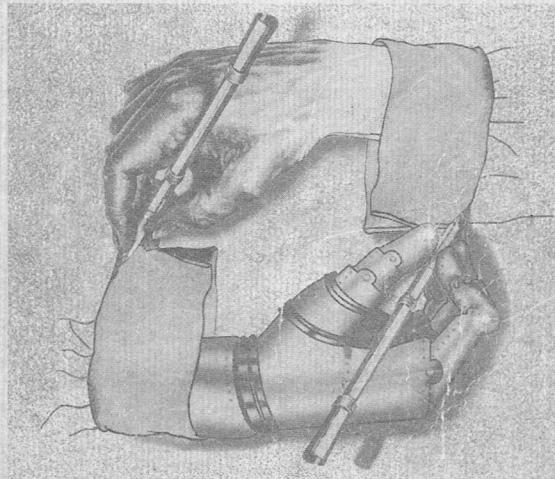


GIOVANNI SEMERARO



Appunti di  
**TEORIA DEI LINGUAGGI FORMALI**



ADRIATICA EDITRICE - BARI

GIOVANNI SEMERARO

**Appunti di  
TEORIA DEI LINGUAGGI FORMALI**

ADRIATICA EDITRICE - BARI 1996

## Indice

<b>Prefazione .....</b>	<b>vii</b>
<b>- 1. Introduzione. ....</b>	<b>1</b>
<b>- 2. Grammatiche e linguaggi. ....</b>	<b>13</b>
2.1 Linguaggi formali e monoidi liberi. ....	13
2.2 Generazione e riconoscimento di linguaggi formali. ....	18
2.3 Grammatiche generative. ....	22
2.4 Derivazioni. ....	24
2.5 Linguaggi generati da grammatiche e correttezza di una grammatica. ....	28
2.6 Esercizi. ....	32
<b>3. Linguaggi liberi da contesto e     linguaggi dipendenti da contesto. ....</b>	<b>59</b>
3.1 Grammatiche e linguaggi liberi da contesto. ....	59
3.2 Grammatiche e linguaggi dipendenti da contesto. ....	60
3.3 Relazione tra linguaggi C.F. e C.S. ....	61
3.4 Grammatiche e linguaggi monotoni. ....	62
<b>4. Linguaggi liberi da contesto. ....</b>	<b>71</b>
4.1 Alberi di derivazione. ....	71
4.2 Princípio di sostituzione di sottoalberi. ....	75
4.3 Pumping Lemma per i linguaggi liberi da contesto. ....	82
4.4 Ambiguità di grammatiche e linguaggi. ....	85
4.5 Esercizi. ....	90

<b>5. Grammatiche e macchine.</b> .....	<b>105</b>
5.1 Classificazione delle grammatiche secondo Chomsky. ....	105
5.2 Teorema della Gerarchia. ....	106
5.3 Lemma della stringa vuota. ....	108
5.4 Operazioni sui linguaggi. ....	109
5.5 Proprietà di chiusura delle classi di linguaggi. ....	112
5.6 L'operazione di riflessione. ....	126
5.7 Esercizi. ....	128
<b>6. Automi a stati finiti (deterministici e non deterministici).</b> .....	<b>137</b>
6.1 Automi a stati finiti deterministici. ....	137
6.2 Linguaggi a stati finiti. ....	141
6.3 Automi a stati finiti non deterministici. ....	142
6.4 Linguaggi accettati da automi non deterministici. ....	146
6.5 Equivalenza delle classi di linguaggi accettati da automi a stati finiti deterministici e non deterministici. ....	146
6.6 Esercizi. ....	150
<b>7. Linguaggi regolari, espressioni regolari e teorema di Kleene.</b> .	<b>157</b>
7.1 Linguaggi regolari ed espressioni regolari. ....	157
7.2 Proprietà delle espressioni regolari. ....	161
7.3 Teorema di Kleene. ....	165
7.4 Pumping Lemma per i linguaggi regolari. ....	173
7.5 Esercizi. ....	175
<b>8. Automi a pila e grammatiche libere.</b> .....	<b>201</b>
8.1 Automi a pila. ....	201
8.2 Linguaggi accettati da automi a pila. ....	206
8.3 Esempi di linguaggi riconosciuti da automi a pila. ....	208
8.4 Forme normali per grammatiche non contestuali. ....	216
<b>Bibliografia .....</b>	<b>237</b>

## Prefazione

Questo lavoro è il risultato di un'opera di riorganizzazione del materiale didattico utilizzato nella prima parte dei corsi universitari di Linguaggi Formali e Compilatori (Corso di Laurea in Scienze dell'Informazione) e di Linguaggi e Traduttori (Corso di Laurea in Informatica).

Sono grato a tutti gli studenti dell'Università di Bari che, con le loro osservazioni e le loro domande, hanno contribuito al miglioramento dei contenuti del testo, ed in particolare a Dario Macchitella per la trasposizione in formato digitale delle dispense manoscritte dei corsi.

Desidero ringraziare il Dott. Cesare Daniele Antifora ed il Dott. Gioacchino De Gennaro per aver letto attentamente e commentato argutamente le numerose revisioni del testo.

Un particolare ringraziamento va infine alla Prof.ssa Floriana Esposito per il costante incitamento e le proficue discussioni. Senza il suo apporto, la realizzazione di questo volume non sarebbe stata possibile.

Questo lavoro è dedicato a Libero Grassi e Carlo Palermo.

Giovanni Semeraro

## 1. Introduzione.

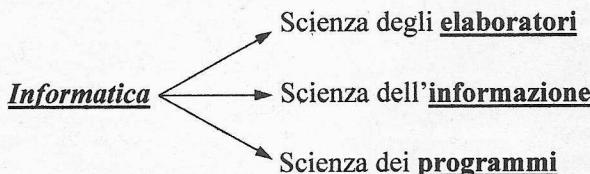
La teoria dei linguaggi formali rappresenta spesso uno scoglio per gli studenti di informatica a causa della sua forte dipendenza dalla notazione.

L'argomento, d'altra parte, non può essere evitato perché ogni laureato in informatica deve possedere una buona comprensione dell'operazione di compilazione e questa, a sua volta, si fonda pesantemente sulla teoria dei linguaggi formali.

Questo testo ha l'intenzione di fornire una base sufficiente alla comprensione della teoria dei linguaggi formali.

In particolare, ci concentreremo quasi esclusivamente su due tipi di grammatiche: regolari e libere da contesto, poiché i linguaggi formali utilizzati in informatica sono per lo più di questi tipi.

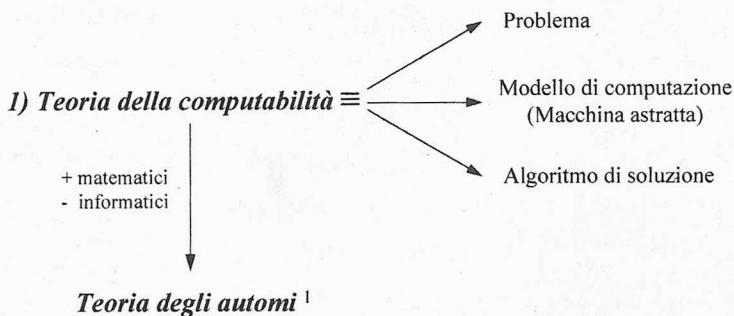
L'informatica teorica è la scienza degli algoritmi, in tutti i loro aspetti, poiché è il concetto di algoritmo che è in qualche modo comune a tutti i settori dell'informatica.



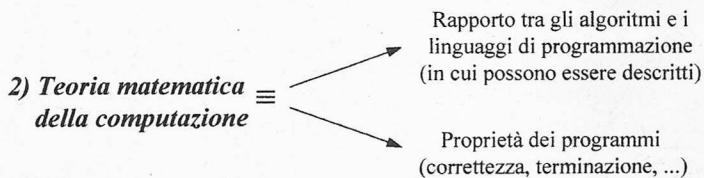
<u>elaboratori</u>	≡ macchine che eseguono gli algoritmi
<u>informazione</u>	≡ materia su cui lavorano gli algoritmi
<u>programmi</u>	≡ algoritmi descritti in un particolare linguaggio

## Aree di ricerca dell'informatica teorica

Lo schema riportato in Figura 1.1 fornisce una panoramica delle aree di ricerca che ricadono nell'ambito più generale dell'informatica teorica. Tali aree sono:

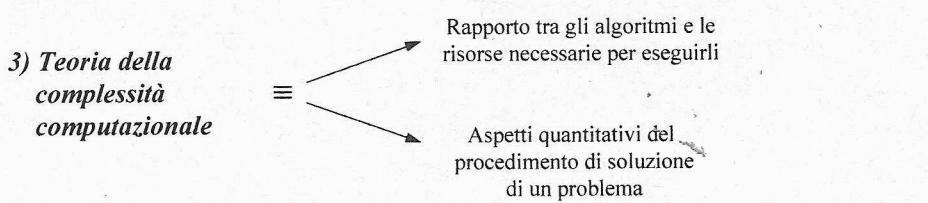


Particolarmente feconda si è rivelata l'interazione tra la **Teoria degli Automi** e la **Teoria dei Linguaggi Formali**. Quest'ultima, nota in ambito linguistico per caratterizzare i linguaggi naturali, è stata sviluppata ed utilizzata dagli informatici teorici, che avevano l'esigenza di descrivere gli algoritmi con linguaggi di programmazione comprensibili dalla macchina, ma non troppo difficili per l'uomo e soprattutto non ambigui.



---

<sup>1</sup> La teoria degli automi si occupa della definizione di diverse classi di dispositivi di calcolo e, per ogni classe, dello studio delle proprietà e delle classi di problemi che sono in grado di risolvere.



Quando iniziamo a studiare un linguaggio, formale o meno, tutti quanti godiamo di un grande vantaggio: siamo esperti in un linguaggio, quello con cui comuniciamo con gli altri. Qualunque sia la nostra lingua madre, l’italiano, l’inglese, o altro, abbiamo sviluppato una competenza considerevole e continueremo a svilupparla per il resto della nostra vita.

Oltre ad essere competenti in uno o più linguaggi naturali, lo studente in informatica ha familiarità con diversi linguaggi di programmazione, come il FORTRAN, il PASCAL, il C, il PROLOG, ...

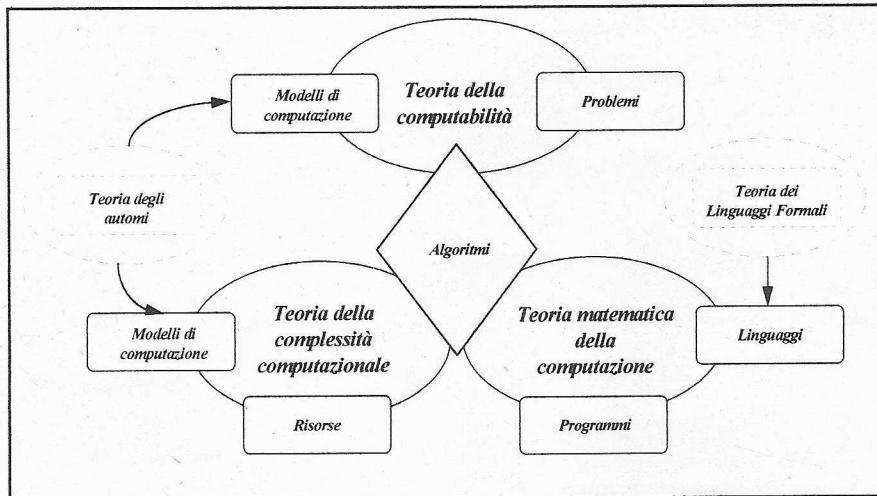


Figura 1.1

Questi linguaggi sono usati per scrivere programmi e quindi per comunicare con il computer. Senza dubbio, chiunque utilizzi un computer potrà notare che esso è particolarmente limitato nel comprendere il significato desiderato dei programmi. Mentre in linguaggio naturale possiamo di solito comunicare anche attraverso frasi mal strutturate e spesso incomplete, quando dobbiamo comunicare con un computer la situazione cambia.

I nostri programmi devono aderire rigorosamente a regole stringenti e anche differenze minori vengono rigettate come errate.

Idealmente, ogni frase in un linguaggio dovrebbe essere corretta sia *semanticamente* (avere cioè il corretto significato) sia *sintatticamente* (avere la corretta struttura grammaticale).

Nell'italiano parlato, le frasi sono spesso sintatticamente sbagliate, ciononostante convogliano la semantica desiderata.

In programmazione, comunque, è essenziale che la sintassi sia corretta al fine di comunicare una qualsivoglia semantica. Quando studiamo la semantica di una frase, ne stiamo studiando il significato. Tutte le frasi che seguono hanno la stessa interpretazione semantica, cioè lo stesso significato, sebbene siano sintatticamente differenti:

*The man hits the dog*

*The dog is hit by the man*

*L'homme frappe le chien*

Lo studio della sintassi è lo studio della grammatica, cioè della struttura delle frasi. La frase:

*The man hits the dog*

può essere analizzata sintatticamente, cioè risolta nelle parti grammaticali componenti, come segue:



ed ogni frase in questa forma è sintatticamente valida in inglese.

Possiamo descrivere un particolare insieme di tali frasi semplici in inglese usando le seguenti regole:

$<\text{frase semplice}> ::= <\text{parte nominale}> <\text{parte verbale}>$   
 $\quad\quad\quad <\text{parte nominale}>$

$<\text{parte nominale}> ::= <\text{articolo}> <\text{nome}>$

$<\text{nome}> ::= \text{car} \mid \text{man} \mid \text{dog}$

$<\text{articolo}> ::= \text{The} \mid \text{a}$

$<\text{parte verbale}> ::= \text{hits} \mid \text{eats}$

Queste regole sono scritte in BNF,<sup>2</sup> una notazione usata comunemente per descrivere la sintassi dei linguaggi di programmazione.

Nell'esempio,  $<\text{frase semplice}>$  è definita come una  $<\text{parte nominale}>$  seguita da una  $<\text{parte verbale}>$  seguita, a sua volta, da un'altra  $<\text{parte nominale}>$ .

Ciascuna delle due occorrenze di  $<\text{parte nominale}>$  deve essere espansa in  $<\text{articolo}>$  seguito da un  $<\text{nome}>$ .

Scegliendo di espandere la prima occorrenza di  $<\text{articolo}>$  in *The*, la prima occorrenza di  $<\text{nome}>$  in *man*, la  $<\text{parte verbale}>$  in *hits*, il secondo  $<\text{articolo}>$  in *The* ed infine l'ultimo  $<\text{nome}>$  in *dog*, si dimostra che la frase

*The man hits the dog*

è una delle nostre frasi semplici.

<sup>2</sup> BNF (Backus Naur Form) è un metalinguaggio.

Tutto ciò si può riassumere nel seguente albero di derivazione (Figura 1.2).

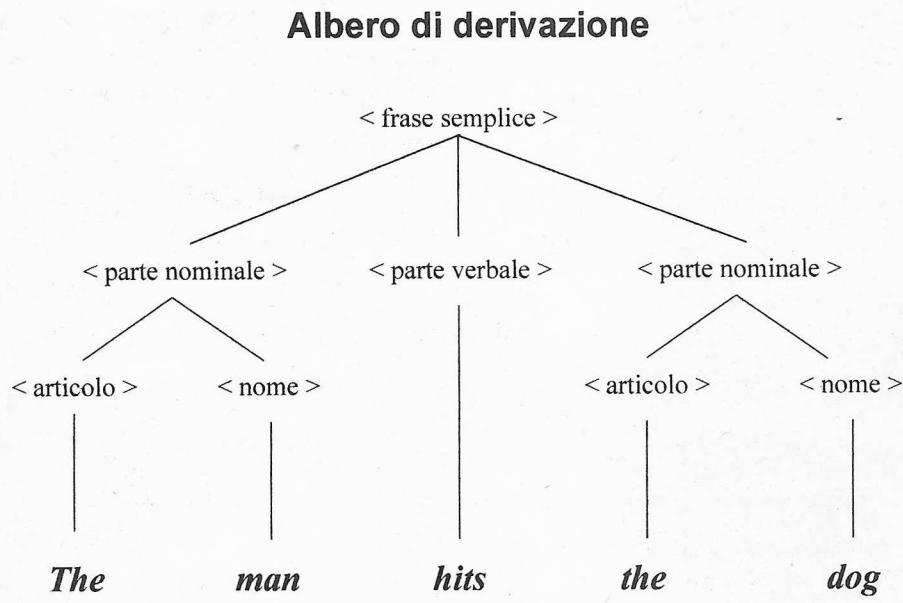


Figura 1.2

La definizione data di *< frase semplice >* dà luogo a 72 diverse frasi derivabili. Sebbene tutte siano sintatticamente corrette, in base alla nostra definizione, alcune non hanno un'interpretazione semantica sensata (non hanno senso compiuto).

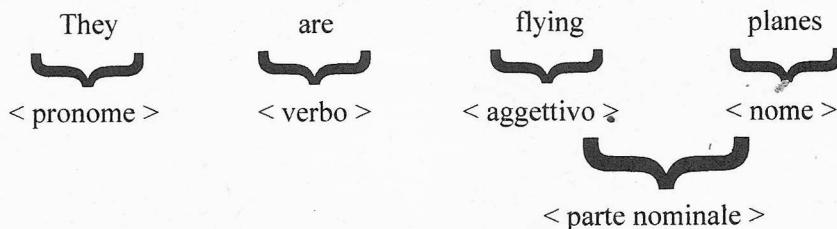
Una di queste è la frase:

*The car eats the man*

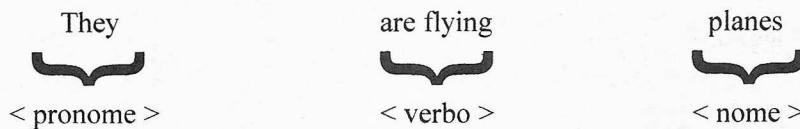
Non necessariamente una frase corretta ha senso. Consideriamo la seguente frase (non derivabile da *<frase semplice>* utilizzando le regole date in precedenza):

### *They are flying planes*

Un'analisi sintattica di questa frase è:



Quest'analisi suggerisce che *they* si riferisce a *planes* (*aerei nel cielo che volano*). Un'analisi alternativa sarebbe



e questa implicherebbe un'interpretazione semantica completamente differente, in cui *they* si riferisce a chiunque sia attualmente al controllo dei *planes* (*essi stanno pilotando gli aerei*).

Nell'esempio, l'analisi sintattica è di aiuto all'interpretazione semantica. Per quanto sia scelto "ad hoc" e mostri un fenomeno poco comune nel linguaggio naturale, l'esempio è illustrativo di un concetto importante in computazione.

Una fase cruciale nel processo di compilazione di un programma è l'analisi sintattica, come passo essenziale per la sua interpretazione semantica.

## ***Teoria dei Linguaggi Formali***

Negli anni '50, N. Chomsky, linguista americano (1928 - ) cerca di descrivere la sintassi del linguaggio naturale secondo semplici regole di riscrittura e trasformazione. Chomsky considera alcune restrizioni sulle regole sintattiche e classifica i linguaggi in base alle restrizioni imposte alle regole che generano tali linguaggi. Una classe importante di regole che generano linguaggi formali va sotto il nome di *grammatiche libere da contesto* (prive di contesto) o *Context-free grammars* (C.F.).

L'importanza di tale classe (e dei linguaggi da essa generati) risiede nel fatto che lo sviluppo dei primi linguaggi di programmazione di alto livello (ALGOL 60), che segue di pochi anni il lavoro di Chomsky, dimostra che le grammatiche C.F. sono strumenti adeguati a descrivere la sintassi di base di molti linguaggi di programmazione.

### **Esempio 1.1 (Linguaggio C.F.)**

Un linguaggio C.F. è il linguaggio delle parentesi ben formate. Tale linguaggio comprende tutte le stringhe di parentesi aperte e chiuse bilanciate correttamente. Ad esempio,

( ) è ben formata;

(( )) è ben formata;

(( )) non è ben formata.

Questo linguaggio gioca un ruolo fondamentale in informatica come notazione per contrassegnare il "raggio d'azione" nelle espressioni matematiche e nei linguaggi di programmazione (in cui spesso si usano *begin* ed *end* - PASCAL, ALGOL 60 - oppure { e } - C - al posto di "(" e ")" , rispettivamente).

( )  
A      A  
A

*S 0  
S(S)  
S-S*

### Definizione 1.1 (Linguaggio delle parentesi ben formate)

- i) La stringa  $( )$  è ben formata;
- ii) se la stringa di simboli  $A$  è ben formata, allora lo è anche  $(A)$ ;
- iii) se le stringhe  $A$  e  $B$  sono ben formate, allora lo è anche la stringa  $AB$ . ■

In corrispondenza di questa definizione induttiva, possiamo considerare un sistema di riscrittura che genera esattamente l'insieme delle stringhe lecite di parentesi ben formate:

(1)  $S \rightarrow ( )$

(2)  $S \rightarrow (S)$

(3)  $S \rightarrow SS$

Le regole di riscrittura (1), (2) e (3) sono dette *produzioni* o *regole di produzione*. Esse stabiliscono che “data una stringa, si può formare una nuova stringa sostituendo una  $S$  o con  $( )$  o con  $(S)$  o con  $SS$ ”.

Se confrontiamo la definizione di parentesi ben formate e le tre produzioni, si osserva una corrispondenza diretta tra le parti i) e ii) della definizione e le produzioni (1) e (2). A differenza delle prime due, la produzione (3) è apparentemente diversa dalla parte (iii) della definizione induttiva data in precedenza. Difatti, abbiamo utilizzato simboli distinti  $A$  e  $B$  nella definizione induttiva per evidenziare il fatto che le due stringhe di parentesi ben formate che consideriamo non sono necessariamente uguali. Nella produzione corrispondente non c'è necessità di usare simboli distinti perché, in  $S \rightarrow SS$ , le due  $S$  che compaiono a destra possono essere sostituite indipendentemente.

Possiamo cioè applicare una produzione ad una delle due  $S$  (alla prima o alla

seconda) indifferentemente. Il risultato non cambia se consideriamo prima la  $S$  a sinistra e poi quella a destra o viceversa.

### Esempio 1.2 (Generazione di $(( ))(( )( ))$ )

Primo modo:

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S \xrightarrow{(1)} (( ))S \xrightarrow{(2)} (( ))(S) \xrightarrow{(3)} (( ))(SS) \xrightarrow{(1)} (( ))(( )S) \xrightarrow{(1)} (( ))(( )( ))$$

La corrispondente sequenza di applicazione delle produzioni è:

$$(3) \rightarrow (2) \rightarrow (1) \rightarrow (2) \rightarrow (3) \rightarrow (1) \rightarrow (1)$$

Secondo modo:

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} S(S) \xrightarrow{(3)} S(SS) \xrightarrow{(1)} S(( )S) \xrightarrow{(1)} S(( )( )) \xrightarrow{(2)} (S)(( )( )) \xrightarrow{(1)} (( ))(( )( ))$$

La corrispondente sequenza di applicazione delle produzioni è:

$$(3) \rightarrow (2) \rightarrow (3) \rightarrow (1) \rightarrow (1) \rightarrow (2) \rightarrow (1)$$

Una sequenza di applicazioni di regole di produzione prende il nome di *derivazione*. 

### Notazione

Nell'esempio 1.2 abbiamo fatto uso della notazione  $\alpha \xrightarrow{(n)} \beta$ . La sua interpretazione è la seguente: "da  $\alpha$  si produce direttamente  $\beta$  per effetto dell'applicazione della regola di riscrittura  $n$ ". Ad esempio,

$$SS \xrightarrow{(2)} (S)S$$

si legge: "SS produce direttamente  $(S)S$  per effetto dell'applicazione della regola di riscrittura (2)".

Le derivazioni precedenti (modi 1 e 2) vengono riassunte attraverso la notazione:

$$S \Rightarrow (( ))(( )( ))$$



che leggiamo come: "S produce (( ))(( )( ))".

Nelle regole di produzione si è fatto uso di due tipi di simboli: caratteri che possono apparire nelle derivazioni, ma non nelle stringhe finali, detti *simboli nonterminali* o *variabili* (nell'esempio,  $S$  è il solo nonterminale) e caratteri che possono apparire nelle stringhe finali, detti *simboli terminali* (nell'esempio, "(" e ")" sono i simboli terminali).

In BNF si usa la seguente notazione:

$$\begin{array}{ll} S & < S > \\ \Rightarrow & :: = \end{array}$$

## 2. Grammatiche e linguaggi.

### 2.1 Linguaggi formali e monoidi liberi.

Il concetto di linguaggio formale è strettamente correlato a quello di *monoide libero* (generato da un insieme).



#### Definizione 2.1 (Alfabeto)

Un insieme  $X$  finito e non vuoto è un *alfabeto*.



#### Esempio 2.1

- L'alfabeto latino, con l'aggiunta dei simboli di interpunkzione e dello spazio bianco:  $a\ b\ c\ \dots\ z\ ;\ ,\ :\ \dots$
- L'insieme delle dieci cifre arabe:  $0\ 1\ \dots\ 9$

Con i simboli primitivi dell'alfabeto si formano le parole (es.:  $abc$ , 127, *casa*,...).

#### Definizione 2.2 (Parola o stringa)



Una sequenza finita di simboli  $x_1x_2\dots x_n$ , dove ogni  $x_i$  è preso da uno stesso alfabeto  $X$  è una *parola* (su  $X$ ).

#### Esempio 2.2

$$X = \{0,1\}.$$

001110 è una parola su  $X$ .

Una parola è ottenuta giustapponendo o concatenando simboli (caratteri) dell'alfabeto.

Se una stringa ha  $m$  simboli (non necessariamente distinti) allora diciamo che ha lunghezza  $m$ .

La lunghezza di una stringa  $w$  è denotata con  $|w|$ . Le parole di lunghezza 1 sono i simboli di  $X$ .

Quindi 001110 è una parola di lunghezza 6:

$$|001110| = 6$$

La parola vuota (o stringa vuota), denotata con  $\lambda$ , è una stringa priva di simboli ed ha lunghezza 0:

$$|\lambda| = 0.$$

### Definizione 2.3 (Uguaglianza tra stringhe)

Due stringhe sono *uguali* se i loro caratteri, letti ordinatamente da sinistra a destra, coincidono. ■

### Definizione 2.4 ( $X^*$ )



L'insieme di tutte le stringhe di lunghezza finita sull'alfabeto  $X$  si denota con  $X^*$ . ■

### Esempio 2.3

Se  $X = \{0,1\}$ , allora  $X^* = \{\lambda, 0, 1, 00, 01, 10, 11, \dots\}$ .

$X^*$  ha un numero di elementi che è un infinito numerabile. Dalla definizione, segue che  $\lambda \in X^*$ , per ogni insieme  $X$ .

### Definizione 2.5 (Concatenazione o prodotto)

Sia  $\alpha \in X^*$  una stringa di lunghezza  $m$  e  $\beta \in X^*$  una stringa di lunghezza  $n$ , la *concatenazione* di  $\alpha$  e  $\beta$ , denotata con  $\alpha\beta$  o  $\alpha \cdot \beta$ , è definita come la

stringa di lunghezza  $m+n$ , i cui primi  $m$  simboli costituiscono una stringa uguale a  $\alpha$  ed i cui ultimi  $n$  simboli costituiscono una stringa uguale a  $\beta$ .

Quindi se  $\alpha = x_1x_2\dots x_m$  e  $\beta = x'_1x'_2\dots x'_n$ , si ha:  $\alpha\beta = x_1x_2\dots x_m x'_1x'_2\dots x'_n$ .

La concatenazione di stringhe su  $X$  è un'operazione binaria su  $X^*$ :

$$\cdot : X^* \times X^* \rightarrow X^*$$

- è associativa:  $(\alpha\beta)\gamma = \alpha(\beta\gamma) = \alpha\beta\gamma$ ,  $\forall \alpha, \beta, \gamma \in X^*$ ;
- non è commutativa:  $\exists \alpha, \beta \in X^* : \alpha\beta \neq \beta\alpha$ ;
- ha elemento neutro  $\lambda$ :  $\lambda\alpha = \alpha\lambda = \alpha$ ,  $\forall \alpha \in X^*$ .

Dunque  $(X^*, \cdot)$  è un monoide (non commutativo). 

■

### Osservazione 2.1

In base alla definizione di prodotto, ogni parola non vuota  $\alpha = x_1x_2\dots x_n$  si può scrivere in uno ed un solo modo come prodotto di parole di lunghezza 1, cioè di elementi di  $X$ .

Ciò si esprime dicendo che:

$(X^*, \cdot)$  è il monoide libero generato dall'insieme  $X$ .

### Definizioni 2.6 (Prefisso, suffisso, sottestringa)

Se  $\gamma \in X^*$  è della forma  $\gamma = \alpha\beta$ , ove  $\alpha, \beta \in X^*$ , allora  $\alpha$  è un *prefisso* di  $\gamma$  e  $\beta$  è un *suffisso* di  $\gamma$ .

Se   $\beta \in X^*$  e  $\delta$  è della forma  $\delta = \alpha\beta\gamma$ , ove  $\alpha, \gamma \in X^*$ , allora  $\beta$  è una *sottestringa* di  $\delta$ . 

### Esempio 2.4

Sia  $\gamma = 00110$ . Allora  $\{\lambda, 0, 00, 001, 0011, \gamma\}$  è l'insieme dei prefissi di  $\gamma$ ,  $\{\lambda, 0, 10, 110, 0110, \gamma\}$  è l'insieme dei suffissi di  $\gamma$  e  $\{\lambda, 0, 1, 00, 01, 001, 011, 110, 0011, 0110, \gamma\}$  è l'insieme delle sottostringhe di  $\gamma$ .

