziata dalle seguenti considerazioni. Se L è un linguaggio libero dal contesto, allora può essere generato da una grammatica G in forma normale di Greibach, cioè con regole di produzione del tipo $A\rightarrow aW$, dove a è un simbolo terminale e W una parola (eventualmente vuota) di metasimboli.

Sappiamo che possiamo limitarci, senza perdita di generalità, a derivazioni in cui le regole di produzione vengono applicate al metasimbolo "più a sinistra". Applicando, a partire dall'assioma S, le regole al metasimbolo più a sinistra, poiché la grammatica è in forma normale di Greibach si derivano in generale parole della forma: $S \Rightarrow_G^* x_1\, x_2 \ldots x_m\, X_1\, X_2 \ldots X_n$, dove $x_1\, x_2 \ldots x_m$ è una parola di simboli terminali mentre $X_1\, X_2 \ldots X_n$ è una parola di metasimboli.
L'applicazione alla parola $x_1\, x_2 \ldots x_m\, X_1\, X_2 \ldots X_n$ di una regola del tipo $X_1 \rightarrow aW$ (dove W è una parola eventualmente vuota di metasimboli) porta alla parola $x_1\, x_2 \ldots x_m\, a\, W\, X_2 \ldots X_s$; si osservi che la nuova parola di metasimboli $W\, X_2 \ldots X_s$ è ottenuta dalla parola precedente $X_1\, X_2 \ldots X_s$ operando come segue:

1. Si cancella il primo metasimbolo X_1 (mediante una operazione POP)
2. Si aggiunge la parola W in testa alla pila (mediante eventuali operazioni PUSH)

Si osservi inoltre che la possibilità di applicare "più a sinistra" una eventuale regola del tipo $A \rightarrow aV$ alla parola $x_1\, x_2 \ldots x_m\, X_1\, X_2 \ldots X_n$ richiede solo di applicare a $X_1\, X_2 \ldots X_n$ l'operazione TOP: dopo aver ottenuto $X_1 = TOP(X_1\, X_2 \ldots X_n)$ basta controllare se X_1=A. Si osservi infine che una derivazione del tipo $S \Rightarrow_G^* x_1\, x_2 \ldots x_m$, che attesta che $x_1\, x_2 \ldots x_m \in L$, corrisponde al caso in cui $X_1\, X_2 \ldots X_n$ è la parola vuota.

Per le considerazioni fatte, si può costruire un riconoscitore per un linguaggio L generato da una grammatica G in forma normale di Greibach gestendo una pila. Informalmente, il riconoscitore è costituito da un nastro, che contiene la parola $w = x_1\, x_2 \ldots x_m$ da riconoscere, e da una pila, che inizialmente contiene l'assioma S. Il riconoscitore scandisce in ordine, partendo dalla prima, le lettere della parola da riconoscere e ad ogni scansione modifica il contenuto della pila. Più precisamente, se il contenuto della pila è la parola $X_1\, X_2 \ldots X_n$ e il riconoscitore legge il simbolo x_k, allora effettua la seguente mossa:

1. prende in considerazione, se ne esiste almeno una, una regola di G del tipo $X_1 \rightarrow x_k W$
2. cancella dalla pila il simbolo X_1 e pone in testa alla pila la parola W, cosicché il nuovo contenuto della pila sia $W\, X_2 \ldots X_s$.

Si osservi che il riconoscitore così costruito è in generale non deterministico, poiché, riguardo al punto 1., possono esistere più regole con lo stesso metasimbolo a sinistra e lo stesso simbolo terminale a destra. La parola w è accettata se esiste almeno una sequenza di mosse per cui il riconoscitore, dopo aver scandito tutta la parola, ha la pila vuota.

Più formalmente:
Definizione 5.1: un *riconoscitore a pila* è descritto da un sistema $\Phi = <\Sigma, K, \delta, S>$ dove;

1. Σ è un alfabeto di simboli terminali
2. K è un alfabeto disgiunto da Σ ed S è un elemento di K.
3. δ è una funzione da $\sum xK$ ai sottoinsiemi finiti di K*

L'interpretazione di $\delta(a,A)=\{W_1, \ldots, W_m\}$ è la seguente: è letto il simbolo a e il simbolo in testa alla pila è A, allora nella pila viene cancellato A e inserita in testa una parola W_k scelta in $\{W_1, \ldots, W_m\}$.
Formalmente, una *configurazione* del riconoscitore a pila è una parola X in K* (da interpretare come contenuto della pila). Se $a \in \Sigma$ e $W_k \in \delta(a,A)$, allora porremo:

$$a: AY \Rightarrow W_k Y$$

Se inoltre abbiamo che:

$$a_j: \Lambda_j \Rightarrow \Lambda_{j+1} \qquad \text{per ogni j tra 1 ed n}$$

allora porremo:

$$a_1 a_2 \ldots a_n : \Lambda_1 \Rightarrow^* \Lambda_{n+1}$$

Definizione 5.2: Il linguaggio L(Φ) riconosciuto (o accettato) dall'automa a pila Φ è il linguaggio:

$$L(\Phi) = \{w \mid w: S \Rightarrow^* \varepsilon\}$$

Vale la seguente:

Proposizione 5.1: L è acontestuale se e solo se è accettato da un riconoscitore a pila.
Dim. Se L è libero dal contesto, esiste una grammatica G = < Σ,Q,*P*,S> in forma normale di Greibach che lo genera. Utilizzando G, costruiamo il riconoscitore a pila Φ = <Σ, K, δ, S> dove:

1. K = Q
2. δ(a,A) = {W | A→ aW è una regola di produzione di G }

Si prova facilmente per induzione che $S \Rightarrow_G^* w$ se e solo se, in Φ, vale w: $S \Rightarrow^* \varepsilon$. Questo implica che L(Φ) =L
Se viceversa L è riconosciuto da un riconoscitore a pila Φ = <Σ, K, δ, S>, allora è generato dalla grammatica G = < Σ,Q,*P*,S> dove:

1. Q = K
2. A→ aW è una regola di produzione di G se e solo se W∈ δ(a,A).

Esempio 5.1:
Si consideri il linguaggio L formato dalle espressioni generate dalla grammatica con assioma E e regole di produzione:

E→(E+E) / x/ y

Possiamo trovare una grammatica in forma normale di Greibach, equivalente alla precedente. Tale grammatica ha assioma E e regole di produzione:

E→**(**EAEB , A→+ , B→ **)** , E→x , E→y

Da questa grammatica otteniamo l'automa riconoscitore Φ = <{**(** , **)** , x , y , +}, {E,A,B}, δ, E>, dove:
δ(**(** , E) = EAEB , δ(+ , A) = δ(**)** , B) = δ(x , E) =δ(y , E) = ε .
La funzione δ è indefinita in ogni altro caso.
Applicando ad esempio il riconoscitore alla parola (x+(y+x)), la parola è accettata perchè:
(: E ⇒ EAEB
x : EAEB ⇒ AEB
+ : AEB ⇒ EB
(: EB ⇒ EAEBB
y : EAEBB ⇒ AEBB
+ : AEBB ⇒ EBB
x : EBB ⇒ BB
) : BB ⇒ B
) : B ⇒ ε