### Definizione 2.7 (Potenza di una stringa)

Data una stringa  $\alpha$  su  $X$ , la *potenza h-esima* di  $\alpha$  è definita (induttivamente) come segue:

$$\alpha^h = \begin{cases} \lambda & \text{se } h = 0 \\ \alpha\alpha^{h-1} & \text{altrimenti,} \end{cases}$$

con  $h = 0, 1, 2, \dots$  ■

Dunque, la potenza  $h$ -esima di una stringa è un caso speciale di concatenamento (in quanto la si ottiene concatenando una stringa  $h$  volte con se stessa).

### Definizione 2.8 (Potenza di un alfabeto)

Sia  $X$  un alfabeto, poniamo:

1)  $X^1 = X$ ;

2)  $X^2 = \{x_1x_2 \mid x_1, x_2 \in X, x_1x_2 \equiv x_1 \cdot x_2\}$

3)  $X^3 = \{x_1x_2x_3 \mid x_1x_2 \in X^2, x_3 \in X, x_1x_2x_3 \equiv x_1x_2 \cdot x_3\}$

...) .....

i)  $X^i = \{x_1x_2\dots x_{i-1}x_i \mid x_1x_2\dots x_{i-1} \in X^{i-1}, x_i \in X, x_1x_2\dots x_i \equiv x_1x_2\dots x_{i-1} \cdot x_i\}$

Se  $i \geq 1$  si ha:

$$X^+ = X \cup X^2 \cup \dots \cup X^i \cup \dots = \bigcup_{i=1}^{+\infty} X^i$$

Se  $\lambda$  è la parola vuota e prendiamo un  $w \in X^+$  tale che  $w \cdot \lambda = \lambda \cdot w = w$  si ha:

$$X^* = \{\lambda\} \cup X^+$$

Inoltre si ha:

$$X^h = \begin{cases} \{\lambda\} & \text{se } h = 0 \\ X \cdot X^{h-1} & \text{altrimenti.} \end{cases}$$

### Definizione 2.9 (Linguaggio Formale)

Un *linguaggio formale*  $L$  su un alfabeto  $X$  è un sottoinsieme di  $X^*$ .

$$L \subseteq X^*$$

### Esempio 2.5

Il linguaggio delle parentesi ben formate è un linguaggio formale in quanto, denotato con  $M$  tale linguaggio, si ha:

$$M \subset \{( , )\}^*$$

I linguaggi formali possono essere di natura molto diversa l'uno dall'altro.

### Esempio 2.6

Un linguaggio di programmazione può essere costruito a partire dall'alfabeto  $X$  dei simboli sulla tastiera.

L'insieme, finito o infinito, dei programmi ben costruiti sintatticamente (ossia, che rispettano la sintassi) costituisce un linguaggio.

### **Esempio 2.7**

Consideriamo l'insieme dei teoremi di una teoria matematica. I teoremi sono particolari stringhe di simboli del nostro alfabeto. L'insieme dei teoremi "ben formati" rappresenta un linguaggio. Ad esempio, la stringa "ab=ba" non è un teorema della teoria dei gruppi, ma della teoria dei gruppi abeliani.

## **2.2 Generazione e riconoscimento di linguaggi formali.**

A noi interessano i linguaggi formali da almeno due *punti di vista*

### **1) Descrittivo/Generativo**

Come possiamo *generare* gli elementi di un dato linguaggio  $L$ ?

Un linguaggio finito può essere descritto/generato per elencazione dei suoi elementi (se il numero dei suoi elementi non è troppo grande).

Un linguaggio infinito non è elencabile. I linguaggi infiniti sono i più interessanti perché devono essere specificati necessariamente attraverso una *proprietà* che ne caratterizza gli elementi, che ne definisce l'intensione. Tale proprietà può essere vista come una regola da seguire per generare gli elementi del linguaggio.

Il vero problema è trovare la(e) regola(e) generativa(e) (di produzione) di un linguaggio. È quello che accade quando si impara un linguaggio: non è possibile memorizzare tutte le frasi del linguaggio.

### **Esempio 2.8**

Non è possibile "elencare" tutti i teoremi della teoria dei gruppi, perché sono infiniti i teoremi realizzabili combinando quelli noti.

Un libro di teoria dei gruppi non è l'elencazione dei teoremi, ma fornisce una serie di assiomi e le regole con le quali, a partire dagli assiomi, è possibile costruire tutti i teoremi della teoria dei gruppi.

Per descrivere la regola di produzione di un linguaggio, utilizzeremo una notazione insiemistica.

### Esempio 2.9

Sia  $L$  il linguaggio su  $X = \{0\}$  costituito da tutte e sole le stringhe che hanno un numero pari di 0, cioè:  $L = \{\lambda, 00, 0000, 000000, \dots\}$ .

### 2) Riconoscitivo

Come possiamo *riconoscere* gli elementi di un dato linguaggio  $L$ ?

Questo secondo punto di vista ha come obiettivo la costruzione di "macchine" in grado di decidere/stabilire se una stringa è un elemento di  $L$  oppure no. Si intende costruire una "macchinetta" (Figura 2.1) cui dare in ingresso una particolare parola e che produce una tra due possibili risposte:

$sì \equiv' \in L'$  e  $no \equiv' \notin L'$

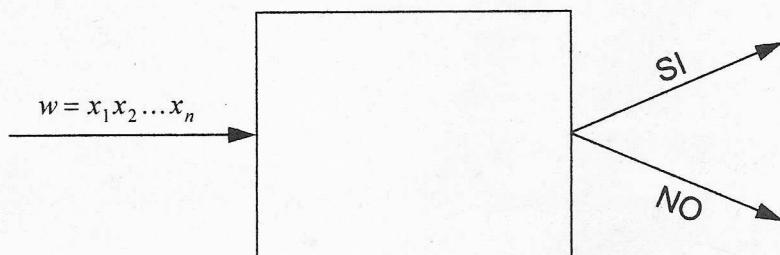


Figura 2.1

### Esempio 2.10

L'esecuzione di un programma errato sintatticamente viene inibita. Questo è indice dell'esistenza di una “macchinetta” che stabilisce se il programma appartiene o no all'insieme dei programmi sintatticamente ben costruiti.

Analizziamo il problema della generazione di  $L$ .

### Esempio 2.11

Sia dato l'alfabeto:

$$X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}.$$

Voglio generare il linguaggio  $L$  dei numeri interi relativi. Ovviamente

$$L \subseteq X^*$$

Più precisamente,   $X^*$  poiché, ad esempio,   $1 + + -$    $L$ .

Non possiamo elencare gli elementi di  $L$ . Cerchiamo dunque una serie di regole mediante le quali è possibile produrre tutti e soli gli elementi di  $L$ .

Assumiamo, per semplicità, che un numero relativo sia costituito da una serie di cifre precedute da  $+$  o  $-$ .

Adottiamo la BNF per descrivere le produzioni:

$$\begin{aligned} < S > & ::= + < I > \mid - < I > \\ < I > & ::= < D > \mid < I > < D > \\ < D > & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

Queste regole generano tutti gli interi relativi purché partiamo dal simbolo nonterminale  $S$ .

$I$  è il simbolo nonterminale (da ora in poi, talvolta abbreviato in  $NT$ ), anche detto *categoria sintattica*, che sta ad indicare (e da cui si genera) la classe dei numeri interi.

$I$  è definito ricorsivamente o come una cifra oppure come un intero seguito da una cifra.

Ogni intero relativo è generato da queste regole e niente che non sia un intero relativo può essere generato da queste regole.

### *Generazione ad albero*

Proviamo a generare l'intero relativo  $-375$ :

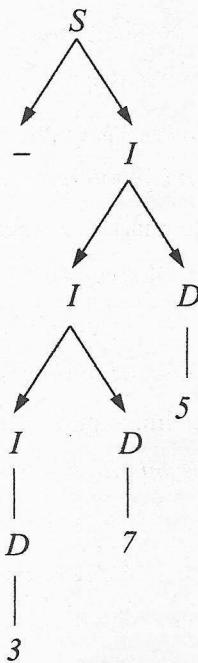


Figura 2.2

Tale albero prende il nome di *albero di derivazione*.

$$S \stackrel{*}{\Rightarrow} -375 \Leftrightarrow -375 \in L$$

Nella notazione vista per il linguaggio delle parentesi ben formate, tipica per i linguaggi formali, la grammatica diventa:

$$S \rightarrow + I | - I$$

$$I \rightarrow D | ID$$

$$D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

### 2.3 Grammatiche generative.

Dagli esempi di linguaggi visti, possiamo trarre le seguenti conclusioni.

Per generare un linguaggio sono necessari:

- un insieme  $X$  di simboli primitivi con cui si formano le parole del linguaggio, detto *alfabeto dei simboli terminali* o *alfabeto terminale*;
- un insieme  $V$  di simboli ausiliari o variabili con cui si identificano le categorie sintattiche del linguaggio, detto *alfabeto dei simboli nonterminali (ausiliari)* o *alfabeto nonterminale* o *alfabeto delle variabili*;
- un simbolo speciale  $S$ , scelto tra i nonterminali, da cui far partire la generazione delle parole del linguaggio. Tale simbolo è detto *assioma* o *scopo* o *simbolo distintivo* o *simbolo di partenza* o *simbolo iniziale*;
- un insieme  $P$  di produzioni, espresse in un formalismo quali regole di riscrittura ( $\alpha \rightarrow \beta$ ), BNF ( $\alpha ::= \beta$ ), carte sintattiche, ...

Possiamo, a questo punto, dare la definizione formale di grammatica (generativa) o grammatica a struttura di frase, come strumento per la generazione di un linguaggio.

#### Definizione 2.10 (Grammatica generativa o a struttura di frase)

Una grammatica generativa o a struttura di frase  $G$  è una quadrupla

$$G = (X, V, S, P)$$

ove:

- (1)  $X$  è l'alfabeto terminale per la grammatica;
  - (2)  $V$  è l'alfabeto nonterminale o delle variabili per la grammatica;
  - (3)  $S$  è il simbolo di partenza per la grammatica;
  - (4)  $P$  è l'insieme delle produzioni della grammatica.
- ed inoltre valgono le seguenti condizioni:

$$X \cap V = \emptyset \text{ e } S \in V$$

### Definizione 2.11 (Produzione)

Una produzione è una coppia  $(v, w)$ ,

ove  $v \in (X \cup V)^+$  e  $v$  contiene un NT

$w \in (X \cup V)^*$  ( $w$  può essere anche  $\lambda$ ).

Un elemento  $(v, w)$  di  $P$  viene comunemente scritto nella forma:

$$v \rightarrow w$$

Una produzione deve, in qualche modo, riscrivere un NT.

Per convenzione, gli elementi di  $X$  sono rappresentati di solito con lettere minuscole (con o senza pedici e di solito sono le prime lettere dell'alfabeto) o cifre ed operatori (connettivi), mentre gli elementi di  $V$  sono rappresentati con lettere maiuscole (con o senza pedici) o con stringhe delimitate dalle parentesi angolari "<" e ">".

La notazione

$$\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_k$$

è impiegata come abbreviazione della seguente:

$$\begin{array}{c} \overbrace{\alpha \rightarrow \beta_1} \\ \overbrace{\alpha \rightarrow \beta_2} \\ \vdots \\ \overbrace{\alpha \rightarrow \beta_k} \end{array}$$

### Esempio 2.12

- La grammatica per il linguaggio delle parentesi ben formate:

$$G_1 = \left( \{(), \{S\}, S, \{S \rightarrow ( ), S \rightarrow (S), S \rightarrow SS\} \right)$$

- La grammatica per il linguaggio dei numeri interi relativi:

$$G_2 = \left( \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}, \{S, I, D\}, S, \{S \rightarrow +I, S \rightarrow -I, I \rightarrow D, I \rightarrow ID, D \rightarrow 0, D \rightarrow 1, \dots, D \rightarrow 9\} \right)$$

## 2.4 Derivazioni.

### Definizioni 2.12 (Derivazione o produzione diretta, derivazione)

Sia  $G = (X, V, S, P)$  una grammatica e siano  $y$  e  $z$  due stringhe finite di simboli in  $X \cup V$  (stringhe di terminali e nonterminali) tali che:

$$y = \gamma \alpha \delta \text{ e } z = \gamma \beta \delta, \text{ ove } y \in (X \cup V)^+, z \in (X \cup V)^*, \alpha, \beta, \gamma, \delta \in (X \cup V)^*$$

- 1) Scriviamo



$$y \Rightarrow z$$

e diciamo che  $y$  produce direttamente  $z$  o che  $z$  è derivata direttamente da  $y$  se:

$$\alpha \rightarrow \beta \in P,$$

ossia se esiste in  $G$  una produzione  $\alpha \rightarrow \beta$ .

2) Scriviamo

$$y \xrightarrow{*} z$$

e diciamo che  $y$  produce  $z$  o che  $z$  è derivabile da  $y$  se  $y=z$  o esiste una sequenza di stringhe  $w_1, w_2, \dots, w_n$ , con  $w_1, w_2, \dots, w_{n-1} \in (X \cup V)^+$ ,  $w_n \in (X \cup V)^*$ ,  $w_1 = y$  e  $w_n = z$  tali che  $\forall i, i = 1, 2, \dots, n: w_i \xrightarrow[G]{} w_{i+1}$  ( $w_i$  produce direttamente  $w_{i+1}$ ),



cioè:

$$y \xrightarrow{*} z \Leftrightarrow \begin{cases} y = z \\ oppure \\ w_1 = y \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \dots \Rightarrow w_{n-1} \Rightarrow w_n = z \end{cases}$$

### Osservazione 2.2

La nozione di derivazione diretta stabilisce una relazione binaria in  $(X \cup V)^*$ .

Date due stringhe  $y$  e  $z$ , il simbolo  $\Rightarrow$  può esserci o meno; dipende dall'esistenza di una produzione. Allora possiamo anche definire una composizione di relazioni:

$$y \xrightarrow[2]{\text{def}} z \Leftrightarrow \exists w: y \Rightarrow w \text{ e } w \Rightarrow z$$

dove 2 è il numero di trascrizioni necessarie per passare da  $y$  a  $z$  (ossia, la lunghezza della derivazione).

Da ciò si ha:

$$\xrightarrow{*} = I \cup \xrightarrow[2]{\text{def}} \cup \xrightarrow[3]{\text{def}} \cup \dots$$

ove  $I$  è la relazione identica e  $\xrightarrow{n}$  indica la composizione della relazione  $\Rightarrow$   $n$  volte con se stessa.

- $\Rightarrow^*$  è la chiusura riflessiva e transitiva della relazione di derivazione diretta
- $\Rightarrow^+$  è la chiusura transitiva della stessa relazione.

### Esempio 2.13

Costruzione di un flow-chart.

Sia dato il flow-chart in Figura 2.3.

La relazione  $\Rightarrow$  è definita come segue: "due nodi sono in relazione  $\Rightarrow$  se esiste un arco orientato tra essi".

- $\Rightarrow^*$  è la chiusura riflessiva e transitiva dell'operatore  $\Rightarrow$  e dunque va interpretata come: "esiste un cammino di lunghezza finita che congiunge i due nodi"

$$R = \Rightarrow$$

		$R^0$				
		$x$	$y$	$z$	$t$	$u$
$R^1$	$x$	1				
	$y$			1		
	$z$				1	
	$t$					1
	$u$					1

		$R^2$				
		$x$	$y$	$z$	$t$	$u$
$R^1$	$x$					1
	$y$					
	$z$			1		
	$t$				1	
	$u$					1

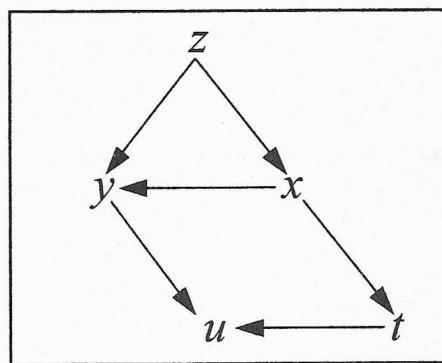


Figura 2.3

$R^2$  si può ottenere anche facendo il prodotto matriciale di  $R$  con se stessa.

$R^3$	$x$	$y$	$z$	$t$	$u$
$x$					
$y$					
$z$			1		
$t$					
$u$					

La  $R^* = \overset{*}{\Rightarrow}$  si ottiene facendo l'unione di queste relazioni, ossia raggruppando tutti gli 1 che compaiono.

$R^*$	$x$	$y$	$z$	$t$	$u$
$x$	1	1		1	1
$y$		1			1
$z$	1	1	1	1	1
$t$				1	1
$u$					1

## 2.5 Linguaggi generati da grammatiche e correttezza di una grammatica.

**Definizione 2.13** (Linguaggio generato da una grammatica)

Sia  $G = (X, V, S, P)$  una grammatica.

Il *linguaggio generato da G*, denotato con  $L(G)$ , è l'insieme delle stringhe di terminali derivabili dal simbolo di partenza  $S$ .

$$L(G) = \left\{ w \in X^* \mid S \xrightarrow[G]{*} w \right\}$$

■

Sono, dunque, stringhe di  $L(G)$  le stringhe che:

- a) consistono di soli terminali;
- b) possono essere derivate da  $S$  in  $G$ .

**Definizione 2.14** (Forma di frase)

Sia  $G = (X, V, S, P)$  una grammatica. Una stringa  $w$ ,  $w \in (X \cup V)^*$ , è una *forma di frase* di  $G$  se  $S \xrightarrow[G]{*} w$ .

■

Alle forme di frasi si applicano le stesse definizioni (es.: potenza) e gli stessi operatori (es.: concatenazione) dati per le stringhe.

**Proposizione 2.1**

Data una grammatica  $G = (X, V, S, P)$ ,  $L(G)$  è l'insieme delle forme di frase terminali (o *frasi*) di  $G$ .

### Definizione 2.15 (Grammatiche equivalenti)

Due grammatiche  $G$  e  $G'$  si dicono *equivalenti* se generano lo stesso linguaggio, ossia se:

$$L(G) = L(G')$$

■

### Esempio 2.14

Sia  $G = (X, V, S, P)$ , ove

$$X = \{a, b\}, \quad V = \{S\}, \quad P = \left\{ S \xrightarrow{(1)} aSb, S \xrightarrow{(2)} ab \right\}$$

Determiniamo  $L(G)$ .

$ab \in L(G)$  poiché  $S \xrightarrow{(2)} ab$ .

Se numeriamo le produzioni, possiamo indicare la produzione usata immediatamente al di sotto del simbolo  $\Rightarrow$ .

$\xrightarrow{(n)}$  ho applicato la produzione n

$y \xrightarrow{k} z$  produce  $z$  in  $k$  passi, dove  $k$  = lunghezza della derivazione.

$$a^2b^2 \in L(G) \text{ poiché } S \xrightarrow{(1)} aSb \xrightarrow{(2)} a^2b^2$$

$$a^3b^3 \in L(G) \text{ poiché } S \xrightarrow{3} a^3b^3$$

⋮

$$\{a^n b^n | n > 0\} \subseteq L(G)$$

Inoltre, qualsiasi derivazione da  $S$  in  $G$  produce frasi del tipo  $a^n b^n$ . Dunque  $L(G) \subseteq \{a^n b^n | n > 0\}$  e quindi  $L(G) = \{a^n b^n | n > 0\}$ .

Per rendere più concisa la descrizione di una grammatica, spesso ci limiteremo ad elencarne le produzioni, quando sia chiaro quale sia il simbolo di partenza e quali siano i terminali ed i nonterminali.

Inoltre, le produzioni con la stessa parte sinistra vengono accorpate attraverso l'uso del simbolo “|” (preso a prestito dalla BNF).

Infine, ometteremo l'indicazione della grammatica dalla simbologia di derivazione e derivazione diretta ( $\xrightarrow[G]{*}$  e  $\xrightarrow[G]$ ) quando sia chiaro dal contesto a quale grammatica si fa riferimento.

### Esempio 2.15

Sia data la seguente grammatica:

$$S \xrightarrow{(1)} A \mid B, A \xrightarrow{(2)} aA \mid a, B \xrightarrow{(3)} bB \mid b$$

Determiniamo  $L(G)$ .

Non sappiamo se applicare  $S \xrightarrow{(1)} A$  oppure  $S \xrightarrow{(2)} B$  inizialmente. I meccanismi di costruzione di un linguaggio sono generalmente *non deterministici*, poiché può non essere univoca la sostituzione da operare ad una forma di frase se uno stesso  $NT$  si trova a sinistra di 2 o più produzioni, come illustrato in Figura 2.4.

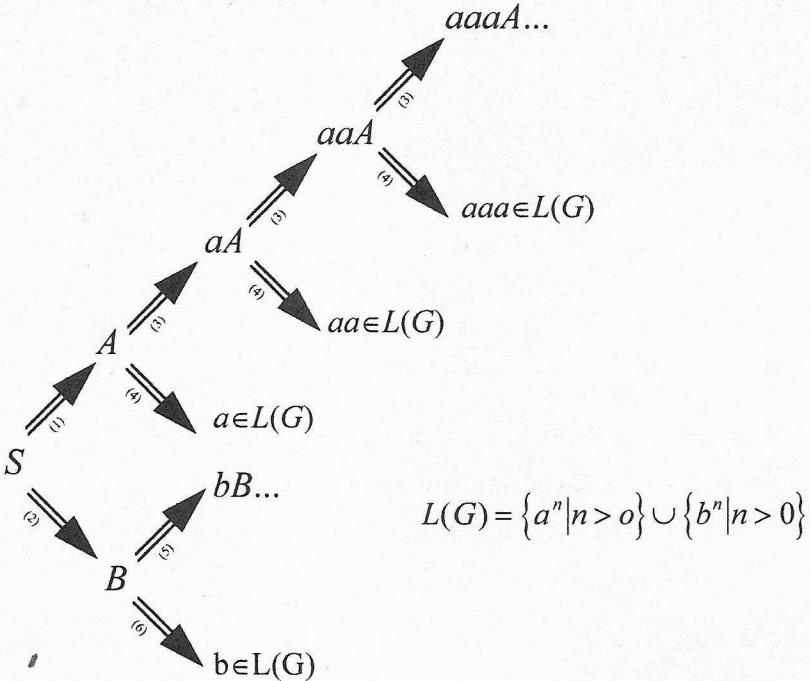


Figura 2.4

Dunque, una grammatica è uno *strumento generativo* di un linguaggio perché, data una qualsiasi parola di quel linguaggio, possiamo risalire mediante le produzioni al simbolo di partenza della grammatica.

Viceversa, dato il simbolo di partenza di una grammatica, seguendo uno qualsiasi dei cammini dell'albero di derivazione, si produce una parola “valida” del linguaggio.

### Osservazione 2.3

In generale, dato un linguaggio  $L$  ed una grammatica  $G$ , non esiste un algoritmo in grado di dimostrare che la grammatica genera il linguaggio, ossia

che  $L=L(G)$ . Più specificamente, non esiste un algoritmo che stabilisce se una data stringa è generata o no dalla grammatica presa in considerazione.

Tutto ciò si riassume nella seguente:

### **Proposizione 2.2**

Il problema di dimostrare la correttezza di una grammatica non è risolubile algoritmicamente, in generale.

In molti casi importanti, però, è possibile dimostrare per induzione che una particolare grammatica genera proprio un particolare linguaggio.

Queste dimostrazioni ci consentono di stabilire se, data una grammatica  $G$  ed un linguaggio  $L$ , risulta:

- 1)  $w \in L(G) \Rightarrow w \in L \quad (L(G) \subseteq L);^1$
- 2)  $w \in L \Rightarrow w \in L(G) \quad (L \subseteq L(G));^2$

e dunque se  $L=L(G)$ .

## **2.6 Esercizi.**

### **Esercizio 2.1**

- 1) Determinare la grammatica che genera il seguente linguaggio:

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

e dimostrare questo risultato.

- 2) Che tipo di grammatica genera  $L$  ?

---

<sup>1</sup> La grammatica  $G$  genera solo stringhe appartenenti al linguaggio  $L$ .

<sup>2</sup> Il linguaggio  $L$  comprende solo parole generabili dalla grammatica  $G$ .

$$1) \quad G = (X, V, S, P)$$

$$X = \{a, b\} \quad V = \{S\} \quad P = \left\{ S \xrightarrow{(1)} aSb, S \xrightarrow{(2)} ab \right\}$$

Dobbiamo dimostrare che  $L = L(G)$ .

i)  $L(G) \subset L$

Sia  $w$  una stringa derivabile da  $S$  (in  $G$ ).

$$w \in L(G) \stackrel{\text{def}}{\Leftrightarrow} S \xrightarrow{*} w, w \in X^*$$

Procediamo per *induzione sulla lunghezza della derivazione* di  $w$  da  $S$ .

Denoto con  $n$  la lunghezza della derivazione di  $w$  da  $S$ .

#### Passo base

$$n = 1$$

$S \xrightarrow{(2)} ab$  è l'unica derivazione di lunghezza  $n=1$  che genera stringhe di terminali.

È immediato verificare che  $ab \in L$ .

#### Passo induttivo

Dimostriamo che, per ogni  $n > 1$ , se supponiamo che il seguente enunciato è vero:

“se  $w' \in L(G)$ ,  $S \xrightarrow{n-1} w'$  ( $w'$  è derivabile in  $n-1$  passi da  $S$ ) allora  $w' \in L$ ”

allora anche l'enunciato:

“se  $w \in L(G)$ ,  $S \xrightarrow{n} w$  allora  $w \in L$ ” risulta vero.

Consideriamo:

$$w \in L(G), \text{ con } S \xrightarrow{n} w.$$

Per definizione (di derivabilità in  $n$  passi), esiste una sequenza di forme di frase  $w_1, w_2, \dots, w_n$  con  $w_n = w$ , tale che  $w_1$  deriva direttamente da  $S$  e, per ogni  $i$  ( $i = 1, 2, \dots, n-1$ ),  $w_{i+1}$  deriva direttamente da  $w_i$ .

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n = w$$

È immediato osservare che il primo passo della derivazione è dato dalla applicazione della produzione (1) di  $G$  (altrimenti otterremmo la stringa  $ab$ , priva di nonterminali ed avremmo finito. Ma allora  $n=1$ ).

Si ha dunque:

$$S \xrightarrow{(1)}^{n-1} aSb \Rightarrow w_n = w$$

Per ipotesi di induzione, ogni stringa derivabile da  $S$  in  $n-1$  passi è una parola di  $L$ . Dunque, da  $S$  è possibile derivare in  $n-1$  passi una stringa del tipo:  $w' = a^k b^k$ ,  $k > 0$ .

Più precisamente,  $w' = a^{n-1} b^{n-1}$ , poiché:

$$S \xrightarrow{(1)}^k a^k S b^k, k > 0.$$

Ma allora la stringa:

$$aw'b = aa^{n-1}b^{n-1}b = a^n b^n$$

è ancora una stringa di  $L$  ed inoltre è derivabile da  $S$  in  $n$  passi.

Si ha, infatti:

$$S \xrightarrow{(1)}^{n-1} aSb \Rightarrow aw'b = a^n b^n = w$$

Risulta così dimostrato  $L(G) \subset L$ .

ii)  $L \subset L(G)$

Sia  $w$  una parola di  $L$ . Procediamo per *induzione sulla lunghezza della stringa  $w$* .

### Passo base

Prendiamo in considerazione la parola di  $L$  di lunghezza minima.

$$n = 1 \Leftrightarrow |w| = 2 \quad w = ab$$

Dobbiamo dimostrare che:  $S \xrightarrow{(2)} ab$ .

Banale. Applichiamo la produzione (2) di  $G$  ed otteniamo che  $w = ab$  è direttamente derivabile da  $S$ .

$$S \xrightarrow{(2)} ab$$

### Passo induttivo

Dimostriamo che, per ogni  $n > 1$ , se supponiamo che il seguente enunciato è vero:

“se  $w' \in L, |w'| = 2(n-1)$  allora  $S \xrightarrow{(2)} w'$ ”

allora anche il seguente enunciato:

“se  $w \in L, |w| = 2n$  allora  $S \xrightarrow{(2)} w$ ”

risulta vero.

Sia  $w$  una parola su  $X$  tale che:

$$w \in L, |w| = 2n, n > 1.$$

Ovviamente,  $w = a^n b^n$  (unica parola di  $L$  di lunghezza  $2n$ ). Nella (ipotetica) derivazione di  $w$  da  $S$ , devo necessariamente applicare la produzione (1) di  $G$ , come 1° passo (altrimenti riotterrei la parola  $ab$ ).

Dunque:

$$S \xrightarrow{(1)} aSb \tag{a}$$

Per ipotesi di induzione, ogni parola di  $L$  di lunghezza  $2(n-1)$  è derivabile da  $S$  (in  $G$ ). Dunque, anche  $w' = a^{n-1} b^{n-1}$  è derivabile da  $S$ :

$$S \xrightarrow{*} w' = a^{n-1} b^{n-1} \tag{b}$$

Ne consegue che  $w = a^n b^n$  è derivabile da  $S$  e la relativa derivazione è ottenuta applicando in successione (a) e (b).

$$S \xrightarrow{(1)} \underbrace{aSb}_{(a)} \xrightarrow{(2)} \underbrace{aw'b}_{(b)} = aa^{n-1}b^{n-1}b = w$$

Dunque,  $L \subset L(G)$  e

$$L = L(G)$$

2)  $G$  è una grammatica libera da contesto.

### Esercizio 2.2

Sia data la seguente grammatica:

$$G = (X, V, S, P)$$

$$X = \{0, 1\} \quad V = \{S, A, B\}$$

$$P = \left\{ S \xrightarrow{(1)} 0B \mid 1A, A \xrightarrow{(2)} 0 \mid 0S \mid 1AA, B \xrightarrow{(3)} 1 \mid 1S \mid 0BB \right\}$$

Determinare il linguaggio generato da  $G$  e dimostrare il risultato.

Il linguaggio generato da  $G$  è il seguente:

$$L = \left\{ w \mid w \in \{0, 1\}^+, w \text{ ha lo stesso numero di } 0 \text{ e di } 1 \right\}$$

dobbiamo dimostrare che:

$$L = L(G)$$

Quindi bisogna dimostrare che:

- i)  $L(G) \subset L$
- ii)  $L \subset L(G)$

i) Dimostriamo che  $L(G) \subset L$ . Questo corrisponde a dimostrare che  $G$  genera solo parole con lo stesso numero di 0 e di 1.

Procediamo per induzione sulla lunghezza della stringa. Possiamo ricondurre questa dimostrazione a dimostrare che:

- i.a) ogni stringa derivabile da  $S$  in  $G$  possiede un numero uguale di 0 e di 1;
- i.b) ogni stringa derivabile da  $A$  ha uno 0 in più;
- i.c) ogni stringa derivabile da  $B$  ha un 1 in più.

Sia  $w$  una parola su  $X = \{0, 1\}$ .

#### Passo base

$$|w| = 1$$

- i.a) Non vi sono parole di lunghezza 1 derivabili da  $S$  in  $G$ ;
- i.b) La sola parola di lunghezza 1 derivabile da  $A$  in  $G$  è  $w = 0$

$$A \xrightarrow{(3)} 0$$

e tale parola ha uno 0 in eccesso;

- i.c) La sola parola di lunghezza 1 derivabile da  $B$  in  $G$  è  $w = 1$

$$B \xrightarrow{(6)} 1$$

e tale parola ha un 1 in eccesso.

$$|w| = 2$$

- i.a) Le sole parole di lunghezza 2 derivabili da  $S$  in  $G$  sono  $w_1 = 01$  e  $w_2 = 10$ .

$$S \xrightarrow{(1)} 0B \xrightarrow{(6)} 01$$

$$S \xrightarrow{(2)} 1A \xrightarrow{(3)} 10$$

sia  $w_1$  sia  $w_2$  hanno lo stesso numero di 0 e di 1;

- i.b) Non vi sono parole (stringhe di soli terminali) di lunghezza 2 derivabili da  $A$  in  $G$ ;

$$A \xrightarrow{(3)} 0$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(3)} 10A$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(3)} 1A0$$

$$A \xrightarrow{(4)} 0S \xrightarrow{(1)} 00B$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(4)} 10SA$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(4)} 1A0S$$

$$A \xrightarrow{(4)} 0S \xrightarrow{(2)} 01A$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(5)} 11AAA$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{(5)} 1A1AA$$

- i.c) Non vi sono parole (stringhe di soli terminali) di lunghezza 2 derivabili da  $B$  in  $G$ ;

$$B \xrightarrow{(6)} 1$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(6)} 01B$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(6)} 0B1$$

$$B \xrightarrow{(7)} 1S \xrightarrow{(1)} 10B$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(7)} 01SB$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(7)} 0B1S$$

$$B \xrightarrow{(7)} 1S \xrightarrow{(2)} 11A$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(8)} 00BBB$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{(8)} 0B0BB$$

### Passo induttivo

Dimostriamo che, per ogni  $n > 2$ , se supponiamo che i.a), i.b) e i.c) siano vere per ogni parola di lunghezza  $m$ , con  $2 \leq m < n$ , allora i.a), i.b) e i.c) risultano vere per parole di lunghezza  $n$ .

- i.a) Sia  $w$  una stringa derivabile da  $S$  in  $G$ , tale che:

$$|w| = n \text{ e } S \xrightarrow{*} w$$

Si hanno due possibili casi:

$$S \xrightarrow{(1)} 0B \xrightarrow{*} w \text{ e } w = 0\beta, \text{ ove } B \xrightarrow{*} \beta, |\beta| = n - 1$$

$$S \xrightarrow{(2)} 1A \xrightarrow{*} w \text{ e } w = 1\alpha, \text{ ove } A \xrightarrow{*} \alpha, |\alpha| = n - 1$$

Per ipotesi di induzione i.b),  $\alpha$  ha uno 0 in più. Tale 0 è bilanciato dall'1 iniziale in  $w$ . Ne segue che  $w$  ha lo stesso numero di 0 e di 1. Analogamente, per ipotesi di induzione i.c),  $\beta$  ha un 1 in eccesso, che viene bilanciato dallo 0 iniziale di  $w$ . Ne consegue che, anche in questo caso,  $w$  ha lo stesso numero di 0 e di 1.

i.b) Sia  $w$  una stringa di lunghezza  $n$  derivabile da  $A$  in  $G$ :

$$|w| = n \text{ e } A \xrightarrow{*} w$$

Si hanno due possibili casi:

$$A \xrightarrow{(4)} 0S \xrightarrow{*} w \text{ e } w = 0\beta, \text{ ove } S \xrightarrow{*} \beta, |\beta| = n - 1$$

$$A \xrightarrow{(5)} 1AA \xrightarrow{*} w \text{ e } w = 1\alpha, \text{ ove } AA \xrightarrow{*} \alpha, |\alpha| = n - 1$$

Nel 1° caso, l'enunciato i.a) appena dimostrato ci garantisce che la stringa  $\beta$  possiede lo stesso numero di 0 e di 1. Dunque  $w = 0\beta$  ha uno 0 in eccesso.

Nel 2° caso, per l'ipotesi di induzione i.b), la stringa  $\alpha$  ha due 0 in eccesso (in quanto concatenazione di due stringhe derivabili da  $A$ ). Di conseguenza,  $w = 1\alpha$  ha uno 0 in eccesso.

i.c) Sia  $w$  una stringa di lunghezza  $n$  derivabile da  $B$  in  $G$ :

$$|w| = n \text{ e } B \xrightarrow{*} w$$

Si hanno due possibili casi:

$$B \xrightarrow{(7)} 1S \xrightarrow{*} w \text{ e } w = 1\alpha, \text{ ove } S \xrightarrow{*} \alpha, |\alpha| = n - 1$$

$$B \xrightarrow{(8)} 0BB \xrightarrow{*} w \text{ e } w = 0\beta, \text{ ove } BB \xrightarrow{*} \beta, |\beta| = n - 1$$

Nel 1° caso, da i.a) sappiamo che  $\alpha$  possiede lo stesso numero di 0 e di 1. Dunque  $w = 1\alpha$  ha un 1 in eccesso.

Nel 2° caso, per l'ipotesi di induzione i.c), la stringa  $\beta$  ha due 1 in eccesso (in quanto concatenazione di due stringhe derivabili da  $B$ ). Di conseguenza,  $w = 0\beta$  ha uno 1 in eccesso.

Risulta così dimostrato il "teorema" i):  $L(G) \subset L$

ii) Dimostriamo che  $L \subset L(G)$ . Questo corrisponde a dimostrare che "ogni parola con uguale numero di 0 e di 1 è generata da  $G$ ".

Procediamo per *induzione sulla lunghezza della stringa*.

Possiamo ricondurre questa dimostrazione a dimostrare che:

- i.a) ogni stringa con uguale numero di 0 e di 1 è derivabile da  $S$  in  $G$ ;
- i.b) ogni stringa con uno 0 in eccesso è derivabile da  $A$  (in  $G$ );
- i.c) ogni stringa con un 1 in eccesso è derivabile da  $B$  (in  $G$ ).

Sia  $w$  una parola su  $L$  (ossia, avente uno stesso numero di 0 e di 1).

Passo base

$$|w| = 1$$

ii.a) Non vi sono parole di lunghezza 1 che abbiano lo stesso numero di 0 e di 1;

ii.b) La sola parola di lunghezza 1 con uno 0 in eccesso è  $w = 0$  ed è derivabile direttamente da  $A$  in  $G$ :

$$A \xrightarrow{(3)} 0$$

ii.c) La sola parola di lunghezza 1 con un 1 in eccesso è  $w = 1$  ed è derivabile da  $B$  in  $G$ :

$$B \xrightarrow{(6)} 1$$

$$|w| = 2$$

ii.a) Le sole parole di lunghezza 2 che possiedono lo stesso numero di 0 e di 1 sono:

$$w_1 = 01 \text{ e } w_2 = 10$$

Entrambe sono derivabili da  $S$  in  $G$  attraverso le seguenti derivazioni:

$$\begin{array}{c} S \Rightarrow 0B \Rightarrow 01 \\ (1) \qquad (6) \end{array}$$

$$\begin{array}{c} S \Rightarrow 1A \Rightarrow 10 \\ (2) \qquad (3) \end{array}$$

ii.b) Non esistono parole di lunghezza due con uno 0 in eccesso.

ii.c) Non esistono parole di lunghezza due con un 1 in eccesso.

#### Passo induttivo

Dimostriamo che, per ogni  $n > 2$ , se supponiamo che ii.a), ii.b) e ii.c) siano vere per ogni parola di lunghezza  $m$ , con  $2 \leq m < n$ , allora ii.a), ii.b) e ii.c) risultano vere per parole di lunghezza  $n$ .

ii.a) Sia  $w$  una parola di lunghezza  $n$  con lo stesso numero di 0 e di 1:

$$|w| = n$$

Supponiamo che  $w$  inizi con uno 0. Dunque,  $w = 0\beta$ , ove  $|\beta| = n - 1$  e  $\beta$  ha un 1 in eccesso. Per ipotesi di induzione ii.c),  $\beta$  è derivabile da  $B$  in  $G$ :

$$B \xrightarrow{*} \beta$$

Ma allora,  $w$  è derivabile da  $S$  in  $G$  e la relativa derivazione è:

$$\begin{array}{c} S \Rightarrow 0B \xrightarrow{*} 0\beta = w \\ (1) \end{array}$$

Supponiamo che  $w$  inizi con un 1. Dunque,  $w = 1\alpha$ , ove  $|\alpha| = n - 1$  e  $\alpha$  ha uno 0 in eccesso.

Per ipotesi di induzione ii.b),  $\alpha$  è derivabile da  $A$  in  $G$ :

$$A \xrightarrow{*} \alpha$$

Ma allora,  $w$  è derivabile da  $S$  in  $G$  e la relativa derivazione è:

$$S \xrightarrow{(2)} 1A \xrightarrow{*} 1\alpha = w$$

ii.b) Sia  $w$  una parola di lunghezza  $n$  con uno 0 in eccesso:

$$|w| = n$$

$w$  può avere una delle seguenti due forme:

- 1)  $w = 0\beta$ , ove  $|\beta| = n - 1$  e  $\beta$  ha lo stesso numero di 0 e di 1
- 2)  $w = 1\alpha$ , ove  $|\alpha| = n - 1$  e  $\alpha$  ha due 0 in eccesso.

Se  $w = 0\beta$ , l'enunciato ii.a) vale per  $\beta$  e si ha:

$$S \xrightarrow{*} \beta$$

Dunque,  $w$  è derivabile da  $A$  in  $G$  e la relativa derivazione è:

$$A \xrightarrow{(4)} 0S \xrightarrow{*} 0\beta = w$$

Se  $w = 1\alpha$ , nella derivazione di  $w$  da  $A$  (in  $G$ ) che stiamo ricercando, il 1° passo consiste nell'applicazione della produzione (5)

$$A \rightarrow 1AA$$

Vogliamo dimostrare che:

$$AA \xrightarrow{*} \alpha$$

ove  $\alpha$  ha due 0 in eccesso. È sempre possibile considerare  $\alpha$  come il risultato della concatenazione di due stringhe,  $\alpha'$  e  $\alpha''$ , entrambe con uno 0 in eccesso:

$$\alpha = \alpha'\alpha'' \quad \text{ove } |\alpha'| < n \text{ e } |\alpha''| < n$$

Per ipotesi di induzione ii.b), si ha:

$$A \xrightarrow{*} \alpha' \quad \text{e} \quad A \xrightarrow{*} \alpha''$$

e dunque una derivazione di  $w$  da  $A$  è:

$$\begin{aligned} A \xrightarrow{*} 1AA \xrightarrow{*} 1\alpha'A \xrightarrow{*} 1\alpha'\alpha'' &= 1\alpha = w \quad (\text{è l'unica derivazione di } w \text{ da } A?) \\ \xrightarrow{(5)} &\Rightarrow 100S \xrightarrow{*} 100\beta = w \end{aligned}$$

ii.c) Sia  $w$  una parola di lunghezza  $n$  con un 1 in eccesso:

$$|w| = n$$

$w$  può avere una delle seguenti due forme:

- 1)  $w = 1\alpha$ , ove  $|\alpha| = n - 1$  e  $\alpha$  ha lo stesso numero di 0 e di 1.
- 2)  $w = 0\beta$ , ove  $|\beta| = n - 1$  e  $\beta$  ha due 1 in eccesso.

Se  $w = 1\alpha$ , l'enunciato ii.a) vale per  $\alpha$  e si ha:

$$S \xrightarrow{*} \alpha$$

Dunque,  $w$  è derivabile da  $B$  in  $G$  e la relativa derivazione è:

$$\xrightarrow{(7)} B \xrightarrow{*} 1S \xrightarrow{*} 1\alpha = w$$

Se  $w = 0\beta$ , nella derivazione di  $w$  da  $A$  (in  $G$ ) che stiamo ricercando, il 1° passo consiste nell'applicazione della produzione (8)

$$B \rightarrow 0BB$$

Vogliamo dimostrare che:

$$BB \xrightarrow{*} \beta$$

ove  $\beta$  ha due 1 in eccesso. È sempre possibile considerare  $\beta$  come il risultato della concatenazione di due stringhe,  $\beta'$  e  $\beta''$ , entrambe con un 1 in eccesso:

$$\beta = \beta'\beta'' \quad \text{ove } |\beta'| < n \text{ e } |\beta''| < n$$

Per ipotesi di induzione ii.c), si ha:

$$B \xrightarrow{*} \beta' \quad \text{e} \quad B \xrightarrow{*} \beta''$$

e dunque una derivazione di  $w$  da  $B$  è:

$$\underset{(8)}{B \xrightarrow{*} 0BB \xrightarrow{*} 0\beta'B \xrightarrow{*} 0\beta'\beta'' = 0\beta = w}$$

Risulta così dimostrato il “teorema” ii):  $L \subset L(G)$ .

Si ha:  $L = L(G)$ .

### Esercizio 2.3

1) Determinare la grammatica che genera il seguente linguaggio:

$$L = \{a^n b^{2n} \mid n > 0\}$$

e dimostrare questo risultato.

2) Di che tipo è la grammatica che genera  $L$ ?

$$1) \qquad G = (X, V, S, P)$$

$$X = \{a, b\} \qquad V = \{S\} \qquad P = \left\{ S \xrightarrow{(1)} aSbb, S \xrightarrow{(2)} abb \right\}$$

Dobbiamo dimostrare:

$$L = L(G)$$

Quindi bisogna dimostrare che:

i)  $L(G) \subset L$

ii)  $L \subset L(G)$

ii)  $L(G) \subset L$

Sia  $w$  una parola derivabile da  $S$  in  $G$ .

$$w \in L(G) \stackrel{\text{def}}{\Leftrightarrow} S \xrightarrow{*} w, \quad w \in X^*$$

Procediamo per induzione sulla lunghezza della derivazione di  $w$  da  $S$ .

Sia  $n$  tale lunghezza.

Passo base

$$n = 1$$

$S \xrightarrow[2]{n} abb$  è la sola derivazione di lunghezza 1 che genera parole

$$\text{su } X = \{a, b\}$$

Si ha che:  $abb \in L$ .

Passo induttivo

Dimostriamo che:

$$\forall n, n > 1 : \left[ \left( \left( w' \in L(G), S \xrightarrow{n-1} w' \right) \Rightarrow w' \in L \right) \Rightarrow \left( \left( w \in L(G), S \xrightarrow{n} w \right) \Rightarrow w \in L \right) \right]$$

Consideriamo:

$$w \in L(G), \text{ con } S \xrightarrow{n} w$$

$$S \xrightarrow{n} w \stackrel{\text{def}}{\Leftrightarrow} \exists w_1, w_2, \dots, w_n : S \Rightarrow w_1, w_i \Rightarrow w_{i+1}, i=1, 2, \dots, n-1 \text{ e } w_n = w$$

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n = w$$

Necessariamente si ha:  $w_1 = aSbb$  (altrimenti  $w_1 = abb$  ed  $n = 1$ ).

Dunque:

$$S \xrightarrow[1]{n-1} aSbb \Rightarrow w_n = w$$

Per ipotesi di induzione, ogni stringa derivabile da  $S$  in  $n-1$  passi è una parola di  $L$ . Dunque, da  $S$  è possibile derivare in  $n-1$  passi una stringa del tipo:

$$w' = a^k b^{2k}, k > 0$$

Più precisamente,  $w' = a^{n-1}b^{2(n-1)}$  poiché:

$$S \xrightarrow[(1)]{k} a^k S b^{2k}, \quad k > 0$$

Ma allora la stringa:

$$aw'bb = aa^{n-1}b^{2(n-1)}bb = a^n b^{2n}$$

è ancora una parola di  $L$  ed è derivabile da  $S$  in  $G$  in  $n$  passi, attraverso la seguente derivazione:

$$S \xrightarrow[(1)]{n-1} aSbb \xrightarrow{n-1} aw'bb = a^n b^{2n} = w$$

Si ha dunque:

$$L(G) \subset L$$

ii)  $L \subset L(G)$

Sia  $w$  una parola di  $L$ .

Procediamo per *induzione sulla lunghezza della parola  $w$* .

#### Passo base

$$n = 1 \Leftrightarrow |w| = 3$$

Sia  $w = abb$ . Dobbiamo determinare una derivazione di  $w$  da  $S$  in  $G$ .

$$S \xrightarrow{(2)} \overbrace{abb}^{(2)}$$

#### Passo induttivo

Dimostriamo che, per ogni  $n > 1$ :

$$\left( (w' \in L, |w'| = n - 1 + 2(n - 1) = 3n - 3) \Rightarrow S \xrightarrow{*} w' \right) \Rightarrow \left( (w \in L, |w| = n + 2n = 3n) \Rightarrow S \xrightarrow{*} w \right)$$

Sia  $w$  una parola su  $X = \{a, b\}$  tale che:

$$w \in L, \quad |w| = 3n, \quad n > 1$$

L'unica parola di  $L$  di lunghezza  $3n$  è:

$$w = a^n b^{2n}$$

Nella derivazione che stiamo cercando, dobbiamo necessariamente applicare la produzione (1) di  $G$ , come 1° passo:

$$S \xrightarrow{(1)} aSbb$$

Per ipotesi di induzione, ogni parola di  $L$  di lunghezza  $3n-3$  è derivabile da  $S$  in  $G$ . Anche  $w' = a^{n-1}b^{2(n-1)}$  è dunque derivabile da  $S$  in  $G$ :

$$S \xrightarrow{*} w' = a^{n-1}b^{2(n-1)}$$

Ne consegue che  $w = a^n b^{2n}$  è derivabile da  $S$  e la sua derivazione è data da:

$$S \xrightarrow{(1)} aSbb \xrightarrow{*} aw'bb = aa^{n-1}b^{2(n-1)}bb = a^n b^{2n} = w$$

Dunque:  $L \subset L(G)$  e  $L = L(G)$ .

2)  $G$  libera da contesto.

#### Esercizio 2.4

Dimostrare per induzione che il linguaggio  $L$  generato dalla seguente grammatica è vuoto:

$$G = (X, V, S, P)$$

$$\begin{array}{lll} X = \{a, b, c\} & V = \{S, A, B\} & P = \left\{ \begin{array}{l|l} S \xrightarrow{(1)} aBS & BA \\ aB \xrightarrow{(3)} Ac & a \\ bA \xrightarrow{(5)} S & Ba \end{array} \right. \end{array}$$

Dobbiamo dimostrare che:

$$L(G) = \emptyset$$

È sufficiente dimostrare che  $L(G) \subset \emptyset$ , in quanto l'inclusione opposta è una tautologia ( $\emptyset \subset L(G)$  è vera per ogni  $G$ ).

Dimostrare che  $L(G) \subset \emptyset$  significa dimostrare che, in qualsiasi passo di una derivazione da  $S$ , la stringa ottenuta presenta almeno un simbolo nonterminale. Cioè:

$$w \in L(G) \Rightarrow w \in \emptyset$$

dove  $L(G) = \left\{ w \in X^* \mid S \xrightarrow[G]{*} w \right\}$  e

$$\forall n, S \xrightarrow[G]{n} w \Rightarrow w = \alpha N \beta, N \in V, \alpha, \beta \in (X \cup V)^*$$

Procediamo per *induzione sulla lunghezza di una derivazione* da  $S$ .

Denotiamo con  $n$  la lunghezza di una derivazione da  $S$ .

### Passo base

$$n = 1$$

$S \xrightarrow[(1)]{n} aBS$  e  $S \xrightarrow[(2)]{n} bA$  sono le uniche derivazioni possibili di lunghezza  $n = 1$ . Entrambe generano stringhe che presentano almeno un nonterminale.

### Passo induttivo

Dimostriamo che, per ogni  $n > 1$ , se supponiamo che il seguente enunciato è vero:

"se  $S \xrightarrow{n} w'$  allora  $\exists N \in V: w' = yNz, y, z \in (V \cup X)^*$ "

allora anche l'enunciato:

"se  $S \xrightarrow{n} w$  allora  $\exists N \in V: w = yNz, y, z \in (V \cup X)^*$ "

risulta vero.

Consideriamo una qualunque derivazione in  $G$  costituita da  $n$  passi:

$$S \xrightarrow{n} w$$

Per definizione (di derivabilità in  $n$  passi), esiste una sequenza di stringhe  $w_1, w_2, \dots, w_n$ , con  $w_n = w$ , tale che  $w_1$  deriva direttamente da  $S$  e, per ogni  $i$ ,  $i = 1, 2, \dots, n-1$ ,  $w_{i+1}$  deriva direttamente da  $w_i$ .

Dunque:

$$S \xrightarrow{n-1} w_{n-1} \Rightarrow w_n = w$$

Per ipotesi di induzione, ogni stringa derivabile da  $S$  in  $n-1$  passi presenta un nonterminale.

Dunque, anche  $w_{n-1}$  presenta un nonterminale (o variabile).

Si hanno le seguenti possibilità:

- ♦ in  $w_{n-1}$  compare il nonterminale  $S$ :

allora in  $w$  abbiamo ancora un nonterminale, in quanto le uniche due produzioni in cui  $S$  compare nella parte sinistra sono la (1)  $S \rightarrow aBS$  e la (2)  $S \rightarrow bA$ .

$$S \xrightarrow{n-1} w_{n-1} = ySz \xrightarrow{(1)} w_n = w = yaBSz$$

$$S \xrightarrow{n-1} w_{n-1} = ySz \xrightarrow{(2)} w_n = w = ybAz$$

- ♦ in  $w_{n-1}$  compare il nonterminale  $A$ :

allora se  $A$  è preceduto dal terminale  $b$  è possibile effettuare l' $n$ -esimo passo della derivazione, altrimenti non esistono derivazioni di lunghezza  $n$ .

Se  $w_{n-1} = ybAz$ , possiamo applicare le produzioni (5) e (6).

$$S \xrightarrow{n-1} w_{n-1} = ybAz \xrightarrow{(5)} w_n = w = ySz$$

$$S \xrightarrow{n-1} w_{n-1} = ybAz \xrightarrow{(6)} w_n = w = yBaz$$

e in  $w$  abbiamo, in entrambi i casi, ancora un nonterminale.

- ♦ in  $w_{n-1}$  compare il nonterminale  $B$ :

se  $B$  non è preceduto dal terminale  $a$  non è possibile effettuare l' $n$ -esimo passo della derivazione.

Se  $w_{n-1} = yaBz$ , possiamo applicare le produzioni (3) e (4).

$$S \xrightarrow{n-1} w_{n-1} = yaBz \xrightarrow{(3)} w_n = w = yAcz$$

$$S \xrightarrow{n-1} w_{n-1} = yaBz \xrightarrow{(4)} w_n = w = yaz$$

Ora, se  $y, z \in X^*$  allora  $w \in X^*$  e dunque non possiamo provare la veridicità dell'enunciato.

Infatti, esiste (almeno) una derivazione di una stringa di soli terminali:

$$S \xrightarrow{(1)} aBS \xrightarrow{(4)} aS \xrightarrow{(2)} abA \xrightarrow{(6)} aBa \xrightarrow{(4)} aa$$

Dunque:  $L(G) \neq \emptyset$ .

### Esercizio 2.5

Si consideri il seguente linguaggio:

$$L = \{a^n b^k c^{2n} \mid n, k \geq 1\}$$

Determinare una grammatica  $G$  tale che  $L = L(G)$  e dimostrare per induzione tale uguaglianza.

$$G = (X, V, S, P)$$

$$X = \{a, b, c\}$$

$$V = \{S, B\}$$

$$P = \left\{ \begin{array}{l} S \rightarrow a^{(1)} Scc \mid a^{(2)} Bcc, \\ B \rightarrow b^{(3)} B \mid b^{(4)} \end{array} \right\}$$

Si intende dimostrare che  $L = L(G)$ .

Possiamo ricondurre questa dimostrazione alla dimostrazione del seguente asserto:

$$S \stackrel{*}{\underset{G}{\Rightarrow}} w \Leftrightarrow w \in I,$$

ove:

$$I = \{a^n Sc^{2n} \mid n \geq 1\} \cup \{a^n Bc^{2n} \mid n \geq 1\} \cup \{a^n b^k Bc^{2n} \mid n, k \geq 1\} \cup \{a^n b^k c^{2n} \mid n, k \geq 1\}$$

$\Rightarrow$ ) Procediamo per induzione sulla lunghezza della derivazione da  $S$  in  $G$ . Sia  $w$  una forma di frase derivabile da  $S$  in  $G$  e denoto con  $t$  la lunghezza della derivazione:

$$S \stackrel{t}{\underset{G}{\Rightarrow}} w$$

### Passo base

$$t = 1$$

Le possibili derivazioni da  $S$  di lunghezza 1 sono:

$$S \stackrel{(1)}{\Rightarrow} aScc \quad \text{ed } aScc \in I \quad (n=1)$$

$$S \stackrel{(2)}{\Rightarrow} aBcc \quad \text{ed } aBcc \in I \quad (n=1)$$

### Passo induttivo

Dimostriamo che, per ogni  $t > 1$ , se supponiamo che il seguente enunciato è vero:

$$"S \stackrel{t}{\underset{G}{\Rightarrow}} w \Rightarrow w \in I",$$

allora anche l'enunciato:

$$" S \xrightarrow[G]{t+1} w \Rightarrow w \in I ",$$

risulta vero.

Consideriamo una generica forma di frase derivabile da  $S$  in  $G$  in  $t+1$  passi e denotiamo tale forma di frase con  $w$ .

Per definizione di derivazione in  $t+1$  passi, esiste una sequenza di forme di frase  $w_1, w_2, \dots, w_{t+2}$ , con  $w_i \in (X \cup V)^+$ ,  $i = 1, 2, \dots, t+2$ ,  $w_1 = S$ ,  $w_{t+2} = w$  tali che:

$$w_1 = S \xrightarrow[\text{\scriptsize $t+1$ passi}]{} w_2 \xrightarrow{} w_3 \xrightarrow{} \dots \xrightarrow{} w_{t+1} \xrightarrow{} w_{t+2} = w$$

Inoltre, per ipotesi di induzione,  $w_{t+1} \in I$ . Dunque,  $w_{t+1}$  è in una delle seguenti forme:

$$w_{t+1} = \begin{cases} a^n Sc^{2n} \\ a^n Bc^{2n} \\ a^n b^k Bc^{2n} \\ a^n b^k c^{2n} \end{cases} \quad k, n \geq 1$$

Analizziamo l'ultimo (il  $t+1$ -esimo) passo di derivazione:

$$S \xrightarrow[G]{t} w_{t+1}$$

Se  $w_{t+1} = a^n Sc^{2n}$ , si hanno due possibili forme per  $w_{t+2}$ :

$$S \xrightarrow[G]{(1)} a^n Sc^{2n} \xrightarrow{} a^{n+1} Sc^{2n+2} = w_{t+2} \text{ e } w_{t+2} \in I$$

$$S \xrightarrow[G]{(2)} a^n Sc^{2n} \xrightarrow{} a^{n+1} Bc^{2n+2} = w_{t+2} \text{ e } w_{t+2} \in I$$

Se  $w_{t+1} = a^n Bc^{2n}$ , si hanno due possibili forme per  $w_{t+2}$ :

$$S \xrightarrow[G]{(3)} a^n Bc^{2n} \xrightarrow{} a^n b Bc^{2n} = w_{t+2} \text{ e } w_{t+2} \in I$$

$$S \xrightarrow{(4)} a^n Bc^{2n} \Rightarrow a^n b c^{2n} = w_{t+2} \text{ e } w_{t+2} \in I$$

Se  $w_{t+1} = a^n b^k Bc^{2n}$ , si hanno due possibili forme per  $w_{t+2}$ :

$$S \xrightarrow{(3)} a^n b^k Bc^{2n} \Rightarrow a^n b^{k+1} Bc^{2n} = w_{t+2} \text{ e } w_{t+2} \in I$$

$$S \xrightarrow{(4)} a^n b^k Bc^{2n} \Rightarrow a^n b^{k+1} c^{2n} = w_{t+2} \text{ e } w_{t+2} \in I$$

Se  $w_{t+1} = a^n b^k c^{2n}$ , si ha:

$$S \xrightarrow{t} a^n b^k c^{2n}$$

e non ci sono ulteriori passi di derivazione possibili (non esistono derivazioni di lunghezza  $t+1$  la cui  $t+1$ -esima forma di frase sia del tipo  $w_{t+1} = a^n b^k c^{2n}$ ).

Risulta così dimostrato l'asserto.

$\Leftarrow$ ) Dimostriamo la seguente implicazione:

$$\begin{aligned} w &\in I \\ I &= I_1 \cup I_2 \cup I_3 \cup I_4 \\ I_1 &= \{a^n Sc^{2n} \mid n \geq 1\} \\ I_2 &= \{a^n Bc^{2n} \mid n \geq 1\} \quad \Rightarrow \quad S \xrightarrow[G]{*} w \\ I_3 &= \{a^n b^k Bc^{2n} \mid k, n \geq 1\} \\ I_4 &= \{a^n b^k c^{2n} \mid k, n \geq 1\} \end{aligned}$$

Sia  $w \in I_1 = \{a^n Sc^{2n} \mid n \geq 1\}$ .

Procediamo per induzione su  $n$ .

$$n = 1 \quad w = aSc^{2n}$$

e la derivazione cercata è:

$$S \xrightarrow{(1)} aSc^{2n}$$

Supponiamo vera:

$$w' = a^{n-1} Sc^{2(n-1)} \xrightarrow[G]{*} S \xrightarrow{*} w'$$

e dimostriamo che  $w = a^n Sc^{2n}$  è derivabile da  $S$  in  $G$ .

Per ipotesi di induzione:

$$S \xrightarrow{*} a^{n-1} Sc^{2(n-1)}$$

Dunque si ha:

$$S \xrightarrow{*} a^{n-1} Sc^{2(n-1)} \xrightarrow{(1)} a^{n-1} a S cc c^{2(n-1)} = a^n Sc^{2n}$$

Sia  $w \in I_2 = \{a^n Bc^{2n} \mid n \geq 1\}$ .

Procediamo per induzione su  $n$ .

$$n=1 \quad w = aBcc$$

e la derivazione cercata è:

$$S \xrightarrow{(2)} aBcc$$

Sia  $w = a^n Bc^{2n}$ . Dal risultato precedente sappiamo che  $(w' \in I_1 \Rightarrow S \xrightarrow{*} w')$  con  $w' = a^{n-1} Sc^{2(n-1)}$ ; si ha:

$$S \xrightarrow{*} a^{n-1} Sc^{2(n-1)}$$

Dunque  $w = a^n Bc^{2n}$  è derivabile da  $S$  in  $G$  attraverso la seguente derivazione:

$$S \xrightarrow{*} a^{n-1} Sc^{2(n-1)} \xrightarrow{(2)} a^{n-1} a B cc c^{2(n-1)} = a^n Bc^{2n}$$

Sia  $w \in I_3 = \{a^n b^k Bc^{2n} \mid k, n \geq 1\}$ .

Fissiamo un generico  $k$  e dimostriamo per induzione su  $n$  che  $w = a^n b^k Bc^{2n}$  è derivabile da  $S$  in  $G$ .

$$n=1 \quad w = ab^k Bcc$$

e la derivazione cercata è:

$$S \xrightarrow{(2)} aBcc \xrightarrow{(3)} ab^k Bcc$$

Sia  $w = a^n b^k Bc^{2n}$ . Per ipotesi di induzione:

$$S \xrightarrow{*} a^{n-1} b^k Bc^{2(n-1)}$$

Dunque  $w = a^n b^k Bc^{2n}$  è derivabile da  $S$  in  $G$  attraverso la seguente derivazione:

$$S \xrightarrow{*} a^n Bc^{2n} \xrightarrow{(3)} ab^k Bc^{2n}$$

in cui abbiamo utilizzato il risultato precedente ( $w' \in I_2 \Rightarrow S \xrightarrow{*} w'$  con  $w' = a^n Bc^{2n}$ ).

Fissiamo ora un generico  $n$  e dimostriamo per induzione su  $k$  che  $w = a^n b^k Bc^{2n}$  è derivabile da  $S$  in  $G$ .

$k = 1$        $w = a^n b Bc^{2n}$  e la derivazione cercata è:

$$S \xrightarrow{*} a^n Bc^{2n} \xrightarrow{(3)} a^n b Bc^{2n}$$

Sia  $w = a^n b^k Bc^{2n}$ . Per ipotesi di induzione, si ha:

$$S \xrightarrow{*} a^n b^{k-1} Bc^{2n}$$

Pertanto,  $w = a^n b^k Bc^{2n}$  è derivabile da  $S$  in  $G$  e la derivazione è:

$$S \xrightarrow{*} a^n b^{k-1} Bc^{2n} \xrightarrow{(3)} a^n b^k Bc^{2n}$$

Sia  $w \in I_4 = \{a^n b^k c^{2n} \mid k, n \geq 1\}$ .

Fissiamo arbitrariamente il valore di  $k$  e dimostriamo per induzione su  $n$  che  $w$  è derivabile da  $S$  in  $G$ .

$n = 1$        $w = ab^k cc$

e la derivazione cercata è:

$$S \xrightarrow{(2)} aBcc \xrightarrow{(3)} ab^{k-1}Bcc \xrightarrow{(4)} ab^kcc$$

Sia  $w = a^n b^k c^{2n}$ . Per ipotesi di induzione:

$$S \xrightarrow{*} a^{n-1} b^k c^{2(n-1)}$$

Dunque  $w = a^n b^k c^{2n}$  è derivabile da  $S$  in  $G$  attraverso la seguente derivazione:

$$S \xrightarrow{*} a^n b^{k-1} Bc^{2n} \xrightarrow{(4)} a^n b^k c^{2n}$$

in cui abbiamo utilizzato il risultato precedente ( $w' \in I_3 \Rightarrow S \xrightarrow{*} w'$  con  $w' = a^n b^{k-1} Bc^{2n}$ ).

Fissiamo ora un generico  $n$  e dimostriamo per induzione su  $k$  che  $w = a^n b^k c^{2n}$  è derivabile da  $S$  in  $G$ .

$k = 1 \quad w = a^n bc^{2n}$  e la derivazione cercata è:

$$S \xrightarrow{*} a^n Bc^{2n} \xrightarrow{(4)} a^n bc^{2n}$$

in cui abbiamo utilizzato il risultato  $w' \in I_2 \Rightarrow S \xrightarrow{*} w'$  con  $w' = a^n Bc^{2n}$ .

Sia  $w = a^n b^k c^{2n}$ . Abbiamo già dimostrato che una forma di frase  $a^n b^k Bc^{2n}$  è derivabile da  $S$  in  $G$  per ogni valore di  $n$  e di  $k$ . Dunque anche  $a^n b^{k-1} Bc^{2n}$  è derivabile.

Ma allora anche  $w = a^n b^k c^{2n}$  è derivabile e la sua derivazione è:

$$S \xrightarrow{*} a^n b^{k-1} Bc^{2n} \xrightarrow{(4)} a^n b^k c^{2n}$$

Risulta così completata la dimostrazione dell'asserto:

$$w \in I \quad \Rightarrow \quad S \xrightarrow[G]{*} w$$

e dunque vale il risultato:

$$S \stackrel{*}{\underset{G}{\Rightarrow}} w \Leftrightarrow w \in I$$

per ogni forma di frase  $w \in (X \cup V)^*$ .

Poiché  $L(G)$  è per definizione costituito di tutte e sole le frasi ( $w \in X^*$ ) derivabili da  $S$  in  $G$ , si ha, come caso particolare, che:

$$S \stackrel{*}{\underset{G}{\Rightarrow}} w \Leftrightarrow w \in \{a^n b^k c^{2n} \mid n, k \geq 1\} = L$$

### 3. Linguaggi liberi da contesto e linguaggi dipendenti da contesto.

#### 3.1 Grammatiche e linguaggi liberi da contesto.

**Definizione 3.1** (Grammatica libera da contesto)

Una grammatica  $G=(X,V,S,P)$  è *libera da contesto* (o context-free - C.F.) se, per ogni produzione  $v \rightarrow w$ ,  $v$  è un nonterminale.

$$G \text{ è libera da contesto} \stackrel{\text{def}}{\Leftrightarrow} \forall v \rightarrow w \in P : v \in V$$

■

**Definizione 3.2** (Linguaggio libero da contesto)

Un linguaggio  $L$  su un alfabeto  $X$  è *libero da contesto* se può essere generato da una grammatica libera da contesto.<sup>1</sup>

$$L \text{ libero da contesto} \stackrel{\text{def}}{\Leftrightarrow} \exists G \text{ libera da contesto tale che } L(G) = L$$

■

La maggior parte dei linguaggi di programmazione sono C.F.

Il termine C.F. nasce dal fatto che la sostituzione di un  $NT$  non è condizionata dal contesto - ossia dai caratteri adiacenti - in cui compare.

Un  $NT A$  in una forma di frase può sempre essere sostituito usando una produzione del tipo  $A \rightarrow \beta$ . La sostituzione è sempre valida.

---

<sup>1</sup> Se si ha una grammatica C.F. che genera  $L$ , non è detto che non esista un'altra grammatica che generi lo stesso linguaggio.

Viceversa, se  $L = L(G)$  e  $G$  non è C.F., non possiamo concludere che  $L$  non è C.F. perché non possiamo escludere che esista una grammatica C.F.  $G'$  per cui  $L = L(G')$ .

### Esempi di linguaggi C.F.

- 1) Il linguaggio delle parentesi ben formate.
- 2) Il linguaggio dei numeri interi relativi.
- 3) Il linguaggio  $\{a^n b^n \mid n > 0\}$ .
- 4) Il linguaggio delle stringhe con ugual numero di 0 e di 1.
- 5) Il linguaggio  $\{a^n b^{2n} \mid n > 0\}$ .

## **3.2 Grammatiche e linguaggi dipendenti da contesto.**

### Definizione 3.3 (Grammatica dipendente da contesto)

Una grammatica  $G=(X,V,S,P)$  è *dipendente da contesto* (o context-sensitive - C.S.) se ogni produzione è in una delle seguenti forme:

(1)  $yAz \rightarrow ywz$ ,

con  $A \in V$ ,  $y,z \in (X \cup V)^*$ ,  $w \in (X \cup V)^+$ ,

che si legge: “ $A$  può essere sostituita con  $w$  nel contesto  $y-z$ ” (contesto sinistro  $y$  e contesto destro  $z$ ).

(2)  $S \rightarrow \lambda$ ,

purché  $S$  non compaia nella parte destra di alcuna produzione. ■

### Definizione 3.4 (Linguaggio dipendente da contesto)

Un linguaggio  $L$  è *dipendente da contesto* se può essere generato da una grammatica dipendente da contesto. ■

### 3.3 Relazione tra linguaggi C.F. e C.S.

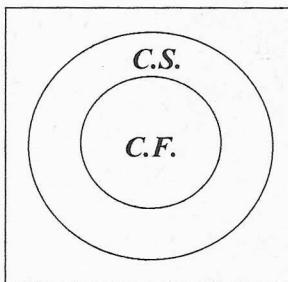


Figura 3.1

La relazione tra l'insieme dei linguaggi C.F. e l'insieme dei linguaggi C.S. è illustrato nella Figura 3.1.

Tale relazione sussiste perché le regole di produzione C.S. sono una generalizzazione di quelle C.F. Le produzioni C.F. sono un caso particolare delle produzioni di tipo (1) delle grammatiche C.S., che si verifica quando:

$$y = z = \lambda \quad \text{contesto destro e sinistro} \\ \text{equivalenti alla parola vuota.}$$

con la seguente eccezione:

osservando con attenzione la definizione di grammatica C.F. si nota che  $w \in (X \cup V)^*$ , mentre nella definizione di grammatica C.S.  $w \in (X \cup V)^+$ .

Dunque le grammatiche C.F. ammettono produzioni del tipo  $A \rightarrow \lambda$ , con  $A$  che può anche non essere il simbolo iniziale, mentre le grammatiche C.S. non ammettono tali produzioni.

Chiameremo tutte le produzioni del tipo  $A \rightarrow \lambda$   *$\lambda$ -produzioni* o  *$\lambda$ -regole*.

#### Esempi di produzioni contestuali

$$bC \rightarrow bc$$

$baACbA \rightarrow baAabA$

**Esempio di grammatica contestuale**

$S \rightarrow \lambda | bC$

$S \rightarrow \lambda$  è una produzione C.S. ed  $S$  non compare a destra di un'altra produzione.

$bC \rightarrow bc$

**Esempio di produzione non C.S. (né C.F.)**

$CB \rightarrow BC$

non è né C.S. né C.F. È una produzione *monotona* perché del tipo  $v \rightarrow w$  con  $|v| \leq |w|$

**3.4 Grammatiche e linguaggi monotoni.**

**Definizione 3.5 (Grammatica monotona)**

Una grammatica  $G=(X,V,S,P)$  è *monotona* se ogni sua produzione è monotona, cioè se  $\forall v \rightarrow w \in P: |v| \leq |w|$ . ■

**Definizione 3.6 (Linguaggio monotono)**

Un linguaggio  $L$  è *monotono* se può essere generato da una grammatica monotona. ■

**Esempio 3.1**

i)  $AB \rightarrow CDEF$

ii)  $CB \rightarrow BC$  sono produzioni monotone.

Una produzione monotona può essere sostituita da una sequenza di produzioni contestuali senza alterare il linguaggio generato.

### Esempio 3.2

- i)  $AB \rightarrow CDEF$  può essere sostituita dalle seguenti produzioni contestuali:
- i.1)  $AB \rightarrow AG$
  - i.2)  $AG \rightarrow CG$
  - i.3)  $CG \rightarrow CDEF$
- ii)  $CB \rightarrow BC$  diventa:
- |                           |   |
|---------------------------|---|
| ii.1) $CB \rightarrow XB$ | ii.1') $CB \rightarrow X_1B$            |
| ii.2) $XB \rightarrow XC$ | oppure ii.2') $X_1B \rightarrow X_1X_2$ |
| ii.3) $XC \rightarrow BC$ | ii.3') $X_1X_2 \rightarrow X_1C$        |
|                           | ii.4') $X_1C \rightarrow BC$            |

Difatti, vale la seguente:

### Proposizione 3.1

La classe dei linguaggi contestuali coincide con la classe dei linguaggi monotoni.

Tale proposizione deriva immediatamente dal seguente risultato:

### Teorema 3.1

Sia  $G$  una grammatica monotona, cioè tale che ogni produzione di  $G$  è della forma  $v \rightarrow w$ , con  $|v| \leq |w|$ , eccetto che ci può essere un'unica  $\lambda$ -produzione  $S \rightarrow \lambda$  se  $S$  non appare alla destra di una produzione. Esiste allora una grammatica C.S.  $G'$  equivalente a  $G$ , cioè tale che  $L(G) = L(G')$ .

Il teorema precedente può essere enunciato anche in una forma leggermente differente:

### Teorema 3.2

Un linguaggio  $L$  è dipendente da contesto se e solo se esiste una grammatica  $G$  tale che  $L = L(G)$  ed ogni produzione di  $G$  nella forma

$$u \rightarrow v$$

ha la proprietà che:  $0 < |u| \leq |v|$ , con una sola eccezione: se  $\lambda \in L(G)$  allora  $S \rightarrow \lambda$  è una produzione di  $G$  ed in tal caso  $S$  non può comparire nella parte destra di altre produzioni.

### Dimostrazione 3.2

$\Rightarrow$ ) Banale.

Se  $L$  è dipendente da contesto allora, per definizione, esiste  $G$  dipendente da contesto tale che  $L = L(G)$ .

$$L \text{ è C.S.} \stackrel{\text{def}}{\Leftrightarrow} \exists G \text{ C.S. : } L = L(G)$$

Allora ogni produzione di  $G$  è in una delle due forme:

$$(1) \quad yAz \rightarrow ywz,$$

con  $A \in V$ ,  $y, z \in (X \cup V)^*$ ,  $w \in (X \cup V)^+$ ,

$$(2) \quad S \rightarrow \lambda,$$

con  $S$  che non compare nella parte destra di alcuna produzione.

Dunque, ogni produzione di  $G$  verifica la condizione  $u \rightarrow v$ , con  $0 < |u| \leq |v|$ , se è del tipo (1), mentre se è del tipo (2) con  $S$  che non compare a destra di alcuna produzione, ricade nell'eccezione. Pertanto  $G$  è la grammatica cercata.

$\Leftarrow$  Sia  $G$  una grammatica in cui ogni produzione è nella forma  $u \rightarrow v$ , con  $0 < |u| \leq |v|$ . Senza ledere la generalità della dimostrazione, possiamo supporre che una generica produzione di  $G$  abbia il formato:  $A_1 A_2 \dots A_m \rightarrow B_1 B_2 \dots B_n$   $m \leq n$  ove  $A_i \in V$ ,  $i = 1, 2, \dots, m$ .

È legittimo fare questa assunzione in quanto, se  $A_j$  fosse un terminale potremmo sostituirlo nella produzione con un nuovo nonterminale  $A'_j$  ed aggiungere la nuova produzione  $A'_j \rightarrow A_j$ .

Denotiamo con  $C_1, C_2, \dots, C_m$   $m$  simboli nonterminali non presenti in  $G$ . Utilizziamo le  $C_k$ ,  $k=1, 2, \dots, m$  per costruire nuove regole contestuali che riscrivono la stringa  $A_1 A_2 \dots A_m$  con  $B_1 B_2 \dots B_n$ .

Le nuove regole sono:

$$\begin{aligned} A_1 A_2 \dots A_m &\rightarrow C_1 A_2 \dots A_m \\ C_1 A_2 \dots A_m &\rightarrow C_1 C_2 A_3 \dots A_m \\ &\dots \\ C_1 C_2 \dots C_{m-1} A_m &\rightarrow C_1 C_2 \dots C_{m-1} C_m B_{m+1} \dots B_n \\ C_1 C_2 \dots C_{m-1} C_m B_{m+1} \dots B_n &\rightarrow C_1 \dots C_{m-1} B_m B_{m+1} \dots B_n \\ &\dots \\ C_1 B_2 \dots B_n &\rightarrow B_1 B_2 \dots B_n \end{aligned}$$

La nuova grammatica  $G'$  che incorpora queste produzioni è contestuale e si può dimostrare che

$$L(G) = L(G').$$

Lasciamo per esercizio tale dimostrazione.

c.v.d.

### Esempio 3.3

Consideriamo il linguaggio:

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

Determiniamo una grammatica che genera tale linguaggio.

Il linguaggio somiglia ad un linguaggio già visto,  $\{a^n b^n | n > 0\}$ , generato dalla grammatica  $S \rightarrow aSb \mid ab$ .

Dunque le produzioni saranno del tipo:

$$S \xrightarrow{(1)} aSBC \mid aBC \quad \begin{array}{l} \text{il } NTB \text{ per generare le } b \\ \text{il } NTC \text{ per generare le } c \end{array}$$

Se applichiamo una volta la (1) e poi la (2), abbiamo però:

$$S \xrightarrow{(1)} aSBC \xrightarrow{(2)} aaBCBC$$

che non è nella forma desiderata in quanto le  $b$  e le  $c$  risulterebbero alternate.

Generalizzando, se applichiamo  $n-1$  volte la (1) e poi la (2), si ha:

$$S \xrightarrow{(1)} a^{n-1} S \underbrace{BCBC \dots BC}_{n-1} \xrightarrow{(2)} a^n \underbrace{BCBC \dots BC}_n = a^n (BC)^n$$

Abbiamo dunque bisogno di una produzione che riporti le  $B$  e le  $C$  in posizione corretta:

$$CB \xrightarrow{(3)} BC$$

con cui:

$$S \xrightarrow{(1)} aSBC \xrightarrow{(2)} aaBCBC \xrightarrow{(3)} aaBBCC = a^2 B^2 C^2$$

e

$$\begin{aligned} S &\xrightarrow{(1)} a^{n-1} S (BC)^{n-1} \xrightarrow{(2)} a^n \underbrace{BCBC \dots BC}_n \xrightarrow{(3)} a^n BBCC \underbrace{BC \dots BC}_{n-2} \xrightarrow{(3)} \\ &\xrightarrow{(3)} a^n BBCBCC \dots BC \xrightarrow{(3)} a^n BBBCCC \underbrace{BC \dots BC}_{n-3} \xrightarrow{*} a^n B^n C^n \end{aligned}$$

Quante volte abbiamo applicato la (3)?  $\sum_{i=1}^{n-1} (n-i)$  volte.

Ora abbiamo bisogno delle produzioni che generano i terminali  $b$  e  $c$ .

Consideriamo la produzione  $B \rightarrow b$ . Non va bene. Perché? Perché altrimenti potremmo applicarla all'inizio di una derivazione, ottenendo stringhe del tipo:

$$a^n b C b C \dots b C$$

che impediscono di applicare la (3) (ed hanno bisogno di ulteriori produzioni per scambiare le  $b$  con le  $C$ ) e quindi di trasformare le  $B$  in  $b$  solo dopo che le  $B$  hanno raggiunto la posizione corretta. Dunque dobbiamo considerare produzioni contestuali che trasformino le  $B$  in  $b$  solo dopo che hanno raggiunto la posizione corretta:

$$\overset{(4)}{aB \rightarrow ab} \quad \text{per la prima occorrenza delle } B$$

$$\overset{(5)}{bB \rightarrow bb} \quad \text{per le restanti occorrenze delle } B$$

Analogamente per le  $C$ :

$$\overset{(6)}{bC \rightarrow bc} \quad \text{per la prima occorrenza delle } C$$

$$\overset{(7)}{cC \rightarrow cc} \quad \text{per le restanti occorrenze delle } C$$

Quindi, la grammatica  $G$  che genera  $L = \{a^n b^n c^n \mid n > 0\}$  è:

$$S \rightarrow \overset{(1)}{aSBC} \mid \overset{(2)}{aBC}$$

$$\overset{(3)}{CB \rightarrow BC}$$

$$\overset{(4)}{aB \rightarrow ab}$$

$$\overset{(5)}{bB \rightarrow bb}$$

$$\overset{(6)}{bC \rightarrow bc}$$

$$\overset{(7)}{cC \rightarrow cc}$$

La grammatica è monotona, ma è facilmente determinabile una grammatica C.S. equivalente.

$$\begin{array}{ll}
 CB \xrightarrow{(3)} BC & CB \xrightarrow{(3.a)} XB \\
 & XB \xrightarrow{(3.b)} XC \\
 & XC \xrightarrow{(3.c)} BC
 \end{array}$$

La grammatica è comunque *non deterministica* perché non è univoca la sostituzione da operare in una forma di frase, come evidenziato dal successivo esempio.

### Esempio 3.5

$$S \xrightarrow{(1)} aSBC \xrightarrow{(2)} aaBCBC \xrightarrow{(4)} aabCBC \xrightarrow{(6)} a^2bcBC$$

e a questo punto non possiamo più andare avanti nella derivazione perché non ci sono produzioni che possiamo applicare.

Si può dimostrare formalmente che questa grammatica genera solo stringhe di tipo  $a^n b^n c^n$ .

$$L = L(G)$$

Non forniamo l'intera dimostrazione, ma ci limitiamo ad osservare che:

$$1) \quad L \subseteq L(G) \quad \text{in quanto}$$

$$\begin{array}{ll}
 a \xrightarrow{(1)} a^{n-1} S(BC)^{n-1} & \text{usando la (1) } n-1 \text{ volte} \\
 a^{n-1} S(BC)^{n-1} \xrightarrow{(2)} a^n (BC)^n & \text{usando la (2) 1 volta} \\
 a^n (BC)^n \xrightarrow{(3)} a^n B^n C^n & \text{usando la (3) } n-1 \text{ volte} \\
 a^n B^n C^n \xrightarrow{(4)} a^n b B^{n-1} C^n & \text{usando la (4) 1 volta} \\
 a^n b B^{n-1} C^n \xrightarrow{(5)} a^n b^n C^n & \text{usando la (5) } n-1 \text{ volte}
 \end{array}$$

$$a^n b^n C^n \xrightarrow{(6)} a^n b^n c C^{n-1} \quad \text{usando la (6) 1 volta}$$

$$a^n b^n c C^{n-1} \xrightarrow{(7)} a^n b^n c^n \quad \text{usando la (7) } n-1 \text{ volte}$$

Dunque:

$$\forall n, n > 0 : a^n b^n c^n \in L(G)$$

2)  $L(G) \subseteq L$  per esercizio.

Dunque:  $L = \{a^n b^n c^n | n > 0\}$  è un linguaggio C.S. (e monotono), per il teorema che stabilisce una relazione tra grammatiche C.S. e grammatiche monotone, mentre  $L = \{a^n b^n c^n | n > 0\}$  è C.F.

## 4. Linguaggi liberi da contesto.

### 4.1 Alberi di derivazione.

Le derivazioni in una grammatica libera da contesto possono essere rappresentate da *alberi* (detti *alberi di derivazione*).

La sequenza delle regole sintattiche utilizzate per generare una stringa  $w$  da una grammatica  $G$  di simbolo iniziale  $S$  definisce la *struttura* di  $w$ , che dunque potrebbe essere rappresentata da una delle derivazioni  $S \Rightarrow^* w$ . Al fine di disporre di una rappresentazione univoca, si preferisce ricorrere agli alberi di derivazione.

#### Definizioni preliminari

Un albero è un grafo orientato, acchiarito, connesso e avente al massimo un lato entrante in ciascun nodo.

La definizione rigorosa di “albero” è la seguente:

#### Definizione 4.1 (Albero)

Un albero  $T$  è un insieme finito, non vuoto, di vertici (detti *nodi*) tali che:

- esiste un vertice speciale, detto *radice* dell’albero;
- esiste una partizione  $T_1, T_2, \dots, T_m$ , con  $m \geq 0$ , degli altri vertici, tale che esista un ordinamento  $T_1, T_2, \dots, T_m$ , e ogni sottoinsieme di vertici  $T_i$ ,  $1 \leq i \leq m$ , sia a sua volta un albero.

Questa definizione è di tipo ricorsivo, ma non è circolare in quanto ogni sottoalbero  $T_i$  contiene meno vertici dell’albero  $T$ .

$T_1, T_2, \dots, T_m$  sono detti *sottoalberi* di  $T$ .

I vertici privi di discendenti sono le *foglie* dell'albero, gli altri sono i *nodi interni*.

La stringa dei simboli che etichettano le foglie di un albero  $T$ , letti nell'ordine da sinistra a destra, prende il nome di *frontiera* di  $T$ .

#### Definizione 4.2 (Partizione)

Gli insiemi (non vuoti)  $T_1, T_2, \dots, T_m, m \geq 0$ , costituiscono una *partizione* di un insieme (non vuoto)  $T$  se:

i)  $T_i \cap T_j = \emptyset$  per  $i \neq j, i, j = 1, 2, \dots, m$ ;

ii)  $\bigcup_{i=1}^m T_i = T$



#### Definizione 4.3 (Lunghezza di un cammino)



Dato un albero di derivazione, la *lunghezza di un cammino* dalla radice ad una foglia è data dal numero di nonterminali su quel cammino.

#### Definizione 4.4 (Altezza o profondità)

L'*altezza* di un albero è data dalla lunghezza del suo cammino più lungo.

■

#### Definizione 4.5 (Albero di derivazione)

Sia  $G = (X, V, S, P)$  una grammatica libera da contesto e  $w \in X^*$  una stringa derivabile da  $S$  in  $G$ ,  $S \xrightarrow{*} w$ .

Dicesi *albero di derivazione* l'albero  $T$  avente le seguenti proprietà:

- (1) la radice è etichettata con il simbolo iniziale  $S$ ;



- (2) ogni nodo interno (nodo non foglia) è etichettato con un simbolo di  $V$  (un nonterminale);
- (3) ogni nodo foglia è etichettato con un simbolo di  $X$  (un terminale) o con  $\lambda$ ;
- (4) se un nodo  $N$  è etichettato con  $A$ , ed  $N$  ha  $k$  discendenti diretti  $N_1, N_2, \dots, N_k$ , etichettati con i simboli  $A_1, A_2, \dots, A_k$ , rispettivamente, allora la produzione:

$$A \rightarrow A_1 A_2 \dots A_k$$

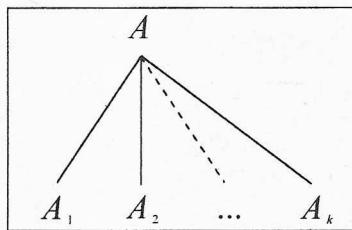


Figura 4.1

- deve appartenere a  $P$  (Figura 4.1);
- (5) la stringa  $w$  può essere ottenuta leggendo (e concatenando) le foglie dell'albero da sinistra a destra. ■

#### Osservazione 4.1

Un albero di derivazione non impone alcun ordine sull'applicazione delle produzioni in una derivazione. In altri termini, *data una derivazione, esiste uno ed un solo albero di derivazione* che la rappresenta, mentre *un albero di derivazione rappresenta in generale più derivazioni* (in funzione dell'ordine col quale si espandono i nonterminali).

### Esempio 4.1

Si consideri la seguente grammatica libera da contesto:

$$G_1 = (X, V, S, P)$$

dove:

$$X = \{a\} \quad V = \{S, H\}$$

$$P = \left\{ S \xrightarrow{(1)} Ha, H \xrightarrow{(2)} HS, H \xrightarrow{(3)} a \right\}$$

La stringa  $w=aaaa$  è derivabile da  $S$  in  $G_1$ . L'albero di derivazione di  $w=aaaa$  è rappresentato in Figura 4.2:

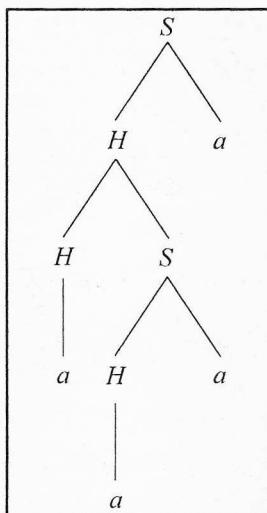


Figura 4.2

Tale albero corrisponde sia alla *derivazione sinistra*:

$$S \xrightarrow{(1)} Ha \xrightarrow{(2)} HSa \xrightarrow{(3)} aSa \xrightarrow{(1)} aHaa \xrightarrow{(3)} aaaa$$

sia alla *derivazione destra*:

$$S \xrightarrow{(1)} Ha \xrightarrow{(2)} HSa \xrightarrow{(1)} HHaa \xrightarrow{(3)} Haaa \xrightarrow{(3)} aaaa$$

## 4.2 Principio di sostituzione di sottoalberi.

**Definizione 4.6** (Derivazione destra/sinistra) 

Data una grammatica  $G=(X,V,S,P)$ , diremo che una *derivazione*  $S \Rightarrow^* w$ , ove:

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n = w^*$$

$$w_i = y_i A z_i, w_{i+1} = y_i w_i z_i, i=1, 2, \dots, n-1$$

è *destra (sinistra)* se, per ogni  $i$ ,  $i=1, 2, \dots, n-1$ , risulta:

$$z_i \in X^* \quad (y_i \in X^*)$$

In altre parole in una *derivazione destra (sinistra)* ad ogni passo si espande il nonterminale posto più a destra (sinistra). ■

### Esempio 4.2

Riconsideriamo la grammatica dell'Esercizio 2.2:

$$\begin{aligned} S &\rightarrow 0B \mid 1A \\ A &\rightarrow 0 \mid 0S \mid 1AA \\ B &\rightarrow 1 \mid 1S \mid 0BB \end{aligned}$$

che genera tutte e sole le stringhe che hanno un ugual numero di 1 e di 0.

Consideriamo la stringa 0011.

Una possibile derivazione è:

$$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011$$

Un'altra derivazione della stessa stringa è:

$$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1 \Rightarrow 0011$$

Il non determinismo insito nella derivazione scompare quando si considera il relativo albero di derivazione (Figura 4.3).

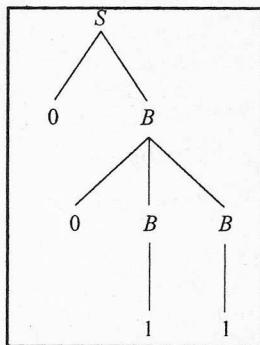


Figura 4.3

Consideriamo ora il sottoalbero con radice nel nodo di profondità minore etichettato con una  $B$  (ed individuato da un cerchio tratteggiato in Figura 4.4).

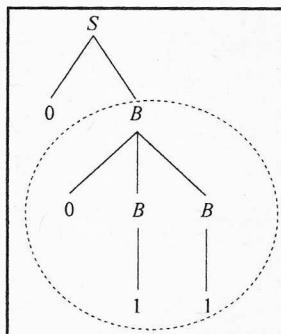


Figura 4.4

Cosa accade se un qualsiasi sottoalbero con radice in un nodo etichettato con una  $B$  viene sostituito con il sottoalbero ?

Il nuovo albero è ancora un albero di derivazione (ovviamente, per una stringa differente da 0011).

Consideriamo, ad esempio, il sottoalbero individuato dal cerchio pieno in Figura 4.5.

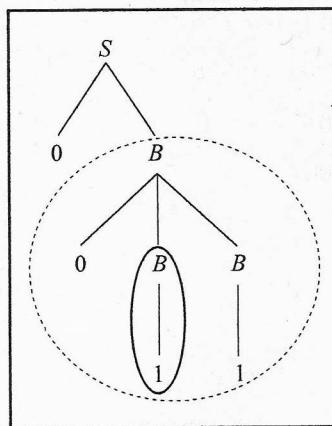


Figura 4.5

Il nuovo albero di derivazione è dato in Figura 4.6.

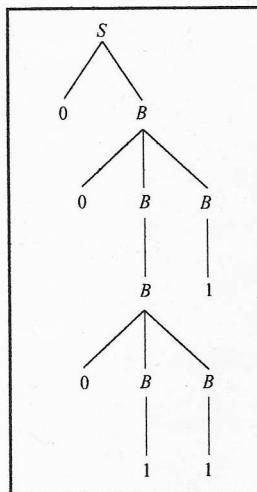


Figura 4.6

per cui:

$$S \xrightarrow{*} 000111$$

Possiamo ripetere indefinitamente questo processo di sostituzione di sottoalberi, ottenendo parole del linguaggio di lunghezza crescente:

$$\begin{array}{ll} 0011 & |w|=4 \\ 000111 & |w|=6 \\ w = 00001111 & |w|=8 \\ \vdots & \vdots \\ 00^n11^n & |w|=2n+2 \quad n=1,2,\dots \end{array}$$

La lunghezza cresce in maniera costante.

Nelle grammatiche C.F., dunque, possiamo sostituire alberi più piccoli con alberi di dimensioni maggiori, purché abbiano la stessa radice - più precisamente, purché i nodi radice siano etichettati con lo stesso *NT* - ottenendo ancora alberi di derivazione validi.

Una caratteristica dei linguaggi C.F., che discende direttamente dal processo descritto in precedenza, è che la lunghezza delle parole cresce in maniera costante.

Dunque, se una grammatica genera parole la cui lunghezza cresce in maniera esponenziale, allora il linguaggio generato non è libero.

Generalizziamo ora il discorso.

Supponiamo di avere un albero di derivazione  $T_z$  per una stringa  $z$  di terminali generata da una grammatica C.F.  $G$ , e supponiamo inoltre che il simbolo  $NT A$  compaia due volte su uno stesso cammino.

La situazione è rappresentata graficamente in Figura 4.7.

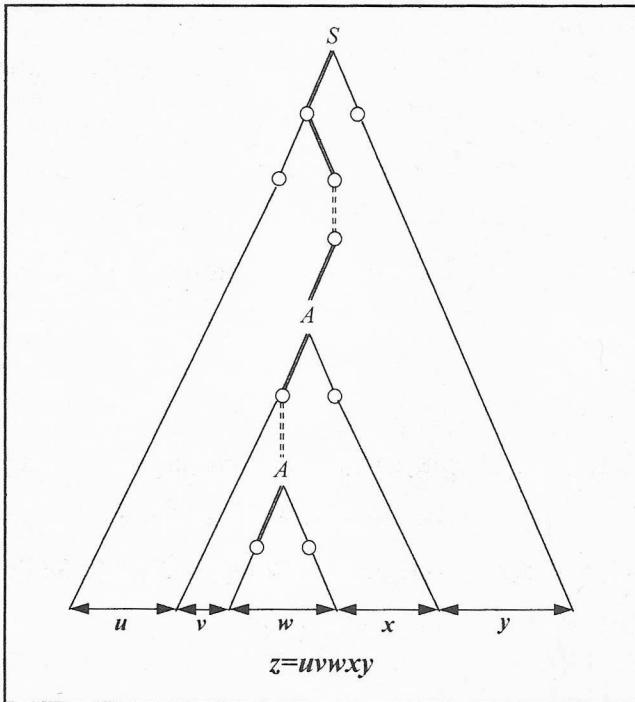


Figura 4.7

Il sottoalbero più in basso con radice in  $A$  genera la sottostringa  $w$ , mentre quello più in alto genera la sottostringa  $vwx$ .

Poiché la  $G$  è C.F., sostituendo il sottoalbero più in alto con quello più in basso, si ottiene ancora una derivazione valida. Il *nuovo albero* genera la *stringa uw*.

Se effettuiamo la sostituzione inversa (il sottoalbero più in basso viene rimpiazzato da quello più in alto), otteniamo un albero di derivazione lecito per la stringa  $uvwxy$ , ossia  $uv^2wx^2y$ . Ripetendo questa sostituzione un numero finito di volte, si ottiene l'insieme di stringhe:

$$\{uv^nwx^ny \mid n \geq 0\}$$

### Proposizione 4.1

Ogni linguaggio C.F. infinito deve contenere almeno un sottoinsieme infinito di stringhe della forma:

$$uv^nwx^ny \quad n \geq 0$$

Formalizziamo ora alcuni risultati connessi con il processo di sostituzione di sottoalberi.

### Lemma 4.1

Sia  $G=(X, V, S, P)$  una grammatica C.F. e supponiamo che:

$$m = \max\{ |w| \mid A \rightarrow v \in P \}$$

Sia  $T_w$  un albero di derivazione per una stringa  $w$  di  $L(G)$ . Se l'altezza di  $T_w$  è al più uguale ad un intero  $j$ , allora:  $|w| \leq m^j$ .

In formule:  $\text{height}(T_w) \leq j \Rightarrow |w| \leq m^j$ .

### Dimostrazione 4.1

La dimostrazione vale per un albero di derivazione che abbia come radice un qualunque simbolo  $NT$ , non necessariamente  $S$ .

Procediamo per induzione su  $j$ .

#### Passo base

$$j = 1$$

$T_w$  rappresenta un'unica produzione (Figura 4.8):

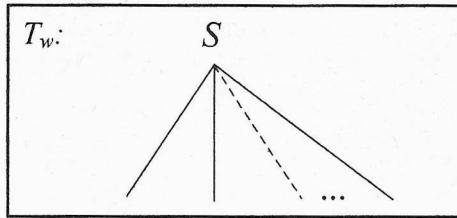


Figura 4.8

dunque la parola generata è composta al più da  $m$  caratteri terminali e si ha:

$$|w| \leq m$$

#### Passo induttivo

Supponiamo che il lemma valga per ogni albero di altezza al più uguale a  $j$  e la cui radice sia un simbolo  $NT$  e dimostriamolo per un albero di altezza  $j+1$ .

Supponiamo che il livello più alto dell'albero rappresenti la produzione  $A \rightarrow v$  (Figura 4.9).

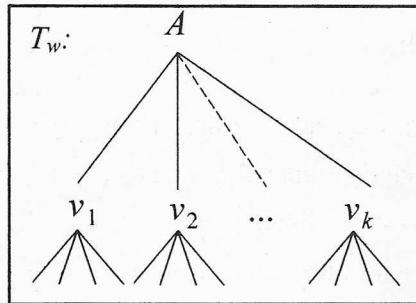


Figura 4.9

dove  $v = v_1 v_2 \dots v_k$ ,  $|v| = k$ ,  $k \leq m$  (il sottoalbero di profondità 1 ha al più  $m$  figli).

Ogni simbolo  $v_i$ ,  $i=1, 2, \dots, k$  di  $v$  può essere radice di un sottoalbero di altezza al più uguale a  $j$ , poiché  $T_w$  ha altezza uguale a  $j+1$  (un  $v_i$  potrebbe anche essere un terminale).

Dunque, per ipotesi di induzione, ciascuno di questi alberi ha al più  $m^j$  foglie.  
 Poiché  $|v| = k \leq m$ , la stringa  $w$ , frontiera dell'albero  $T_w$ , ha lunghezza:

$$|w| \leq \underbrace{m^j + m^j + \dots + m^j}_{|v|=k \text{ volte}} = |v| \cdot m^j = k \cdot m^j \leq m \cdot m^j = m^{j+1} \quad c.v.d.$$

### 4.3 Pumping Lemma per i linguaggi liberi da contesto.

**Teorema 4.2** (Pumping Lemma per i linguaggi liberi o Teorema  $uvwxy$ )

Sia  $L$  un linguaggio libero da contesto. Allora esiste una costante  $p$ , che dipende solo da  $L$ , tale che se  $z$  è una parola di  $L$  di lunghezza maggiore di  $p$  ( $|z| > p$ ), allora  $z$  può essere scritta come  $uvwxy$  in modo tale che:

- (1)  $|vwx| \leq p$ ;
- (2) al più uno tra  $v$  e  $x$  è la parola vuota ( $vx \neq \lambda$ );
- (3)  $\forall i, i \geq 0 : uv^i wx^i y \in L$ .

#### Osservazione 4.2

- (a) Dato un linguaggio generato da una grammatica che non è context-free, non possiamo escludere immediatamente che non esiste una grammatica C.F. che generi lo stesso linguaggio.
- (b) Se un linguaggio infinito non obbedisce al Pumping Lemma, non può essere generato da una grammatica C.F.

Il Pumping Lemma per i linguaggi liberi fornisce una *condizione necessaria* affinché un linguaggio sia libero.

$$L \text{ libero} \Rightarrow \exists p, \dots$$

Dunque può essere utilizzato per dimostrare che un linguaggio non è libero (con una dimostrazione per assurdo - modus tollens).

$$\nexists p, \dots \Rightarrow L \text{ non è libero}$$

### **Dimostrazione 4.2**

Sia  $G=(X,V,S,P)$  una grammatica C.F. che genera  $L$ .

Denotiamo con  $m$  il numero di simboli della più lunga parte destra di una produzione di  $G$ :

$$m = \max\{ |v| \mid A \rightarrow v \in P \}$$

Denotiamo con  $k$  la cardinalità dell'alfabeto nonterminale di  $G$ :

$$k = |V|$$

Poniamo  $p = m^{k+1}$  e sia  $z \in L$ , con  $|z| > p$ .

Per il lemma precedente (Lemma 4.1):

$$\left[ \left( \text{height}(T_z) \leq j \Rightarrow |z| \leq m^j \right) \Leftrightarrow \left( |z| > m^j \Rightarrow \text{height}(T_z) > j \right) \right],$$

se  $|z| > p = m^{k+1}$  allora ogni albero di derivazione per  $z$  deve avere altezza maggiore di  $k+1$ .

Dunque, in ogni albero di derivazione per  $z$  deve esistere almeno un cammino di lunghezza non inferiore a  $k+2$ .

Poiché  $k = |V|$ , almeno due  $NT$  devono comparire duplicati su quel cammino o un  $NT$  deve essere ripetuto 3 o più volte sul cammino.

Senza ledere la generalità della dimostrazione, denotiamo con  $A$  il  $NT$  che compare duplicato per ultimo su quel cammino (si faccia riferimento alla Figura 4.7).

Non vi sono pertanto altri  $NT$  ripetuti almeno due volte al di sotto della  $A$  più in alto (della coppia di  $A$ ).

Questa considerazione implica che il cammino dalla  $A$  più in alto della coppia ad una foglia ha lunghezza al più  $k+1$ .

Indichiamo con  $vwx$  la sottostringa derivata dal sottoalbero avente radice nella  $A$  più alta della coppia, ove  $w$  è la sottostringa derivata dal sottoalbero avente radice nella  $A$  più bassa della coppia.

Dal Lemma 4.1, si ha:

$$|vwx| \leq m^{k+1} = p$$

per cui risulta dimostrata la (1) del Pumping Lemma.

Scriviamo  $z$  nella forma  $uvwxy$ , ed applichiamo il principio di sostituzione di sottoalberi, visto in precedenza.

Se sostituiamo al sottoalbero avente radice nella  $A$  più alta della coppia quello avente radice nella  $A$  più bassa, otteniamo un albero di derivazione per la stringa:

$$uwy = uv^0wx^0y$$

che è la (3) per  $i = 0$ .

Se operiamo la sostituzione inversa, otteniamo un albero di derivazione per la stringa:

$$uvvvwxxxy = uv^2wx^2y$$

che è la (3) per  $i = 2$ .

Se ripetiamo la suddetta sostituzione  $i - 1$  volte, la stringa derivata è:

$$\underbrace{uvv\dots v}_{i \text{ volte}} \underbrace{wx\dots x}_i y = uv^iwx^i y$$

per cui la (3) risulta dimostrata.

La (2) può essere dimostrata per assurdo.

Sia:

$$v = \lambda = x$$

La sostituzione del sottoalbero avente radice nella  $A$  più alta della coppia con quello avente radice nella  $A$  più bassa non provoca alcun cambiamento nella stringa  $z$  derivata dall'intero albero.

Ma tale sostituzione provoca la diminuzione della lunghezza del cammino che dalla  $A$  (in origine quella più in alto) porta ad una foglia.

Dunque, anche il cammino di lunghezza non inferiore a  $k+2$  (su cui compariva la coppia di  $A$ ) nell'albero di derivazione per  $z$  risulta accorciato.

In questo modo abbiamo ottenuto un albero di derivazione per  $z$  con altezza al più uguale a  $k+1$ . Ma questo è assurdo per il Lemma 4.1. *c.v.d.*

#### 4.4 Ambiguità di grammatiche e linguaggi.

Definizione 4.7 (Grammatica ambigua)

Una grammatica  $G$  libera da contesto è *ambigua* se esiste una stringa  $x$  in  $L(G)$  che ha due alberi di derivazione differenti. ■

In modo alternativo,  $G$  è ambigua se esiste una stringa  $x$  in  $L(G)$  che ha due derivazioni sinistre (o destre) distinte.

#### Esempio 4.3

La seguente grammatica libera da contesto:

$$G_2 = (X, V, S, P)$$

$$X = \{a, +\}, \quad V = \{S\}, \quad P = \{S \rightarrow S + S, S \rightarrow a\}$$

è ambigua. Infatti la stringa  $w = a + a + a$  in  $L(G_2)$  ha due differenti alberi di derivazione (Figura 4.10).

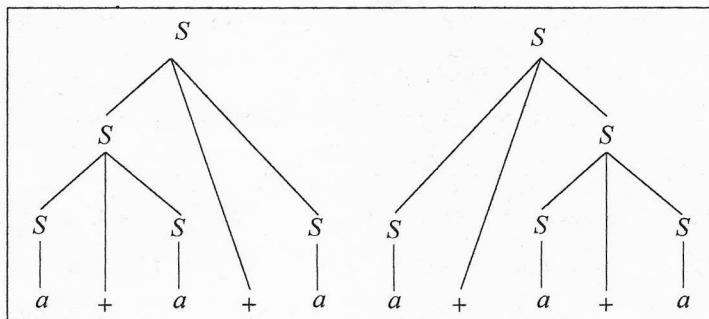
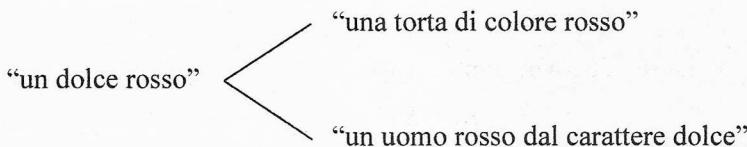
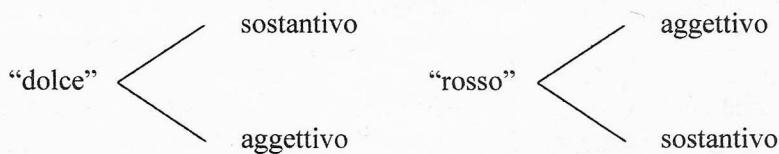


Figura 4.10

Nel linguaggio naturale, l'ambiguità (sintattica) è un fenomeno intrinseco, che si verifica frequentemente.

#### Esempio 4.4

“un dolce rosso”



Agli effetti delle applicazioni ai linguaggi di programmazione, l'*ambiguità* è una proprietà negativa. Infatti, il significato di una frase può essere definito come una funzione del suo albero di derivazione.

Sicché, se esiste più di un albero di derivazione per una frase stessa, essa può avere un significato non univoco.

Si preferisce perciò cercare una grammatica differente che, pur generando lo stesso linguaggio, non sia ambigua.

L'unico vantaggio delle grammatiche ambigue risulta essere, in generale, il minore numero di regole che possono avere rispetto ad una grammatica non ambigua.

#### Esempio 4.5

Il linguaggio  $L(G_2)$  (dell'Esempio 4.3) può essere generato anche dalla seguente grammatica:

$$G_3 = (X, V, S, P)$$

$$X = \{a, +\}, \quad V = \{S\}, \quad P = \{S \rightarrow S + a, S \rightarrow a\}$$

$G_3$  non è ambigua.

Inoltre:  $L(G_2) = L(G_3) = \{aw \mid w \in \{+a\}^*\} = a \cdot \{+a\}^*$ .

Esistono però dei linguaggi, detti *inerentemente ambigui*, per i quali tutte le grammatiche che li generano risultano ambigue.

#### Definizione 4.8 (Linguaggio inerentemente ambiguo)

Un linguaggio  $L$  è *inerentemente ambiguo* se ogni grammatica che lo genera è ambigua.

$$(\forall G) \ L = L(G) : G \text{ ambigua}$$

■

#### Esempio 4.6

Un linguaggio inerentemente ambiguo è:

$$L = \{a^i b^j c^k \mid i, j, k > 0, (i = j \vee j = k)\}$$

Intuitivamente, ci si rende conto di ciò pensando che in ogni grammatica che genera  $L$  devono esistere due diversi insiemi di regole per produrre le stringhe  $a^i b^j c^k$  e le stringhe  $a^j b^i c^l$ . Le stringhe  $a^i b^j c^k$  saranno necessariamente prodotte in due modi distinti, con distinti alberi di derivazione e conseguente ambiguità.

Per esercizio: determinare una grammatica che genera  $L$  e provare questo risultato.

#### Esempio 4.7 (Linguaggi di programmazione)

Si consideri la seguente grammatica  $G = (X, V, S, P)$ :

$$X = \{\underline{\text{if}}, \underline{\text{then}}, \underline{\text{else}}, \text{a}, \text{b}, \text{p}, \text{q}\}$$

$$V = \{S, C\}$$

$$P = \{S \rightarrow \underline{\text{if}} \ C \ \underline{\text{then}} \ S \ \underline{\text{else}} \ S \mid \underline{\text{if}} \ C \ \underline{\text{then}} \ S \mid \text{a} \mid \text{b}, C \rightarrow \text{p} \mid \text{q}\}$$

$G$  è ambigua. Infatti la stringa:

$$w = \underline{\text{if}} \ p \ \underline{\text{then}} \ \underline{\text{if}} \ q \ \underline{\text{then}} \ a \ \underline{\text{else}} \ b$$

può essere generata in due modi, descritti in Figura 4.11 e 4.12:

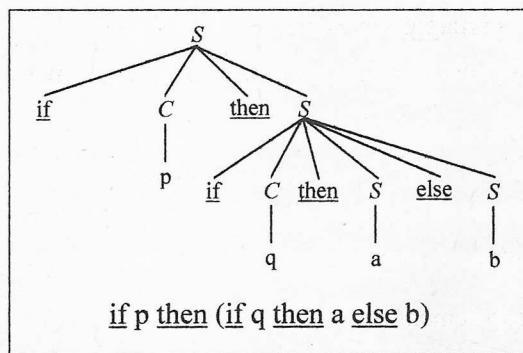


Figura 4.11

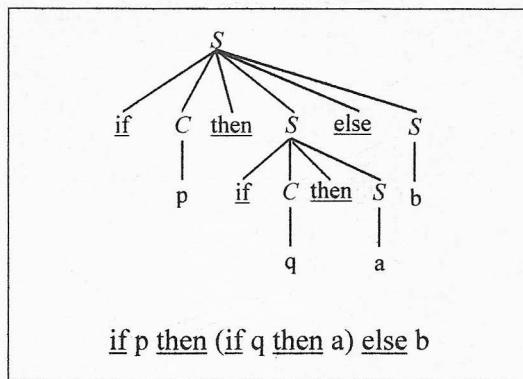


Figura 4.12

Per rendere non ambigua  $G$  si usa solitamente la convenzione di associare ogni istruzione else con l'istruzione if più vicina.

La grammatica che riflette questa convenzione è la seguente:

$$G' = (X, V, S, P)$$

$$X = \{\underline{\text{if}}, \underline{\text{then}}, \underline{\text{else}}, \text{a}, \text{b}, \text{p}, \text{q}\}$$

$$V = \{S, S_1, S_2, C, T\}$$

$$P = \{S \rightarrow S_1 \mid S_2,$$

$$S_1 \rightarrow \underline{\text{if}} \ C \ \underline{\text{then}} \ S_1 \ \underline{\text{else}} \ S_2 \mid T,$$

$$S_2 \rightarrow \underline{\text{if}} \ C \ \underline{\text{then}} \ S \mid \underline{\text{if}} \ C \ \underline{\text{then}} \ S_1 \ \underline{\text{else}} \ S_2 \mid T$$

$$C \rightarrow \text{p} \mid \text{q},$$

$$T \rightarrow \text{a} \mid \text{b}\}$$

In  $G'$  la stringa  $w = \underline{\text{if}} \ p \ \underline{\text{then}} \ \underline{\text{if}} \ q \ \underline{\text{then}} \ a \ \underline{\text{else}} \ b$  ha un unico albero di derivazione (Figura 4.13):

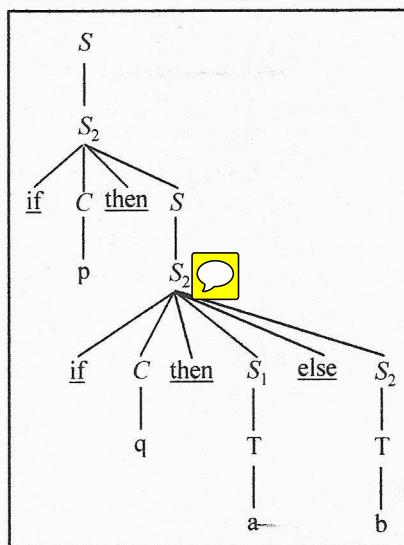


Figura 4.13

Ma è proprio vero che l'albero di derivazione per  $w$  è unico?

Si tenga conto che l'ambiguità di un linguaggio prescinde dal nome delle variabili ma dipende dalla struttura.

#### 4.5 Esercizi.

##### Esercizio 4.1

Sia dato il linguaggio:

$$L = \{a^t \mid t \text{ numero primo}\}$$

Stabilire se  $L$  è libero e giustificare formalmente la risposta.

Analizziamo le parole che costituiscono  $L$ .

$$L = \{a, a^2, a^3, a^5, a^7, a^{11}, a^{13}, a^{17}, \dots\}$$

$ a  = 1$	$ a^{13}  = 13$	$\dots$
$ a^2  = 2$	$ a^{17}  = 17$	$ a^n  = n$
$ a^3  = 3$	$ a^{19}  = 19$	$\dots$
$ a^5  = 5$	$ a^{23}  = 23$	
$ a^7  = 7$	$ a^{29}  = 29$	
$ a^{11}  = 11$	$ a^{31}  = 31$	

La lunghezza delle parole di  $L$  non cresce in maniera costante, né è possibile determinare un sottoinsieme infinito di  $L$  le cui parole hanno lunghezze che crescono in maniera costante. Dunque  $L$  non è un linguaggio libero da contesto.

Proviamo formalmente ciò.

Supponiamo per assurdo che  $L$  sia libero da contesto.

Per il Pumping Lemma per i linguaggi liberi deve esistere una costante intera  $p$ , dipendente unicamente da  $L$ , tale che:

$$\forall z \ z \in L, |z| > p \Rightarrow z = uvwxy$$

ed inoltre valgono le seguenti:

- (1)  $|vwx| \leq p$ ;
- (2)  $vx \neq \lambda$ ;
- (3)  $\forall i, i \geq 0 : uv^iwx^i y \in L$

Consideriamo la seguente parola di  $L$ :

$$z = a^{f(p+1)}$$

ove  $f$  è una funzione definita in  $\mathbb{N}$  ed a valori in  $\mathbb{N}$ , che associa ad ogni intero  $t$  il  $t$ -esimo numero primo.

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

$\forall t \in \mathbb{N} : f(t) = t\text{-esimo numero primo.}$

Pertanto  $z$  è la parola di  $L$  ottenuta concatenando un numero di  $a$  pari al  $p+1$ -esimo numero primo.

È immediato osservare che valgono le seguenti relazioni:

$$\begin{aligned}\forall t \in \mathbb{N} : f(t) &\geq t \quad (f(t) > t \quad \forall t, t > 3) \\ f(t+1) &> t \\ f(t+1) &> f(t)\end{aligned}$$

Pertanto risulta:

$$|z| = f(p+1) > p$$

e  $z$  può essere scritta nella forma  $z = uvwxy$  in modo tale che valgano le (1), (2) e (3).

Consideriamo la parola:

$$uv^{f(p+1)+1}wx^{f(p+1)+1}y$$

Per la (3), con  $i = f(p+1)+1$ , si deve verificare che tale parola è un elemento di  $L$ .

D'altra parte, se valutiamo la lunghezza della parola  $uv^{f(p+1)+1}wx^{f(p+1)+1}y$  si ha:

$$\left|uv^{f(p+1)+1}wx^{f(p+1)+1}y\right| = \left|uvv^{f(p+1)}wx^{f(p+1)}y\right| = |uvwxy| + |v^{f(p+1)}| + |x^{f(p+1)}| \quad (\text{a})$$

Poiché:

$$|\alpha'| + |\beta'| = i \cdot |\alpha| + i \cdot |\beta| = i \cdot (|\alpha| + |\beta|) = i \cdot |\alpha\beta|$$

la (a) può essere scritta nella forma:

$$|z| + f(p+1) \cdot |vx| = f(p+1) + f(p+1) \cdot |vx| = f(p+1) (1 + |vx|) \quad (\text{b})$$

Ora denotiamo con  $k$  la lunghezza della stringa  $vx$ :

$$k = |vx|$$

Per la (2),  $k > 0$  e per la (1) si ha che  $k \leq |vwx| \leq p$ .

Dunque:

$$0 < k \leq p$$

Ma allora la (b) diventa:

$$f(p+1)(1 + |vx|) = f(p+1) \cdot (1 + k) \quad (\text{c})$$

e, per la (3), deve risultare che:

$$f(p+1) \cdot (1 + k)$$

sia un numero primo. Ma questo è un assurdo, poiché  $k+1 \neq 1$  per ogni  $k$ .

Dunque:

$$uv^{f(p+1)+1}wx^{f(p+1)+1}y \notin L$$

contro la (3).

L'assurdo è derivato dall'aver supposto che  $L$  fosse libero da contesto. Possiamo concludere che  $L$  non è libero da contesto, in quanto  $L$  è infinito ma non verifica il Pumping Lemma per i linguaggi liberi.

Inoltre, se  $L$  fosse libero da contesto, il seguente sarebbe un teorema dell'aritmetica formale (*teoria assiomatica dei numeri*):

$$\begin{aligned} t \text{ numero primo} &\Rightarrow t + k \cdot i \text{ numero primo} & 0 < k \leq p, \forall i, i \geq 0 \\ t \text{ numero primo} &\Rightarrow t + 2k \cdot i \text{ numero primo} \end{aligned}$$

Il teorema precedente è paleamente falso. Esso sarebbe un corollario del Pumping Lemma per i linguaggi liberi, nell'ipotesi che  $L$  fosse libero. Il perché è semplice.

È sufficiente considerare

$$z = a^t, t \text{ numero primo e } t > p.$$

Consideriamo le parole:

$$uv^{i+1}wx^{i+1}y \quad \forall i, i \geq 0$$

Per la (3),

$$uv^{i+1}wx^{i+1}y \in L \quad \forall i, i \geq 0$$

Ma si ha:

$$|uv^{i+1}wx^{i+1}y| = |uvwxy| + |v^i x^i| = t + i \cdot |vx| \quad (\text{a}')$$

Posto  $k = |vx|$  e  $0 < k \leq p$ , per la (1) e la (2) si ha che (a') diventa:

$$t + i \cdot |vx| = t + i \cdot k \quad \forall i, i \geq 0 \quad (\text{b}')$$

e  $t + i \cdot k$  deve essere un numero primo per un certo  $k$ ,  $0 < k \leq p$ , e per tutti gli interi non negativi  $i$  ( $i \geq 0$ ). Ma questo è assurdo.

### Esercizio 4.2

1) Determinare la grammatica che genera il seguente linguaggio:

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

e dimostrare la correttezza di tale grammatica.

2) Di che tipo è la grammatica che genera  $L$ ?

3) Applicare il Pumping Lemma per dimostrare che  $L$  non è libero.

1)

$$G = (X, V, S, P)$$

$$\begin{aligned} X &= \{a, b, c\} & V &= \{S, A, B, C\} & P = & \left. \begin{aligned} S &\xrightarrow{(1)} aBC, \quad S \xrightarrow{(2)} aSBC \\ &\xrightarrow{(3)} CB \xrightarrow{(4)} BC, \quad aB \xrightarrow{(4)} ab \\ &\xrightarrow{(5)} bB \xrightarrow{(6)} bb, \quad bC \xrightarrow{(6)} bc \\ &\xrightarrow{(7)} cC \xrightarrow{(7)} cc \end{aligned} \right\} \end{aligned}$$

Vediamo come ricavare le produzioni di  $G$ .

Una derivazione della parola di lunghezza minima in  $L$ ,  $w = abc$ , è la seguente:

$$S \xrightarrow{(1)} aBC \xrightarrow{(4)} abC \xrightarrow{(6)} abc$$

Le derivazioni (sinistre) delle altre parole in  $L$  sono del tipo:

$$S \xrightarrow{(2)} aSBC \xrightarrow{(2)} aaSBCBC$$

La struttura della stringa  $aaSBCBC$  non è quella desiderata (che contraddistingue le parole di  $L$ ).

È necessaria una produzione che effettui uno scambio di nonterminali:

$$CB \rightarrow BC^1$$

Grazie a questa regola di produzione si ha:

$$S \xrightarrow{(2)} aSBC \xrightarrow{(1)} aaBCBC \xrightarrow{(3)} aaBBCC \xrightarrow{(4)} aabBCC \xrightarrow{(5)} aabbCC \xrightarrow{(6)} aabbC \xrightarrow{(7)} aabbcc$$

Le derivazioni di parole più lunghe in  $L$  non hanno bisogno di ulteriori produzioni:

$$\begin{aligned} S &\xrightarrow{(2)} aSBC \xrightarrow{(2)} \overset{2}{aaa}SBCBC \xrightarrow{(1)} \overset{2}{aaaa}BCBCBC \xrightarrow{(3)} \overset{2}{aaaa}BBCCBC \xrightarrow{(3)} \\ &\xrightarrow{(3)} \overset{2}{aaaa}BBCBC \xrightarrow{(3)} a^4B^3CCCBC \xrightarrow{(3)} a^4B^3CCBCC \xrightarrow{(3)} a^4B^4C^4 \xrightarrow{(4)} \\ &\xrightarrow{(4)} a^4bB^3C^4 \xrightarrow{(5)} a^4b^2B^2C^4 \xrightarrow{(5)} \overset{2}{a^4b^4C^4} \xrightarrow{(6)} a^4b^4cC^3 \xrightarrow{(7)} a^4b^4c^2C^2 \xrightarrow{(7)} \overset{2}{a^4b^4c^4} \end{aligned}$$

- 2)  $G$  è una grammatica monotona. Per il teorema di equivalenza delle grammatiche monotone e contestuali (Teorema 3.2), esiste una grammatica contestuale  $G'$  equivalente a  $G$ .
- 3) Supponiamo, per assurdo, che  $L$  sia libero da contesto. Per il Pumping Lemma sui linguaggi liberi, esiste una costante  $p$  tale che:

$$\forall z \quad z \in L, \quad |z| > p \quad \Rightarrow \quad z = uvwxy$$

<sup>1</sup> È una produzione monotona.

ed inoltre valgono le seguenti:

- (1)  $|vwx| \leq p$ ;
- (2)  $vx \neq \lambda$ ;
- (3)  $\forall i, i \geq 0 : uv^iwx^i y \in L$

Consideriamo una parola in  $L$ :

$$z = a^p b^p c^p.$$

Il Pumping Lemma può essere applicato a tale parola poiché  $|z| = 3p > p$  e dunque  $z$  può essere scritta nella forma  $z = uvwxy$ , in modo tale che:

$$|vwx| \leq p$$

Poiché la stringa  $vwx$  ha lunghezza al più uguale a  $p$ , si hanno le seguenti possibilità:

- (i)  $vwx$  è formata da sole  $a$ , cioè è del tipo  $vwx = a^k$ ,  $0 < k \leq p$ ;
- (ii)  $vwx$  è formata da sole  $b$ , cioè è del tipo  $vwx = b^k$ ,  $0 < k \leq p$ ;
- (iii)  $vwx$  è formata da sole  $c$ , cioè è del tipo  $vwx = c^k$ ,  $0 < k \leq p$ ;
- (iv)  $vwx$  è a cavallo tra  $a$  e  $b$ , cioè è del tipo  $vwx = a^k b^r$ ,  $0 < k + r \leq p$ , e  $k, r > 0$ ;
- (v)  $vwx$  è a cavallo tra  $b$  e  $c$ , cioè è del tipo  $vwx = b^k c^r$ ,  $0 < k + r \leq p$ , e  $k, r > 0$ .

È immediato osservare che  $vwx$  non può essere formata da  $a$ ,  $b$  e  $c$  (ossia non può essere contemporaneamente a cavallo tra  $a$  e  $b$  e tra  $b$  e  $c$ ), perché non è sufficientemente lunga.

Consideriamo la stringa (*pompata*):

$$uv^2wx^2y$$

per ognuno dei casi (i) - (v).

Per la (3) del Pumping Lemma sui linguaggi liberi, si dovrebbe avere:

$$uv^2wx^2y \in L$$

Ma nel caso:

- (i) aggiungiamo almeno una  $a$ , ed al più  $p$   $a$ ;  
 $uv^2wx^2y = a^{p+t}b^pc^p$ ,  $0 < t \leq p$ ;
- (ii) aggiungiamo almeno una  $b$ , ed al più  $p$   $b$ ;  
 $uv^2wx^2y = a^pb^{p+t}c^p$ ,  $0 < t \leq p$ ;
- (iii) aggiungiamo almeno una  $c$ , ed al più  $p$   $c$ ;  
 $uv^2wx^2y = a^pb^pc^{p+t}$ ,  $0 < t \leq p$ ;
- (iv) per la (2) del Pumping Lemma, si hanno le seguenti possibilità:

- (iv.a)  $v \neq \lambda$ ,  $x \neq \lambda$ ;
- (iv.b)  $v \neq \lambda$ ,  $x = \lambda$ ;
- (iv.c)  $v = \lambda$ ,  $x \neq \lambda$ .

Osserviamo preliminarmente che, se  $v \neq \lambda$ , allora  $v$  è costituita da sole  $a$ . Infatti, se  $v$  fosse del tipo  $v = a^k b^{r'}$ , con  $0 < r' \leq r$ , si avrebbe  $uv^2wx^2y = a^{p-k}a^kb^{r'}a^kb^{r'}b^sc^p \notin L$ , con  $p \leq s \leq 2(r - r') + p - r$ .

Analogamente, se  $x \neq \lambda$ , allora  $x$  è costituita da sole  $b$ . Infatti, se  $x$  fosse del tipo  $v = a^{k'}b^r$ , con  $0 < k' \leq k$ , si avrebbe  $uv^2wx^2y = a^sa^{k'}b^ra^kb^rc^p \notin L$ , con  $p \leq s \leq 2(k - k') + p - k$ .

Per cui:

- (iv.a) se  $v \neq \lambda$ ,  $x \neq \lambda$ , per l'osservazione precedente  $v = a^{k'}$ , con  $0 < k' \leq k$  e  $x = b^{r'}$ ,  $0 < r' \leq r$  e si ha che  $uv^2wx^2y = a^{p+k'}b^{p+r'}c^p \notin L$ , poiché  $k', r' > 0$ ;

(iv.b) se  $v \neq \lambda$ ,  $x = \lambda$ , si ha  $v = a^{k'}$ , con  $0 < k' \leq k$  e  
 $uv^2wx^2y = a^{p+k'}b^pc^p \notin L$ , poiché  $k' > 0$ ;

(iv.c) se  $v = \lambda$ ,  $x \neq \lambda$ , si ha  $x = b^{r'}$ , con  $0 < r' \leq r$  e  
 $uv^2wx^2y = a^pb^{p+r'}c^p \notin L$ , poiché  $r' > 0$ ;

(v) Si lascia per esercizio.

In ciascuno dei casi (i) - (v)  $uv^2wx^2y \notin L$ .

Assurdo. Ne segue che  $L$  non è un linguaggio libero da contesto, poiché è un linguaggio infinito per il quale non vale il Pumping Lemma.

### Esercizio 4.3

Applicare il Pumping Lemma per dimostrare che il seguente linguaggio  $L$  non è libero da contesto:

$$L = \{a^{n^2} \mid n \geq 0\}$$

Analizziamo le parole che costituiscono  $L$ .

$$L = \{\lambda, a, a^4, a^9, a^{16}, \dots\}$$

Supponiamo, per assurdo, che  $L$  sia libero da contesto.

Per il Pumping Lemma sui linguaggi liberi, esiste un numero naturale  $p$ , dipendente solo da  $L$ , tale che se  $z \in L$  e  $|z| > p$ , allora:

$$z = uvwxy$$

- (1)  $|vwx| \leq p$ ;
- (2)  $vx \neq \lambda$ ;
- (3)  $uv^iwx^i y \in L, \forall i \geq 0$ .

Consideriamo la parola:

$$z = a^{p^2}$$

$z \in L$  ed inoltre  $|z| = p^2 > p$ .

Per il Pumping Lemma, possiamo scrivere:

$$z = uvwxy \text{ ove } |vwx| \leq p.$$

Consideriamo la stringa:

$$uv^2wx^2y$$

Per la (3) del Pumping Lemma, si deve avere:

$$uv^2wx^2y \in L$$

Ma:

$$|uv^2wx^2y| = |uvwxy| + |vx| \leq |uvwxy| + |vwx| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

Dunque:

$$uv^2wx^2y \notin L$$

Assurdo. Ne segue che  $L$  non è un linguaggio libero da contesto.

#### Esercizio 4.4

Applicare il Pumping Lemma per dimostrare che il seguente linguaggio  $L$  non è libero da contesto:

$$L = \{a^i b^j \mid i = 2^j, i, j \geq 0\}$$

Analizziamo le parole che costituiscono  $L$ .

$$L = \{a^{2^j} b^j \mid j \geq 0\} = \{a, a^2 b, a^4 b^2, a^8 b^3, a^{16} b^4, \dots\}$$

Per assurdo, supponiamo  $L$  libero da contesto. Vale, dunque, per  $L$  il Pumping Lemma sui linguaggi liberi.

Dunque, si ha:

$\exists p \in \mathbb{N}, p$  dipendente solo da  $L$ , tale che se  $z \in L, |z| > p$ , allora:

$L = \{a^{2^p} b^p, a^{2^{p+1}} b^{p+1}, \dots\}$

$$z = uvwxy$$

- (1)  $|vwx| \leq p$ ;
- (2)  $vx \neq \lambda$ ;
- (3)  $uv^iwx^iy \in L, \forall i \geq 0$ .

Consideriamo la parola:

$$z = a^{2^p} b^p$$

$z \in L$  ed inoltre  $|z| = 2^p + p > p$

Per il Pumping Lemma, possiamo scrivere:

$$z = uvwxy$$

ove  $|vwx| \leq p$ . Consideriamo la stringa:

$$uv^2wx^2y$$

Per la (3) del Pumping Lemma, si deve avere:

$$uv^2wx^2y \in L$$

Ma:

$$\begin{aligned} 2^p + p < |uv^2wx^2y| &= |uvwxy| + |vx| \leq |uvwxy| + |vwx| \leq \underbrace{2^p}_{?} + p + p \leq 2^p + 2^p + p = \\ &= 2 \cdot 2^p + p = 2^{p+1} + p < 2^{p+1} + p + 1 \end{aligned}$$

Dunque la stringa pompata  $uv^2wx^2y$  non è del tipo  $a^{2^j} b^j$ , ossia:

$$uv^2wx^2y \notin L$$

Assurdo. Ne segue che  $L$  non è un linguaggio libero da contesto.

### Esercizio 4.5

Dimostrare che il seguente linguaggio

$$L = \{a^k b^r \mid k > 0, r > k^2\}$$

non è libero.

Analizziamo le parole che costituiscono  $L$ :

$$L = \left\{ ab^2, ab^3, ab^4, \dots \right. \\ \left. a^2b^5, a^2b^6, a^2b^7, \dots \right. \\ \left. a^3b^{10}, a^3b^{11}, a^3b^{12}, \dots \right\}$$

Supponiamo, per assurdo, che  $L$  sia libero da contesto.

Per il Pumping Lemma sui linguaggi liberi, esiste un numero naturale  $p$ , dipendente da  $L$ , tale che, se  $z \in L$  e  $|z| > p$  allora:

$$z = uvwxy$$

- (1)  $|vwx| \leq p$ ;
- (2)  $vx \neq \lambda$ ;
- (3)  $uv^iwx^i y \in L, \forall i \geq 0$ .

Consideriamo la parola:

$$z = a^p b^{p^2+1}$$

$z \in L$  ed inoltre  $|z| = p + p^2 + 1 > p$ .

Per il Pumping Lemma, possiamo scrivere:

$$z = uvwxy.$$

Per la sottostringa  $vwx$  si hanno le seguenti possibilità:

- (i)  $vwx$  è formata da sole  $a$ ;
- (ii)  $vwx$  è formata da sole  $b$ ;
- (iii)  $vwx$  è a cavallo tra  $a$  e  $b$ .

Nel caso (i), per la (1) e la (2) del Pumping Lemma, si ha:

$$0 < |vwx| \leq p \quad \text{e} \quad 0 < |vx| \leq p$$

Dunque  $vwx$  è formata da almeno una  $a$  ed al più  $p$   $a$ . Consideriamo la parola:

$$uv^2wx^2y.$$

Tale parola ha almeno una ed al più  $p$   $a$  in più di  $z$ . Dunque,  $uv^2wx^2y$  ha almeno  $p+1$   $a$  (ed al più  $2p$   $a$ ), mentre il numero delle  $b$  non è mutato. Ossia, denotato con  $\#(x)$  il numero di occorrenze del simbolo  $x$  nella parola  $uv^2wx^2y$ :

$$p+1 \leq \#(a) \leq 2p \text{ e } \#(b) = p^2 + 1$$

Ma questo vuol dire che:

$$\#(b) = p^2 + 1 < (p+1)^2 \leq \#(a)^2 \leq 4p^2$$

Dunque  $r < k^2$  e  $uv^2wx^2y \notin L$ .

Nel caso (ii),  $vwx$  è formata da almeno una  $b$  ed al più  $p$   $b$ . Consideriamo la parola:

$$uv^0wx^0y.$$

Tale parola ha almeno una ed al più  $p$   $b$  in meno di  $z$ , in quanto per la (1) e la (2) del Pumping Lemma:

$$0 < |vwx| \leq p.$$

Dunque,  $uv^0wx^0y$  ha al più  $p^2$   $b$  (ed almeno  $p^2 + 1 - p$   $b$ ), mentre il numero delle  $a$  non è cambiato. Quindi si ha che:

$$p^2 - p + 1 \leq \#(b) \leq p^2 \text{ e } \#(a) = p$$

da cui:

$$\#(b) \leq \#(a)^2 = p^2.$$

Dunque si ha  $r \leq k^2$  e  $uv^0wx^0y \notin L$ .

Nel caso (iii),  $vwx$  è formata sia da  $a$  sia da  $b$  e

$$0 < \#(a) + \#(b) \leq p$$

poiché per le (1) e (2) del Pumping Lemma:

$$0 < |vwx| \leq p$$

Se  $v \neq \lambda$ , allora  $v$  contiene solo  $a$  (altrimenti  $v^i$ ,  $i > 1$ , conterrebbe delle  $a$  alternate a delle  $b$ ). Analogamente, se  $x \neq \lambda$  allora  $x$  contiene solo  $b$ .

Dunque, ci sono tre possibilità:

(iii.a)  $v \neq \lambda$ ,  $x = \lambda$ ;

(iii.b)  $v = \lambda$ ,  $x \neq \lambda$ ;

(iii.c)  $v \neq \lambda$ ,  $x \neq \lambda$ .

Nel caso (iii.a), consideriamo la parola  $uv^2wx^2y$ . Come nel caso (i), tale parola ha almeno una  $a$  in più, rispetto a  $z$  (ed al più  $p-1$   $a$  in più di  $z$ , poiché ci deve essere almeno una  $b$  in  $vwx$  e questa deve essere necessariamente in  $w$ ).

Il numero delle  $b$  in  $uv^2wx^2y$  non è mutato. Si ha che:

$$p+1 \leq \#(a) \leq p+p-1 = 2p-1 \text{ e } \#(b) = p^2 + 1$$

e quindi poiché:

$$\#(b) = p^2 + 1 < (p+1)^2 \leq \#(a)^2$$

si ha che:

$$uv^2wx^2y \notin L.$$

Nel caso (iii.b), consideriamo la parola  $uv^0wx^0y$ . Come nel caso (ii), tale parola ha almeno una  $b$  in meno rispetto a  $z$  (ed al più  $p-1$   $b$  in meno di  $z$ , poiché ci deve essere almeno una  $a$  in  $vwx$  e questa deve essere necessariamente in  $w$ ), mentre il numero delle  $a$  in  $uv^0wx^0y$  non cambia. Si ha che:

$$p^2 + 1 - (p-1) \leq \#(b) \leq p^2 \text{ e } \#(a) = p$$

e quindi poiché:

$$\#(b) \leq \#(a)^2 = p^2$$

si ha che:

$$uv^0wx^0y \notin L.$$

Nel caso (iii.c), consideriamo la parola  $uv^2wx^2y$ .

Poiché  $v \neq \lambda$ ,  $v$  contiene almeno una  $a$ . Dunque  $v^2$  contiene almeno due  $a$  e  $uv^2wx^2y$  contiene almeno  $p+1$   $a$ .

Ma:

$$\begin{aligned} |uv^2wx^2y| &= |uvwxy| + |vx| = |z| + |vx| \leq |z| + |vwx| \leq p + p^2 + 1 + p = p^2 + 2p + 1 = \\ &= (p+1)^2 < p+1+(p+1)^2+1 \end{aligned}$$

Dunque  $uv^2wx^2y$  ha almeno  $p+1$   $a$ , ma la sua lunghezza è strettamente minore della lunghezza della parola di  $L$  con almeno  $p+1$   $a$  e di lunghezza minima. Ne consegue che:

$$uv^2wx^2y \notin L.$$

In ciascuno dei casi (e sottocasi di) (i), (ii) e (iii) risulta violata la (3) del Pumping Lemma per i linguaggi liberi.

Assurdo.

L'assurdo deriva dall'aver assunto  $L$  libero da contesto. Dunque  $L$  non è un linguaggio libero.

## 5. Grammatiche e macchine.

### 5.1 Classificazione delle grammatiche secondo Chomsky.

Definizione 5.1 (Gerarchia di Chomsky (1956, 1959))

Sia  $G=(X,V,S,P)$  una grammatica. Dalla definizione di grammatica, si ha:

$$P = \{v \rightarrow w \mid v \in (X \cup V)^+ \text{ e } v \text{ contiene almeno un } NT, w \in (X \cup V)^*\}.$$

A seconda delle restrizioni imposte sulle regole di produzione, si distinguono le varie classi di grammatiche.

**Tipo '0'** Quando le stringhe che appaiono nella produzione  $v \rightarrow w$  non sono soggette ad alcuna limitazione.

**Tipo '1'** **Dipendente da contesto**, quando le produzioni sono limitate alle forme:

$$(1) \quad yAz \rightarrow ywz,$$

con  $A \in V$ ,  $y,z \in (X \cup V)^*$ ,  $w \in (X \cup V)^+$ ;

$$(2) \quad S \rightarrow \lambda,$$

purché  $S$  non compaia nella parte destra di alcuna produzione.

**Tipo '2'** **Libera da contesto**, quando le produzioni sono limitate alla forma:

$$v \rightarrow w \text{ con } v \in V.$$

**Tipo '3'** **Lineare destra**, quando le produzioni sono limitate alle forme:

$$(1) \quad A \rightarrow bC \text{ con } A,C \in V \text{ e } b \in X;$$

$$(2) \quad A \rightarrow b \text{ con } A \in V \text{ e } b \in X \cup \{\lambda\}$$

Una grammatica di tipo ‘3’ è detta lineare destra perché il  $NT$ , se c’è, compare a destra (nella parte destra della produzione). Un linguaggio generato da una tale grammatica è detto *di tipo ‘3’* o *lineare a destra*. ■

### Esempi di linguaggi di tipo ‘3’

- Consideriamo la grammatica  $G_1$ :

$$S \rightarrow \lambda \mid 0 \mid 0S \mid 1 \mid 1S$$

$G_1$  è lineare destra.  $L(G_1) = \{0,1\}^*$  (l’insieme di tutte le stringhe binarie).

- Consideriamo la grammatica  $G_2$ :

$$\begin{aligned} S &\rightarrow \lambda \mid 0S \mid 1T \\ T &\rightarrow 0T \mid 1S \end{aligned}$$

$G_2$  è lineare destra.  $L(G_2) = \{w \in \{0,1\}^* \mid w \text{ ha un numero pari di } 1\}$

## 5.2 Teorema della Gerarchia.

Dimostriamo formalmente che le quattro classi di linguaggi viste costituiscono una gerarchia (Figura 5.1).

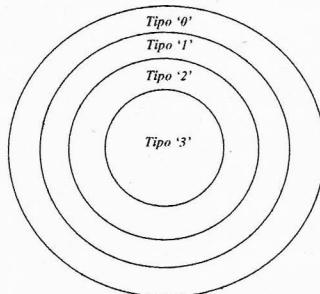


Figura 5.1

Vale infatti il seguente:

### **Teorema 5.1** (della Gerarchia)

Denotiamo con  $\mathcal{L}_i$  il seguente insieme:

$$\mathcal{L}_i = \{L \subset X^* \mid L = L(G), G \text{ di tipo } i\} \text{ (classe dei linguaggi di tipo } i\text{).}$$

La gerarchia di Chomsky è una gerarchia in senso stretto di classi di linguaggi:



$$\mathcal{L}_3 \subset_{\neq} \mathcal{L}_2 \subset_{\neq} \mathcal{L}_1 \subset_{\neq} \mathcal{L}_0$$

### **Dimostrazione 5.1**

1)  $\mathcal{L}_3 \subset_{\neq} \mathcal{L}_2$

Per dimostrare che la classe di linguaggi  $\mathcal{L}_3$  è *inclusa propriamente* nella classe di linguaggi  $\mathcal{L}_2$  si deve dimostrare che ogni linguaggio di tipo ‘3’ può essere generato da una grammatica di tipo ‘2’ e che esiste almeno un linguaggio C.F. (di tipo ‘2’) che non può essere generato da una grammatica di tipo ‘3’.

$\mathcal{L}_3 \subset \mathcal{L}_2$  discende direttamente dalle definizioni di linguaggio di tipo ‘3’ e di grammatica di tipo ‘2’. Infatti, si osserva facilmente che ogni grammatica di tipo ‘3’ è anche una grammatica di tipo ‘2’.

$\mathcal{L}_3 \neq \mathcal{L}_2$ : posponiamo questa dimostrazione.

Mostreremo (si veda Esercizio 7.8) che:  $L = \{a^k b^k \mid k > 0\}$  non è di tipo ‘3’ (abbiamo già determinato una grammatica di tipo ‘2’ che genera  $L$ ).

2)  $\mathcal{L}_2 \subset_{\neq} \mathcal{L}_1$

Abbiamo già osservato che le produzioni C.F. sono un caso particolare delle produzioni C.S. con l'unica eccezione rappresentata dalle produzioni:

$$A \rightarrow \lambda, \quad A \neq S$$

che sono C.F. ma **non** C.S.

Dunque:

$$\forall L : L \in \mathcal{L}_2 \stackrel{\text{def}}{\iff} \exists G, G \text{ è C.F.} : L = L(G)$$

Se  $A \rightarrow \lambda, A \in V \setminus \{S\}$  non è una produzione di  $G$ , allora  $G$  è anche C.S. (di tipo '1') e l'asserto è dimostrato. Il problema sorge se  $G$  ha almeno una  $\lambda$ -produzione. In tal caso, ci avvaliamo del seguente risultato:

### 5.3 Lemma della stringa vuota.

#### Lemma 5.2 (della stringa vuota)

Sia  $G = (X, V, S, P)$  una grammatica C.F. con almeno una  $\lambda$ -produzione. Allora esiste una grammatica C.F.  $G'$  tale che:

- i)  $L(G) = L(G')$  ( $G'$  è equivalente a  $G$ );
- ii) se  $\lambda \notin L(G)$  allora in  $G'$  non esistono produzioni del tipo  $A \rightarrow \lambda$ ;
- iii) se  $\lambda \in L(G)$  allora in  $G'$  esiste un'unica produzione  $S' \rightarrow \lambda$ , ove  $S'$  è il simbolo iniziale di  $G'$  ed  $S'$  non compare nella parte destra di alcuna produzione di  $G'$ .

#### Dimostrazione 5.2

Si rimanda la dimostrazione di questo lemma alla trattazione della forma normale di Greibach per le grammatiche libere da contesto (Paragrafo 8.4).

Riprendiamo la dimostrazione di  $\mathcal{L}_2 \subset \mathcal{L}_1$

$$\forall L : L \in \mathcal{L}_2 \stackrel{\text{def}}{\iff} \exists G, G \text{ è di tipo '2'} : L = L(G)$$

Se  $G$  ha almeno una  $\lambda$ -produzione, utilizziamo il Lemma della stringa vuota per determinare una grammatica C.F.  $G'$  equivalente a  $G$ , ma priva di  $\lambda$ -produzioni (al più, in  $G'$  compare la produzione  $S' \rightarrow \lambda$ , ed  $S'$  non compare nella parte destra di alcuna produzione di  $G'$ ).  $G'$  è di tipo ‘1’. Questo dimostra che  $\mathcal{L}_2 \subseteq \mathcal{L}_1$ .

Dimostriamo che  $\mathcal{L}_2 \neq \mathcal{L}_1$ .

Dall’Esercizio 4.2, segue immediatamente che:

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

è di tipo ‘1’ ma non di tipo ‘2’. Si osservi che, per assereire che  $L$  è di tipo ‘1’, ci siamo avvalsi del teorema che stabilisce l’equivalenza delle classi di linguaggi contestuali e monotoni.

3)  $\mathcal{L}_1 \underset{\neq}{\subseteq} \mathcal{L}_0$

Non lo dimostriamo formalmente. La dimostrazione comporta la conoscenza degli automi limitati lineari e delle macchine di Turing (che riconoscono linguaggi di tipo ‘0’ o ricorsivamente enumerabili). Ci limitiamo ad osservare che  $\mathcal{L}_1 \subseteq \mathcal{L}_0$  discende direttamente dalle definizioni di linguaggio di tipo ‘1’ e di grammatica di tipo ‘0’. *c.v.d.*

## 5.4 Operazioni sui linguaggi.

### Definizione 5.2 (Operazioni sui linguaggi)

Siano  $L_1$  ed  $L_2$  due linguaggi definiti su uno stesso alfabeto  $X$  ( $L_1, L_2 \subseteq X^*$ ).

(i) L’*unione* di  $L_1$  ed  $L_2$  è:

$$L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\};$$

- (ii) La *concatenazione* di  $L_1$  ed  $L_2$  (anche detta il *prodotto* di  $L_1$  ed  $L_2$  o “ $L_1$  punto  $L_2$ ”) è:

$$L_1 \cdot L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\};$$

- (iii) L'*iterazione* di  $L_1$  (o *chiusura riflessiva e transitiva* di  $L_1$  rispetto all'operazione di concatenazione, anche detta *stellatura* di  $L_1$  o “ $L_1$  star” o *chiusura di Kleene*) è:

$$L_1^* = \{w \mid w = w_1 w_2 \dots w_n, n \geq 0 \text{ e } \forall i : w_i \in L_1\};$$

- (iv) Il *complemento* di  $L_1$  è:

$$\overline{L_1} = X^* - L_1$$

- (v) L'*intersezione* di  $L_1$  ed  $L_2$  è:

$$L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$$

■

### Proprietà 5.1



Dati  $L_1, L_2, L_3 \subseteq X^*$  ( $\equiv L_1, L_2, L_3 \in 2^{X^*}$ ), risulta:

- 1)  $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$  (proprietà associativa);
- 2)  $L_1 \cdot L_2 \neq L_2 \cdot L_1$ ;
- 3)  $L_1 \cdot \{\lambda\} = \{\lambda\} \cdot L_1 = L_1$ , ( $\{\lambda\}$  è l'elemento neutro rispetto all'operazione di concatenazione di linguaggi).

Dunque anche  $(2^{X^*}, \cdot)$  è un monoide;

- 4)  $L_1 \cdot \emptyset = \emptyset \cdot L_1 = \emptyset$  ( $\emptyset$  è l'elemento assorbente);
- 5) Se  $\lambda \in L_1$ :  $L_2 \subseteq L_1 \cdot L_2$   
 $L_2 \subseteq L_2 \cdot L_1$

6) Se  $\lambda \in L_2$ :  $L_1 \subseteq L_1 \cdot L_2$

$$L_1 \subseteq L_2 \cdot L_1$$

### Esempio 5.1

Si considerino i linguaggi:

$$L_1 = \{a^{2n} \mid n \geq 0\} \quad L_2 = \{b, cc\}$$

$$L_1 \cdot L_2 = \{b, cc, a^2b, a^2cc, a^4b, a^4cc, \dots\}$$

$$L_2 \cdot L_1 = \{b, cc, ba^2, cca^2, ba^4, cca^4, \dots\}$$

Dunque:

$$L_2 \subset L_1 \cdot L_2 \quad \text{e} \quad L_2 \subset L_2 \cdot L_1,$$

mentre:

$$L_1 \not\subseteq L_1 \cdot L_2 \quad \text{e} \quad L_1 \not\subseteq L_2 \cdot L_1$$

$$L_1 \cup L_2 = L_2 \cup L_1 = \{\lambda, b, cc, a^2, a^4, a^6, \dots\}$$

$$L_2^* = \{\lambda, b, cc, bb, bcc, ccb, bbb, cccc, \dots\}$$

$$L_1^* = \{\lambda, a^2, a^4, a^6, \dots\} = L_1$$

### Definizione 5.3 (Potenza di un linguaggio)

Sia  $L$  un linguaggio definito su un alfabeto  $X$ . Dicesi *potenza n-esima* di  $L$ , e si denota con  $L^n$ ,  $n \geq 0$ , il seguente linguaggio:

$$L^n = \begin{cases} \{\lambda\} & \text{se } n = 0 \\ L^{n-1} \cdot L & \text{altrimenti} \end{cases}$$

Posto:

$$L^+ = \bigcup_{i \geq 1} L^i$$

si ha:

$$L^* = \{\lambda\} \cup L^+ = L^0 \cup L^+ = \bigcup_{i \geq 0} L^i$$

$L^+$  è detta chiusura transitiva rispetto alla operazione di concatenazione. ■

Dunque, si ha:

$$L^0 = \{\lambda\}, \quad L^1 = L^0 \cdot L = L, \quad L^2 = L^1 \cdot L = (L^0 \cdot L) \cdot L$$

$$L^3 = L^2 \cdot L = (L^1 \cdot L) \cdot L = ((L^0 \cdot L) \cdot L) \cdot L$$

### Proposizione 5.3

Sia  $L$  un linguaggio definito su un alfabeto  $X$ . Si ha:

$$L^* = \bigcup_{i \geq 0} L^i = \{\lambda\} \cup L \cup L^2 \cup L^3 \cup \dots$$

### Esempio 5.2

Si consideri nuovamente il linguaggio  $L_2 = \{b, cc\}$ . Si ha:

$$L_2^0 = \{\lambda\}, \quad L_2^1 = L_2, \quad L_2^2 = \{bb, bcc, ccb, cccc\}$$

$$L_2^* = L_2^0 \cup L_2^1 \cup L_2^2 \cup \dots = \{\lambda, b, cc, bb, bcc, ccb, cccc, bbb, \dots\}$$

## 5.5 Proprietà di chiusura delle classi di linguaggi.

### Definizione 5.4 (Linguaggio - Classe di linguaggi)

1)  $L$  linguaggio definito su  $X \stackrel{\text{def}}{\Rightarrow} L \subseteq X^* \Leftrightarrow L \in 2^{X^*}$

2)  $\mathcal{L}$  classe di linguaggi su  $X \stackrel{\text{def}}{\Rightarrow} \mathcal{L} \subseteq 2^X \Leftrightarrow \mathcal{L} \in 2^{2^X}$  ■

### **Definizione 5.5** (Chiusura)

Sia  $\mathcal{L}$  una classe di linguaggi su  $X$ .

Sia  $\alpha$  un'operazione binaria sui linguaggi di  $\mathcal{L}$ :

$$\alpha : 2^{X^*} \times 2^{X^*} \rightarrow 2^{X^*}, \quad (L_1, L_2) \mapsto \alpha(L_1, L_2)$$

Sia  $\beta$  un'operazione unaria sui linguaggi di  $\mathcal{L}$ :

$$\beta : 2^{X^*} \rightarrow 2^{X^*}, \quad L \mapsto \beta(L)$$

$\mathcal{L}$  è *chiusa* rispetto ad  $\alpha \stackrel{\text{def}}{\Leftrightarrow} \forall L_1, L_2 \in \mathcal{L} : \alpha(L_1, L_2) \in \mathcal{L}$ .

$\mathcal{L}$  è *chiusa* rispetto a  $\beta \stackrel{\text{def}}{\Leftrightarrow} \forall L_1 \in \mathcal{L} : \beta(L_1) \in \mathcal{L}$  ■

### **Esempio 5.3**

$\mathcal{L}$  è chiusa rispetto all'iterazione se  $\forall L_1 \in \mathcal{L} : L_1^* \in \mathcal{L}$ .

### **Teorema 5.3**

La classe dei linguaggi di tipo  $i$ ,  $i=0,1,2,3$ , è chiusa rispetto alle operazioni di *unione*, *concatenazione* ed *iterazione*.

### **Dimostrazione 5.3**

La dimostrazione di questo teorema è *costruttiva*.

Siano  $L_1$  ed  $L_2$  due linguaggi:

$$L_1 = L(G_1) \quad G_1 = (X, V_1, S_1, P_1)$$

$$L_2 = L(G_2) \quad G_2 = (X, V_2, S_2, P_2)$$

Assumiamo che:<sup>1</sup>

$$V_1 \cap V_2 = \emptyset \quad S \notin V_1 \cup V_2$$

Poniamo:

$$V = V_1 \cup V_2 \cup \{S\}$$

Lo schema generale della dimostrazione è il seguente:

- consideriamo un'operazione alla volta (denotata con  $\alpha$ );
- date  $G_1$  e  $G_2$ , costruiamo  $G$ ;
- si dimostra che, se  $G_1$  e  $G_2$  sono di tipo  $i$ , allora  $G$  è di tipo  $i$ ;
- si dimostra che  $L(G) = \alpha(L_1, L_2)$  e dunque la classe di linguaggi  $\mathcal{L}_i$  è chiusa rispetto alla operazione  $\alpha$ .

### **UNIONE**

Costruiamo la grammatica:

$$G_3 = (X, V, S, P_3)$$

ove:

$$P_3 = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2$$

Osserviamo che, se  $G_1$  e  $G_2$  sono entrambe di tipo  $i$ ,  $i=0,1,2$ , lo è anche  $G_3$ .

In ciascuno di questi casi, si ha:

$$L(G_3) = L_1 \cup L_2$$

Infatti una derivazione da  $S$  in  $G_3$  deve necessariamente iniziare o con:

$$S \Rightarrow S_1 \quad \text{ed in tal caso può generare unicamente parole di } L(G_1)$$

o con

$$S \Rightarrow S_2 \quad \text{ed in tal caso genera una parola di } L(G_2).$$

---

<sup>1</sup> Nel caso in cui tale assunzione non sia vera, ridenominiamo i nonterminali in comune.

Dunque, risulta dimostrato che  $\underline{G_0}$ ,  $\underline{G_1}$  e  $\underline{G_2}$ , sono chiuse rispetto all'unione.

Però, se  $G_1$  e  $G_2$  sono di tipo '3',  $G_3$  non è lineare destra, perché le produzioni  $S \rightarrow S_1$  ed  $S \rightarrow S_2$  non sono ammesse.

Per avere ancora produzioni lineari destre che simulino il passo iniziale di una derivazione in  $G_1$  ed anche il passo iniziale di una derivazione in  $G_2$ , costruiamo pertanto la grammatica:

$$G_4 = (X, V, S, P_4)$$

ove:

$$P_4 = \left\{ S \rightarrow w \mid S_1 \rightarrow w \in P_1 \right\} \cup \left\{ S \rightarrow w \mid S_2 \rightarrow w \in P_2 \right\} \cup P_1 \cup P_2$$

$G_4$  è lineare destra se  $G_1$  e  $G_2$  lo sono e inoltre:

$$L(G_4) = L_1 \cup L_2$$

ed  $\underline{G_3}$  è chiusa rispetto all'unione.

#### Esempio 5.4

$$G_1 \text{ con } P_1 = \{S_1 \rightarrow aA, A \rightarrow b\}$$

$$G_2 \text{ con } P_2 = \{S_2 \rightarrow cC, C \rightarrow b\}$$

$$V_1 \cap V_2 = \emptyset \quad \text{mentre} \quad X_1 = \{a, b\} \text{ ed } X_2 = \{b, c\}$$

Consideriamo

$$X = \{a, b, c\} = X_1 \cup X_2$$

e

$$G_4 \text{ con } P_4 = \{S \rightarrow aA, S \rightarrow cC, S_1 \rightarrow aA, A \rightarrow b, S_2 \rightarrow cC, C \rightarrow b\}$$

$$L(G_1) = \{ab\} \quad L(G_2) = \{cb\} \quad L(G_4) = \{ab, cb\}$$

dunque

$$L(G_4) = L(G_1) \cup L(G_2)$$

e

$$G_4 = (X = \{a, b, c\}, V = V_1 \cup V_2 \cup \{S\} = \{S, S_1, S_2, A, C\}, S, P_4)$$

Osserviamo che  $S_1, S_2$  e le produzioni  $S_1 \rightarrow aA, S_2 \rightarrow cC$  non sono di alcuna utilità, in quanto non possono essere utilizzate in alcuna derivazione da  $S$  in  $G_4$ .  $S_1$  ed  $S_2$  sono detti *nonterminali inutili* (Definizione 8.12) perché non compariranno mai all'interno di una forma di frase generata da  $S$  in  $G_4$ .

La grammatica  $G_4$ , per effetto della eliminazione dei nonterminali inutili, diventa:

$$G'_4 = (X' = \{a, b, c\}, V' = \{S, A, C\}, S, P'_4 = \{S \rightarrow aA | cC, A \rightarrow b, C \rightarrow b\})$$

e  $L(G'_4) = L(G_4)$  (dimostrazione per esercizio).

### CONCATENAZIONE

Costruiamo la grammatica:

$$G_5 = (X, V, S, P_5)$$

ove:

$$P_5 = \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2$$

Osserviamo che, se  $G_1$  e  $G_2$  sono entrambe di tipo  $i$ ,  $i = 0, 1, 2$ , lo è anche  $G_5$ .

Se  $G_1$  e  $G_2$  sono C.F. (tipo '2'), allora si ha:

$$L(G_5) = L_1 \cdot L_2$$

in quanto ogni derivazione da  $S$  in  $G_5$  ha la seguente "struttura":

$$S \Rightarrow S_1 S_2 \xrightarrow[G_1]{*} w_1 S_2 \xrightarrow[G_2]{*} w_1 w_2$$

ove, evidentemente, si ha:

$$S_1 \xrightarrow{*} w_1 \quad S_2 \xrightarrow{*} w_2$$

Il seguente controesempio mostra che la condizione che  $G_1$  e  $G_2$  siano di tipo ‘2’ è fondamentale per la validità di  $L(G_5) = L_1 \cdot L_2$ .

### Esempio 5.5 (Controesempio)

Consideriamo  $G_1$  con  $P_1 = \{S_1 \rightarrow b\}$  e  $G_2$  con  $P_2 = \{bS_2 \rightarrow bb\}$

Da cui  $L_1 = L(G_1) = \{b\}$  ed  $L_2 = L(G_2) = \emptyset$ .

Dunque:

$$L_1 \cdot L_2 = \emptyset.$$

Se costruiamo  $G_5$ , si ha:

$$P_5 = \{S \rightarrow S_1S_2, S_1 \rightarrow b, bS_2 \rightarrow bb\}$$

ed  $L(G_5) \neq \emptyset$  in quanto  $S \xrightarrow[G_5]{*} bb$  attraverso la seguente derivazione:

$$S \Rightarrow S_1S_2 \Rightarrow bS_2 \Rightarrow bb.$$

Dunque la  $G_5$  va bene solo per grammatiche di tipo ‘2’ e risulta dimostrato che  $\mathcal{L}_2$  è chiusa rispetto alla concatenazione. La dimostrazione fatta non va bene per grammatiche di tipo ‘0’ e di tipo ‘1’, in quanto entrambe queste classi di grammatiche presentano produzioni dipendenti da contesto.

In presenza di grammatiche di tali tipi è necessario impedire che le derivazioni da  $S_2$  si servano di precedenti derivazioni da  $S_1$  e/o viceversa. In altri termini, è necessario evitare interferenze tra derivazioni da  $S_1$  e derivazioni da  $S_2$  nella definizione dei contesti. È possibile ottenere ciò considerando copie distinte di  $X$  in derivazioni distinte (da  $S_1$  e da  $S_2$ ), in modo che nessun  $NT$  derivato da  $S_1$  possa far uso di parte di una forma di frase derivata da  $S_2$  come contesto per l’applicazione di una produzione (e viceversa).

Siano pertanto:

$$X' = \{x' | x \in X\} \quad \text{e} \quad X'' = \{x'' | x \in X\}$$

due copie distinte di  $X$  tali che:

$$X' \cap X'' = \emptyset \quad X' \cap X = \emptyset \quad X' \cap V = \emptyset$$

$$X'' \cap X = \emptyset \quad X'' \cap V = \emptyset$$

e sia  $P'_1$  l'insieme delle produzioni ottenute da  $P_1$  sostituendo ogni occorrenza di un terminale  $x$  in  $X$  con il corrispondente nonterminale  $x'$  in  $X'$ :

$$P'_1 = P_1 \left[ \begin{matrix} x' \\ / \\ x \end{matrix} \right]$$

Similmente, sia  $P''_2$  l'insieme delle produzioni ottenute da  $P_2$  sostituendo ogni occorrenza di un terminale  $x$  in  $X$  con il corrispondente  $x''$  in  $X''$ :

$$P''_2 = P_2 \left[ \begin{matrix} x'' \\ / \\ x \end{matrix} \right]$$

In questo modo evitiamo l'interferenza tra contesti.

Costruiamo ora la grammatica:

$$G_6 = (X, V \cup X' \cup X'', S, P_6)$$

ove:

$$P_6 = \{S \rightarrow S_1 S_2\} \cup P'_1 \cup P''_2 \cup \{x' \rightarrow x \mid x \in X\} \cup \{x'' \rightarrow x \mid x \in X\}$$

Se  $G_1$  e  $G_2$  sono entrambe di tipo  $i$ ,  $i = 0, 1$ , lo è anche  $G_6$ . Inoltre, si ha:

$$L(G_6) = L_1 \cdot L_2$$

Il controesempio visto in precedenza non dà più problemi:

$$P'_1 = \{S_1 \rightarrow b'\} \quad P''_2 = \{b'' S_2 \rightarrow b'' b''\}$$

e

$$G_6 = (X, V \cup \{b'\} \cup \{b''\}, S, P_6)$$

dove

$$P_6 = \{S \rightarrow S_1 S_2, S_1 \rightarrow b', b'' S_2 \rightarrow b'' b'', b' \rightarrow b, b'' \rightarrow b\}$$

ed  $L(G_6) = \emptyset$  in quanto  $S \xrightarrow[G_6]{*} bb$ ,  $S \Rightarrow S_1 S_2 \Rightarrow b' S_2 \Rightarrow b S_2$ .

Risulta così dimostrato che  $\mathcal{L}_0$  ed  $\mathcal{L}_1$  sono chiuse rispetto alla concatenazione.

È immediato osservare che, se  $G_1$  e  $G_2$  sono di tipo ‘3’ né  $G_5$  né  $G_6$  sono di tipo ‘3’, per la presenza della produzione  $S \rightarrow S_1 S_2$ .

Dobbiamo simulare l’effetto della produzione  $S \rightarrow S_1 S_2$ , che determina la concatenazione delle parole generate da  $S_1$  e da  $S_2$ . A tale scopo osserviamo che, data una grammatica di tipo ‘3’, ogni forma di frase derivata dal simbolo iniziale di tale grammatica ha due peculiarità:

- 1) in essa compare al più un  $NT$ ;
- 2) se in essa compare un  $NT$ , questo è il simbolo più a destra  
 $(S \Rightarrow x_1 A \Rightarrow x_1 x_2 B \xrightarrow{*} x_1 x_2 x_3 \dots x_n N \Rightarrow x_1 x_2 x_3 \dots x_n x_{n+1}).$

Modifichiamo pertanto ogni produzione del tipo  $A \rightarrow b$  in  $G_1$  in modo che essa non costituisca l’ultima produzione applicata in una derivazione da  $S_1$  in  $G_1$ , ma possa innescare una derivazione da  $S_2$  in  $G_2$ . Poniamo dunque a destra della  $b$  il simbolo iniziale  $S_2$  di  $G_2$ .

Le produzioni del tipo  $A \rightarrow b$  in  $G_1$  vengono trasformate in:  $A \rightarrow b S_2$ .

Costruiamo dunque la grammatica:

$$G_7 = (X, V - \{S\}, S_1, P_7)$$

ove:

$$\begin{aligned} P_7 = & \{A \rightarrow bB \mid A \rightarrow bB \in P_1\} \cup \{A \rightarrow bS_2 \mid A \rightarrow b \in P_1, b \neq \lambda\} \cup \\ & \cup \{A \rightarrow bS_2 \mid A \rightarrow bB \in P_1, B \rightarrow \lambda \in P_1\} \end{aligned} \quad (*)$$

$$\cup \left\{ S_1 \rightarrow w \mid S_2 \rightarrow w \in P_2, S_1 \rightarrow \lambda \in P_1 \right\} \cup \\ \cup P_2$$

Le (\*) e (\*\*) sono state introdotte per garantire la correttezza della grammatica generata, anche in presenza di  $\lambda$ -produzioni in  $G_1$ .

$G_7$  è di tipo '3' se  $G_1$  e  $G_2$  lo sono ed inoltre:

$$L(G_7) = L_1 \cdot L_2$$

ed  $G_7$  è chiusa rispetto alla concatenazione.

### ITERAZIONE

Costruiamo la grammatica:

$$G_8 = (X, V_1 \cup \{S\}, S, P_8)$$

ove:

$$P_8 = \{S \rightarrow \lambda, S \rightarrow S_1 S\} \cup P_1$$

Data la grammatica  $G_1 = (X, V_1, S_1, P_1)$  che genera  $L_1$ , la grammatica  $G_8$  genera la parola vuota  $\lambda$  e tutte le parole che si possono ottenere per concatenazione di parole generate da  $S_1$  in  $G_1$ .

Si ha infatti:

$$S \xrightarrow[G_8]{n} \underbrace{S_1 S_1 \dots S_1}_n S \Rightarrow S_1 S_1 \dots S_1 \xrightarrow{*} w_1 w_2 \dots w_n$$

con  $w_i \in L_1$ ,  $i = 1, 2, \dots, n$ .

Vediamo per quali classi di grammatiche la  $G_8$  è la grammatica che stiamo cercando:

- se  $G_1$  è di tipo '3',  $G_8$  non è di tipo '3', perché  $S \rightarrow S_1 S$  non è lineare destra;

- se  $G_1$  è di tipo ‘2’,  $G_8$  è di tipo ‘2’ e si ha:

$$L(G_8) = L_1^*$$

e risulta dimostrato che  $\mathcal{L}_2$  è chiusa rispetto all’iterazione.

- se  $G_1$  è di tipo ‘1’,  $G_8$  non è di tipo ‘1’, perché  $S$  compare nella parte destra della produzione  $S \rightarrow S_1S$  ed  $S \rightarrow \lambda$  è una produzione in  $P_8$ ; possiamo trasformare facilmente  $G_8$  nella grammatica equivalente  $G'_8$ , definita come segue:

$$G'_8 = (X, V_1 \cup \{S, S'\}, S, P'_8)$$

$$P'_8 = \{S \rightarrow \lambda | S', S' \rightarrow S_1 | S_1 S'\} \cup P_1$$

ma  $G'_8$  incorre nello stesso problema di interferenza nella definizione dei contesti visto nella dimostrazione per la concatenazione.

- se  $G_1$  è di tipo ‘0’, lo è anche  $G_8$ , ma si incorre nello stesso problema di interferenza nella definizione dei contesti visto nella dimostrazione per la concatenazione.

Il problema dell’interferenza dei contesti, comune agli ultimi due casi -  $G_1$  di tipo  $i$ ,  $i = 0, 1$  - può essere risolto come segue.

Eliminiamo dapprima le produzioni del tipo  $S_1 \rightarrow \lambda$  (si veda il Lemma della stringa vuota - Lemma 5.2).

Utilizziamo gli insiemi ausiliari di  $NT$   $X'$  e  $X''$  per evitare interferenze:

$$X' = \{x' | x \in X\} \quad X'' = \{x'' | x \in X\}$$

$$X' \cap X'' = \emptyset$$

$$X' \cap X = \emptyset$$

$$X' \cap V_1 = \emptyset$$

$$X'' \cap X = \emptyset$$

$$X'' \cap V_1 = \emptyset$$

Costruiamo due copie di  $G_1$ :

$$G'_1 = (X, V \cup X' \cup \{S_1\}, S_1, P'_1)$$

$$G_1'' = \left( X, V \cup X'' \cup \{S_2\}, S_2, P_1'' \right)$$

ove:

$$P_1' = P_1 \left[ \frac{x'}{x} \right] \quad P_1'' = P_1 \left[ \frac{x''}{x} \right]$$

Infine, combiniamo  $G_1'$ ,  $G_1''$  con produzioni che costruiscono sequenze finite di copie di  $S_1$  ed  $S_2$  che si alternano, in modo da ottenere  $L_1^*$ .

Dunque, la grammatica che otteniamo è:

$$G_9 = \left( X, V \cup X' \cup X'' \cup \{S_1, S_1', S_2, S_2'\} \cup \{S\}, S, P_9 \right)$$

ove:

$$\begin{aligned} P_9 = & \{S \rightarrow \lambda \mid S_1' \mid S_2', \quad S_1' \rightarrow S_1 \mid S_1 S_2', \quad S_2' \rightarrow S_2 \mid S_2 S_1'\} \cup P_1' \cup P_1'' \cup \\ & \cup \{x' \rightarrow x \mid x \in X\} \cup \{x'' \rightarrow x \mid x \in X\} \end{aligned}$$

Se  $G_1$  è di tipo  $i$ ,  $i = 0, 1$ , lo è anche  $G_9$  e si ha:

$$L(G_9) = L_1^*$$

e risulta dimostrato che  $\mathcal{L}_0$  e  $\mathcal{L}_1$  sono chiuse rispetto all'iterazione.

Resta da dimostrare che  $\mathcal{L}_3$  è chiusa rispetto all'iterazione.

Per costruire la nuova grammatica, introduciamo dapprima un nuovo simbolo iniziale  $S$  e la produzione  $S \rightarrow \lambda$  che genera la stringa vuota.

Inoltre, eliminiamo da  $P_1$ , se c'era, la produzione  $S_1 \rightarrow \lambda$  ed aggiungiamo una produzione  $S \rightarrow w$  per ogni produzione "iniziale"  $S_1 \rightarrow w$  in  $P_1$ .

Infine, per ogni produzione la cui parte destra contiene solo un terminale, del tipo  $A \rightarrow b$ , nell'insieme delle produzioni che stiamo costruendo, aggiungiamo la produzione  $A \rightarrow bS$  in modo che, avendo derivato una forma di frase del tipo  $w_1 w_2 \dots w_j A$  tale che ogni sottostringa  $w_1, w_2, \dots, w_j = w'_j b$  è una parola di  $L_1$ , abbiamo la possibilità di terminare la derivazione con

$w_1 w_2 \dots w_j$  o di continuarla generando la forma di frase  $w_1 w_2 \dots w_j S$ , che consente di generare una parola più lunga di  $L_1^*$ .

Si noti che, per garantire la correttezza della grammatica generata anche in presenza di  $\lambda$ -produzioni in  $G_1$ , occorre aggiungere una produzione  $A \rightarrow bS$  per ogni produzione  $A \rightarrow bB$  nell'insieme delle produzioni che stiamo costruendo, quando  $B \rightarrow \lambda$  è pure una produzione di tale insieme, alla stregua di quanto fatto per la concatenazione in  $\mathcal{L}_3$ .

Costruiamo dunque la grammatica:

$$G_{10} = (X, V_1 \cup \{S\}, S, P_{10})$$

ove:

$$\begin{aligned} P_{10} = & \{S \rightarrow \lambda\} \cup \left( P_1 - \{S_1 \rightarrow \lambda\} \right) \cup \left\{ S \rightarrow w \mid S_1 \rightarrow w \in P_1 \right\} \cup \\ & \cup \left\{ A \rightarrow bS \mid A \rightarrow b \in P_{10}, b \neq \lambda \right\} \cup \left\{ A \rightarrow bS \mid A \rightarrow bB \in P_{10}, b \rightarrow \lambda \in P_{10} \right\} \end{aligned}$$

Si noti che la definizione di  $P_{10}$  è ricorsiva. Se  $G_1$  è di tipo '3', lo è anche  $G_{10}$  e si ha:

$$L(G_{10}) = L_1^*$$

e risulta dimostrato che  $\mathcal{L}_3$  è chiusa rispetto all'iterazione.

c.v.d.

Per la loro importanza pratica, riassumiamo le modalità di costruzione delle grammatiche che generano i linguaggi *unione/concatenazione/iterazione* di  $L_1$  ed  $L_2$  nella Tavola 1 che segue.

	<b>UNIONE</b>	<b>CONCATENAZIONE</b>	<b>ITERAZIONE</b>
$\mathcal{Q}_0$	$G_3 = (X, V, S, P_3)$ $P_3 = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2$	$X = \{x'   x \in X\}, X'' = \{x''   x \in X\}, X' \cap X'' = \emptyset, X \cap V = \emptyset,$ $X'' \cap V = \emptyset, X'' \cap X = \emptyset, X''' \cap X = \emptyset,$ $X''' \cap V = \emptyset, X''' \cap X = \emptyset, X''' \cap X'' = \emptyset,$ $X'' \cap V = \emptyset, X'' \cap X = \emptyset, X''' \cap X = \emptyset, X''' \cap X'' = \emptyset,$ $X''' \cap V = \emptyset, X''' \cap X = \emptyset, X''' \cap X'' = \emptyset, X''' \cap X''' = \emptyset,$ $X''' \cap V = \emptyset, X''' \cap X = \emptyset, X''' \cap X'' = \emptyset, X''' \cap X''' = \emptyset, X''' \cap V = \emptyset,$ $X''' \cap V = \emptyset, X''' \cap X = \emptyset, X''' \cap X'' = \emptyset, X''' \cap X''' = \emptyset, X''' \cap V = \emptyset, X''' \cap X''' = \emptyset$	<p>Eliminiamo le produzioni del tipo:</p> $S_1 \rightarrow \lambda$ $X' = \{x'   x \in X\}, X'' = \{x''   x \in X\}, X' \cap X'' = \emptyset,$ $X'' \cap V = \emptyset, X'' \cap X = \emptyset, X''' \cap X = \emptyset, X''' \cap V = \emptyset,$ <p>Costruiamo due copie di <math>G_1</math>:</p> $G'_1 = \left( X, V \cup X' \cup \{S_1\}, S_1, P'_1 \right)$ $G''_1 = \left( X, V \cup X'' \cup \{S_2\}, S_2, P''_1 \right)$ $P'_1 = R_1 \left[ \frac{x'}{x} \right] \quad P''_1 = R_1 \left[ \frac{x''}{x} \right]$ $G_6 = \left( X, V \cup X' \cup X'', S, P_6 \right)$ $P_6 = \left\{ S \rightarrow S_1 S_2 \right\} \cup P' \cup P''$ $\cup \left\{ x' \rightarrow x   x \in X \right\} \cup \left\{ x'' \rightarrow x   x \in X \right\}$ $G_9 = \left( X, V \cup X'' \cup X''' \cup \{S_1, S'_1, S_2, S''_2, S_3\}, P_9 \right)$ $R_9 = \left\{ S \rightarrow \lambda S'_1 \left  \begin{matrix} S'_1 \\ S_2 \end{matrix} \right. \right\} \cup \left\{ S'_1 \rightarrow S_1 \left  \begin{matrix} S'_1 \\ S_2 \end{matrix} \right. \right\} \cup \left\{ S_2 \left  \begin{matrix} S'_1 \\ S_2 \end{matrix} \right. \right\}$ $\cup P'_1 \cup P''_1 \cup \left\{ x' \rightarrow x   x \in X \right\} \cup \left\{ x'' \rightarrow x   x \in X \right\}$
$\mathcal{Q}_1$	IDEIM		
$\mathcal{Q}_2$	IDEIM	$G_5 = \left( X, V, S, P_5 \right)$ $P_5 = \left\{ S \rightarrow S_1 S_2 \right\} \cup P_1 \cup P_2$	$G_8 = \left( X, V_1 \cup \{S\}, S, P_8 \right)$ $P_8 = \left\{ S \rightarrow \lambda   S_1 S \right\} \cup P_1$
$\mathcal{Q}_3$	$G_4 = \left( X, V, S, P_4 \right)$ $P_4 = \left\{ \begin{array}{l} S \rightarrow w   S_1 \rightarrow w \in P_1 \\ S \rightarrow w   S_2 \rightarrow w \in P_2 \end{array} \right\} \cup P_1 \cup P_2$	$G_7 = \left( X, V - \{S\}, S, P_7 \right)$ $P_7 = \left\{ \begin{array}{l} A \rightarrow bB   A \rightarrow bB \in P_1 \\ A \rightarrow bS_2   A \rightarrow bB \in P_1, b \neq \lambda \end{array} \right\} \cup \left\{ \begin{array}{l} A \rightarrow bS_2   A \rightarrow b \in P_1 \\ A \rightarrow bS_2   A \rightarrow b \in P_1, b \neq \lambda \end{array} \right\}$ $\left( \cup \left\{ \begin{array}{l} S_1 \rightarrow w   S_2 \rightarrow w \in P_2 \\ S_1 \rightarrow w   S_2 \rightarrow w \in P_1 \end{array} \right\} \text{ se } S_1 \rightarrow \lambda \in P_1 \right) \cup P_2$	$G_{10} = \left( X, V_1 \cup \{S\}, S, P_{10} \right)$ $P_{10} = \left\{ \begin{array}{l} S \rightarrow \lambda \right\} \cup \left( P_1 - \{S_1 \rightarrow \lambda\} \right) \cup \left\{ S \rightarrow w   S_1 \rightarrow w \in P_1 \right\}$ $\cup \left\{ \begin{array}{l} A \rightarrow bS_1   A \rightarrow bB \in P_1, b \in P_1 \\ A \rightarrow bS_1   A \rightarrow bB \in P_1, b \neq \lambda \end{array} \right\}$ $\cup \left\{ \begin{array}{l} A \rightarrow bS_1   A \rightarrow bB \in P_1, b \neq \lambda \\ A \rightarrow bS_1   A \rightarrow bB \in P_1, b \in P_1 \end{array} \right\}$ $\cup \left\{ \begin{array}{l} A \rightarrow bS_1   A \rightarrow bB \in P_1, b \in P_1 \\ A \rightarrow bS_1   A \rightarrow bB \in P_1, b \neq \lambda \end{array} \right\}$

Tavola 1. Proprietà di Chiusura

### **Teorema 5.4**

- 1) La classe dei linguaggi lineari destri (tipo ‘3’) è chiusa rispetto al complemento ed all’intersezione.
- 2) La classe dei linguaggi non contestuali (tipo ‘2’) non è chiusa rispetto al complemento ed all’intersezione.
- 3) La classe dei linguaggi contestuali (tipo ‘1’) è chiusa rispetto al complemento (e dunque anche rispetto all’intersezione).
- 4) La classe dei linguaggi di tipo ‘0’ non è chiusa rispetto al complemento.

### **Dimostrazione 5.4**

- 1) Dimostreremo prossimamente la chiusura di  $\mathcal{L}_3$  rispetto al complemento.

Dimostriamola rispetto all’intersezione:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \quad (\text{Leggi di De Morgan})$$

La chiusura di  $\mathcal{L}_3$  rispetto all’intersezione discende direttamente da questo risultato.

- 2) Consideriamo i linguaggi:

$$L_1 = \{a^n b^n c^m \mid n, m > 0\} \qquad L_2 = \{a^n b^m c^m \mid n, m > 0\}$$

$L_1$  e  $L_2$  sono linguaggi liberi (perché? dimostrarlo), mentre  $L_1 \cap L_2$  non lo è:

$$L_1 \cap L_2 = \{a^k b^k c^k \mid k > 0\}.$$

Il complemento è:  $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ .

Dunque, se  $L_1, L_2 \in \mathcal{L}_2$  e se  $\mathcal{L}_2$  fosse chiusa rispetto al complemento, ...

- 3) Un risultato recente ha stabilito che  $\mathcal{L}_1$  è chiusa rispetto al complemento (e quindi all’intersezione). Non se ne conosce la dimostrazione.
- 4) Non lo dimostriamo. *c.v.d.*

## 5.6 L'operazione di riflessione.

Definizione 5.6 (Stringa riflessa)

Sia  $w$  una parola su un alfabeto  $X = \{x_1, x_2, \dots, x_k\}$ ,  $w = x_{i_1} x_{i_2} \dots x_{i_{n-1}} x_{i_n}$ . Dicesi *stringa riflessa* (o *riflessione*) di  $w$  la stringa:

$$w^R = x_{i_n} x_{i_{n-1}} \dots x_{i_2} x_{i_1}$$

■

Definizione 5.7 (Operazione di riflessione)

Sia  $w$  una parola su un alfabeto  $X = \{x_1, x_2, \dots, x_k\}$  e sia  $w^R$  la stringa riflessa di  $w$ . L'operazione che trasforma  $w$  in  $w^R$  è detta *operazione di riflessione*.

■

Definizione 5.8 (Parola palindromica)

Un *palindromo* (o *parola palindromica*) è una parola la cui lettura a ritroso riproduce la parola di partenza:

$$w \text{ palindromo} \stackrel{\text{def}}{\Leftrightarrow} w = w^R$$

Un palindromo è dunque una parola che coincide con la sua riflessione. ■

■

Esempio 5.1

Alcuni palindromi sull'alfabeto  $\{a, b, \dots, z\}$  sono:

- $a$ ;
- $ii$  (plurale di io?);
- $non, ala, ara, ici$ ;
- $osso, alla, arra$ ;
- $radar, alalà, arerà$  (ignorando l'accento);
- $osseoso, ingegni$ ;

- *avallava, ovattavo;*
- *onorarono;*
- *accavallavacca, accumolomucca;*
- *fecì nulla all'Unicef, ogni tela male tingo* (ignorando spazi bianchi, punteggiatura e differenza tra maiuscole e minuscole).

I palindromi (su un qualunque alfabeto) sono di due tipi:

- ◆ palindromi di lunghezza pari;
- ◆ palindromi di lunghezza dispari.

I primi hanno un “*asse di simmetria*” costituito dalla parola vuota; i secondi hanno un “*asse di simmetria*” costituito da uno dei simboli dell’alfabeto.

Più precisamente, si ha la seguente caratterizzazione (senza dimostrazione):

### Teorema 5.5

Sia  $w$  una parola su un alfabeto  $X$ .

$w$  è palindromo se e solo se  $w = \alpha x \alpha^R$ ,  $x \in X \cup \{\lambda\}$ .

### Teorema 5.6

La classe dei linguaggi non contestuali (tipo ‘2’) è chiusa rispetto all’operazione di riflessione.

### Dimostrazione 5.6

Sia  $G_1 = (X, V_1, S_1, P_1)$  una grammatica non contestuale. Dobbiamo dimostrare che:

$L(G_1)$  non contestuale  $\Rightarrow (L(G_1))^R = \{w^R \mid w \in L(G_1)\}$  è non contestuale.

Costruiamo la grammatica:

$$G_{11} = (X, V_1, S_1, P_{11})$$

ove:

$$P_{11} = \{ A \rightarrow w^R \mid A \rightarrow w \in P_1 \}$$

Risulta allora:

$$L(G_{11}) = (L(G_1))^R$$

Dimostrare per esercizio.

Quindi se in  $P_1$  abbiamo la produzione:

$$A \rightarrow BaC$$

in  $P_{11}$  avremo la produzione:

$$A \rightarrow CaB$$

c.v.d.

## 5.7 Esercizi.

### Esercizio 5.1

Dimostrare che il linguaggio:

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

è non contestuale.

Consideriamo i linguaggi:

$$L_1 = \{a^n b^n \mid n > 0\} \quad L_2 = \{c^m \mid m > 0\} = \{c\}^+ = \{c\}^* - \{\lambda\}$$

Si ha:

$$L = L_1 \cdot L_2$$

$L_1$  è un linguaggio non contestuale ed  $L_2$  è un linguaggio lineare destro. Si ha infatti:

$$L_2 = \{c\}^* - \{\lambda\} = \{c\}^* \cap \overline{\{\lambda\}}$$

e  $\{c\}^*$  è lineare destro (per esercizio determinare la grammatica che genera  $\{c\}^*$ ).

Poiché  $\{\lambda\}$  è lineare destro, per la chiusura dei linguaggi di tipo ‘3’ rispetto al complemento si ha che:

$$\overline{\{\lambda\}}$$

è lineare destro. Dunque  $L_2$  è lineare destro e

$$L = L_1 \cdot L_2$$

è non contestuale, poiché si ottiene per concatenazione di due linguaggi non contestuali ( $L_2 \in \mathcal{L}_2 \subset \mathcal{L}_3$ ).

La grammatica  $G_1$  che genera  $L_1$  è data da:

$$G_1 = (X_1, V_1, S_1, P_1)$$

$$X_1 = \{a, b\} \quad V_1 = \{S_1\} \quad P_1 = \left\{ S_1 \xrightarrow{(1)} aS_1b, S_1 \xrightarrow{(2)} ab \right\}.$$

La grammatica  $G_2$  che genera  $L_2$  è data da:

$$G_2 = (X_2, V_2, S_2, P_2)$$

$$X_2 = \{c\} \quad V_2 = \{C\} \quad S_2 = C \quad P_2 = \left\{ C \xrightarrow{(1)} cC, C \xrightarrow{(2)} c \right\}.$$

La grammatica  $G$  che genera  $L = L_1 \cdot L_2$  è data da:

$$G = (X, V, S, P)$$

$$X = X_1 \cup X_2 = \{a, b, c\} \quad V = V_1 \cup V_2 \cup \{S\} = \{S, S_1, C\}$$

$$P = \{S \rightarrow S_1 C\} \cup P_1 \cup P_2 = \left\{ S \xrightarrow{(1)} S_1 C, S_1 \xrightarrow{(2)} aS_1b, S_1 \xrightarrow{(3)} ab, C \xrightarrow{(4)} cC, C \xrightarrow{(5)} c \right\}.$$

$G$  è non contestuale.

### Esercizio 5.2

Siano  $L_1 = \{a^n b^n \mid n \geq 0\}$  ed  $L_2 = \{a\}^* \cdot \{bb\}^*$ . Dimostrare che il linguaggio  $L = L_1 \cap L_2$  è non contestuale.

Il linguaggio  $L_1$  può essere riguardato come l'unione di due insiemi:

$$L_1 = \{a^n b^n \mid n > 0\} \cup \{\lambda\}$$

e tale linguaggio è non contestuale.

La grammatica che lo genera è:

$$G_1 = (X_1, V_1, S_1, P_1)$$

$$X_1 = \{a, b\} \quad V_1 = \{S_1\} \quad P_1 = \left\{ S_1 \xrightarrow{(1)} aS_1b, S_1 \xrightarrow{(2)} ab, S_1 \xrightarrow{(3)} \lambda \right\}.$$

$L_2 = \{a\}^* \cdot \{bb\}^*$  è lineare destro. Infatti, il linguaggio  $L_3 = \{a\}^*$  è lineare destro e la grammatica che lo genera è:

$$G_3 = (X_3, V_3, S_3, P_3)$$

$$X_3 = \{a\} \quad V_3 = \{A\} \quad S_3 = A$$

$$P_3 = \left\{ A \xrightarrow{(1)} aA, A \xrightarrow{(2)} a, A \xrightarrow{(3)} \lambda \right\}.$$

Ma anche il linguaggio  $L_4 = \{bb\}^*$  è lineare destro e la grammatica che lo genera è:

$$G_4 = (X_4, V_4, S_4, P_4)$$

$$X_4 = \{b\} \quad V_4 = \{B, B_1\} \quad S_4 = B$$

$$P_4 = \left\{ B \xrightarrow{(1)} bB_1, B \xrightarrow{(2)} \lambda, B_1 \xrightarrow{(3)} bB, B_1 \xrightarrow{(4)} b \right\}$$

Dunque la grammatica che genera  $L_2 = L_3 \cdot L_4$  è:

$$G_2 = (X_2, V_2, S_2, P_2)$$

$$X_2 = X_3 \cup X_4 = \{a, b\} \quad V_2 = V_3 \cup V_4 = \{A, B, B_1\} \quad S_2 = S_3 = A$$

$$\begin{aligned} P_2 &= \{A \rightarrow aA\} \cup \{A \rightarrow aB\} \cup \{B \rightarrow bB_1, B \rightarrow \lambda, B_1 \rightarrow bB, B_1 \rightarrow b\} \cup \\ &\quad \cup \{A \rightarrow bB_1, A \rightarrow \lambda\} = \\ &= \left\{ \begin{array}{l} A \xrightarrow{(1)} aA, A \xrightarrow{(2)} aB, A \xrightarrow{(3)} bB_1, A \xrightarrow{(4)} \lambda, B \xrightarrow{(5)} bB_1, B \xrightarrow{(6)} \lambda, B_1 \xrightarrow{(7)} bB, B_1 \xrightarrow{(8)} b \end{array} \right\} \end{aligned}$$

$L = L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ , per la legge di De Morgan. Non possiamo però dire nulla sulla classe di linguaggi cui  $L$  appartiene, dato che  $L_1$  è non contestuale e la classe dei linguaggi non contestuali non è chiusa rispetto al complemento. Procediamo in modo diverso.

Il linguaggio  $L = L_1 \cap L_2$  non è altro che:

$$L = L_1 \cap L_2 = \left\{ w \mid w \in \{a, b\}^*, w = a^n b^n, n \geq 0 \text{ AND } w = a^k (bb)^m, k, m \geq 0 \right\}$$

Poiché necessariamente si deve avere  $k = n$ , si ha:

$$\begin{aligned} L &= \left\{ w \mid w \in \{a, b\}^*, w = a^n b^n, n \geq 0 \text{ AND } w = a^n (bb)^m, m \geq 0 \right\} = \\ &= \left\{ w \mid w \in \{a, b\}^*, w = a^n b^n, n \geq 0 \text{ AND } w = a^n b^{2m}, m \geq 0 \right\} \end{aligned}$$

ma, poiché si deve avere  $n = 2m$ , si ha:

$$L = \left\{ w \mid w \in \{a, b\}^*, w = a^{2m} b^{2m}, m \geq 0 \right\} = \{a^{2m} b^{2m} \mid m \geq 0\}$$

$L$  è un linguaggio non contestuale e la grammatica che genera  $L$  è:

$$G = (X, V, S, P)$$

ove:

$$X = \{a, b\} \quad V = \{S\} \quad P = \left\{ S \xrightarrow{(1)} aaSbb, S \xrightarrow{(2)} aabb, S \xrightarrow{(3)} \lambda \right\}.$$

### Esercizio 5.3

Si utilizzi la proprietà di chiusura della classe  $\mathcal{L}_2$  rispetto all'unione per dimostrare che ciascuno dei seguenti linguaggi è non contestuale:

(a)  $L = \{a^i b^j \mid i \neq j, i, j \geq 0\}$

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

(c)  $L = \{a, b\}^* - \{a^i b^i \mid i \geq 0\}$

(a) Analizziamo le parole del linguaggio:

$$\begin{aligned} L &= \{a^i b^j \mid i \neq j, i, j \geq 0\} = \\ &= \left\{ b, b^2, \dots, b^n, \dots, a, ab^2, \dots, ab^n, \right. \\ &\quad \left. a^2, a^2b, a^2b^3, \dots, a^2b^n, \dots, a^3b, a^3b^2, a^3b^4, \dots, a^3b^n, \dots \right\} = \\ &= \{a^n b^m \mid 0 \leq n < m\} \cup \{a^m b^n \mid 0 \leq n < m\} \end{aligned}$$

Poniamo:

$$L_1 = \{a^n b^m \mid 0 \leq n < m\}$$

$$L_2 = \{a^m b^n \mid 0 \leq n < m\}$$

Si ha:

$$L = L_1 \cup L_2.$$

$L_1$  è generato da:

$$G_1 = (X_1, V_1, S_1, P_1)$$

ove:

$$X_1 = \{a, b\} \quad V_1 = \{S_1, B\} \quad P_1 = \left\{ S_1 \xrightarrow{(1)} aS_1b, S_1 \xrightarrow{(2)} Bb, B \xrightarrow{(3)} Bb, B \xrightarrow{(4)} \lambda \right\}.$$

Analogamente,  $L_2$  è generato da:

$$G_2 = (X_2, V_2, S_2, P_2)$$

ove:

$$X_2 = \{a, b\} \quad V_2 = \{S_2, A\}$$

$$P_2 = \left\{ S_2 \xrightarrow{(1)} aS_2b, S_2 \xrightarrow{(2)} aA, A \xrightarrow{(3)} aA, A \xrightarrow{(4)} \lambda \right\}.$$

$L_1$  ed  $L_2$  sono non contestuali perché generati da grammatiche non contestuali.

Poiché la classe dei linguaggi non contestuali è chiusa rispetto all'unione, anche:

$$L = L_1 \cup L_2$$

è non contestuale. La grammatica che genera  $L$  è:

$$G = (X, V, S, P)$$

ove:

$$X = \{a, b\} \quad V = V_1 \cup V_2 \cup \{S\} = \{S, S_1, S_2, A, B\}$$

$$P = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2 =$$

$$= \{S \rightarrow S_1, S \rightarrow S_2, S_1 \rightarrow aS_1b \mid Bb, B \rightarrow Bb \mid \lambda,$$

$$S_2 \rightarrow aS_2b \mid aA, A \rightarrow aA \mid \lambda\}$$

Si provi a svolgere lo stesso esercizio *escludendo* la possibilità che  $i$  e  $j$  assumano valore zero.

$$(b) \quad L = \left\{ w \in \{a, b\}^* \mid w = w^R \right\}$$

Analizziamo le parole che costituiscono  $L$ :

$$L = \left\{ w \in \{a, b\}^* \mid w = w^R \right\} =$$

$$= \{\lambda, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, \dots\}$$

$L$  è l'insieme dei palindromi sull'alfabeto  $\{a, b\}$ .

Dunque, per il teorema di caratterizzazione, si ha che:

$$L = \left\{ \alpha x \alpha^R \mid \alpha \in \{a,b\}^*, x \in \{a,b,\lambda\} \right\} = \\ = \left\{ \alpha \alpha^R \mid \alpha \in \{a,b\}^* \right\} \cup \left\{ \alpha a \alpha^R \mid \alpha \in \{a,b\}^* \right\} \cup \left\{ \alpha b \alpha^R \mid \alpha \in \{a,b\}^* \right\}$$

Poniamo:

$$L_1 = \left\{ \alpha \alpha^R \mid \alpha \in \{a,b\}^* \right\}, \quad L_2 = \left\{ \alpha a \alpha^R \mid \alpha \in \{a,b\}^* \right\} \\ L_3 = \left\{ \alpha b \alpha^R \mid \alpha \in \{a,b\}^* \right\}$$

La grammatica che genera  $L_1$  è:

$$G_1 = (X, V_1, S_1, P_1)$$

ove:

$$X = \{a,b\} \quad V_1 = \{S_1\} \quad P_1 = \{S_1 \rightarrow aS_1a \mid bS_1b \mid \lambda\}.$$

La grammatica che genera  $L_2$  è:

$$G_2 = (X, V_2, S_2, P_2)$$

ove:

$$X = \{a,b\} \quad V_2 = \{S_2\} \quad P_2 = \{S_2 \rightarrow aS_2a \mid bS_2b \mid a\}.$$

La grammatica che genera  $L_3$  è:

$$G_3 = (X, V_3, S_3, P_3)$$

ove:

$$X = \{a,b\} \quad V_3 = \{S_3\} \quad P_3 = \{S_3 \rightarrow aS_3a \mid bS_3b \mid b\}.$$

$L_1$ ,  $L_2$  ed  $L_3$  sono linguaggi non contestuali perché  $G_1$ ,  $G_2$  e  $G_3$  sono grammatiche non contestuali.  $L = L_1 \cup L_2 \cup L_3$  è un linguaggio non contestuale dato che la classe dei linguaggi di tipo ‘2’ è chiusa rispetto all’unione.

La grammatica che genera  $L$  è:

$$G = (X, V, S, P)$$

ove:

$$\begin{aligned} X &= \{a, b\} \quad V = V_1 \cup V_2 \cup V_3 \cup \{S\} = \{S, S_1, S_2, S_3\} \\ P &= \{S \rightarrow S_1, S \rightarrow S_2, S \rightarrow S_3\} \cup P_1 \cup P_2 \cup P_3 = \\ &= \{S \rightarrow S_1 | S_2 | S_3, S_1 \rightarrow aS_1a | bS_1b | \lambda, \\ &\quad S_2 \rightarrow aS_2a | bS_2b | a, S_3 \rightarrow aS_3a | bS_3b | b\} \end{aligned}$$

(c) Risolvere per esercizio.

#### Esercizio 5.4

Dimostrare che il linguaggio:

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

non è lineare destro, utilizzando le proprietà di chiusura di questa classe di linguaggi.

Se  $L$  fosse lineare destro, allora, per le proprietà di chiusura dei linguaggi lineari destri, anche il linguaggio:

$$L_1 = \{a^n b^n \mid n > 0\}$$

sarebbe un linguaggio lineare destro. Infatti, si ha:

$$L_1 = L_2 - L$$

ove  $L_2 = \{a^n b^m \mid n, m > 0\}$ .  $L_2$  è un linguaggio lineare destro. Inoltre abbiamo che:

$$L_1 = L_2 - L = L_2 \cap \overline{L}$$

Se  $L$  fosse lineare destro,  $\overline{L}$  sarebbe lineare destro (poiché  $\mathcal{L}_3$  è chiusa rispetto al complemento).

Di conseguenza, anche  $L_1$  sarebbe lineare destro in quanto  $L_2$  è lineare destro ed  $\mathcal{L}_3$  è chiusa anche rispetto all'operazione di intersezione. Ma  $L_1$  è un linguaggio libero da contesto e non è lineare destro (si veda la dimostrazione del Teorema 5.1). Dunque  $L$  non è lineare destro.

Riassumiamo lo schema di ragionamento seguito nel precedente esercizio in Tavola 2.

<b>SCHEMA DI RAGIONAMENTO PER UTILIZZO PROPRIETÀ DI CHIUSURA</b>	
<i>ESATTO</i>	<i>ERRATO</i>
<p><u>Siano <math>L, L_1</math> ed <math>L_2</math> tre linguaggi tali che: <math>L = \alpha(L_1, L_2)</math> ove <math>\alpha = \cup, \cdot</math>.</u></p> <p><u>Supponiamo che <math>L_2 \in \mathcal{L}_i</math>.</u></p> <p><u>Se <math>L \notin \mathcal{L}_i</math> allora <math>L_1 \notin \mathcal{L}_i</math>.</u></p>	<p><u>Se <math>L_j \notin \mathcal{L}_i</math> allora <math>L \notin \mathcal{L}_i</math></u>  <math>j=1,2</math></p>

Tavola 2

## 6. Automi a stati finiti (deterministici e non deterministici).

### 6.1 Automi a stati finiti deterministic.

**Definizione 6.1** (Automa a stati finiti o accettore a stati finiti o FSA)

Sia  $X$  un alfabeto. Un *automa a stati finiti* (*FSA*) è una quadrupla:

$$M = (Q, \delta, q_0, F)$$

ove:

- $X$  è detto *alfabeto di ingresso*;
- $Q$  è un insieme finito e non vuoto di *stati*;
- $\delta$  è una funzione da  $Q \times X$  in  $Q$ , detta *funzione di transizione*:

$$\delta : Q \times X \rightarrow Q$$

- $q_0$  è lo *stato iniziale*;
- $F \subseteq Q$  è l'insieme degli *stati di accettazione o finali*. ■

Talora i valori della funzione di transizione  $\delta$  non sono definiti per tutte le coppie (stato-simbolo di ingresso)  $(q, x)$ . In tal caso, si dice che  $\delta$  è una funzione parziale o definita parzialmente. Questo significa che la lettura di  $x$  dà luogo in  $q$  ad un comportamento dell'automa che non si ritiene utile descrivere ai fini del riconoscimento (nel senso che produrrebbe stringhe non accettate).

Evidentemente questo fatto può essere descritto in modo equivalente, seguendo la definizione data di automa a stati finiti, passando da  $q$ , per effetto

di  $x$ , in uno stato  $q'$  dal quale non si possa mai raggiungere uno stato finale (*stato pozza*).

Un FSA può essere rappresentato mediante:

i) *Grafo degli Stati* o *Diagramma di Transizione* o *Diagramma di Stato*

È una rappresentazione grafica in cui:

- ogni stato  $q \in Q$  è rappresentato da un cerchio (*nodo*) con etichetta  $q$ ;
- lo stato iniziale (nodo  $q_0$ ) ha un arco orientato entrante libero (ossia, che non proviene da nessun altro nodo);
- per ogni stato  $q \in Q$  e per ogni simbolo  $x$  dell'alfabeto di ingresso,  $x \in X$ , se  $\delta(q, x) = q'$ , esiste un arco orientato etichettato con  $x$  uscente dal nodo  $q$  ed entrante nel nodo  $q'$  (Figura 6.1).

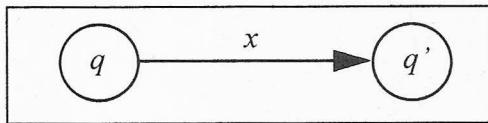


Figura 6.1

ii) *Tavola di Transizione*

È una tabella in cui sono riportati gli stati sulle righe e i simboli dell'alfabeto di ingresso sulle colonne.

Per ogni coppia (stato-ingresso) si legge nella tavola lo stato successivo:

$\delta$	$x_1$	$x_2$	...	....	$x_n$
$q_0$	$q_0^1$	$q_0^2$	...	...	$q_0^n$
$q_1$	$q_1^1$	$q_1^2$	...	...	$q_1^n$
...	...	...	...	...	...
...	...	...	...	...	...
$q_m$	$q_m^1$	$q_m^2$	...	...	$q_m^n$

dove l'alfabeto di ingresso e l'insieme degli stati sono rispettivamente:

$X = \{x_1, x_2, \dots, x_n\}$  e  $Q = \{q_0, q_1, \dots, q_m\}$ . Quindi si ha che:

$$\delta(q_i, x_j) = q_i^j \text{ con } q_i, q_i^j \in Q, x_j \in X.$$

Si può definire un'estensione della funzione di transizione  $\delta$  come segue:

### Definizione 6.2 ( $\delta^*$ per FSA)

Dato un FSA  $M = (Q, \delta, q_0, F)$  con alfabeto di ingresso  $X$ , definiamo per induzione la funzione:

$$\delta^* : Q \times X^* \rightarrow Q$$

tale che  $\delta^*(q, w)$ , per  $q \in Q$  e  $w \in X^*$ , sia lo stato in cui  $M$  si porta avendo in ingresso la parola  $w$  su  $X$  e partendo dallo stato  $q$ .

$$\begin{cases} \delta^*(q, \lambda) = q \\ \delta^*(q, wx) = \delta(\delta^*(q, w), x) \end{cases} \text{ per ogni } q \in Q, x \in X, w \in X^*$$

■

Si veda la Figura 6.2 per una descrizione grafica della definizione precedente.

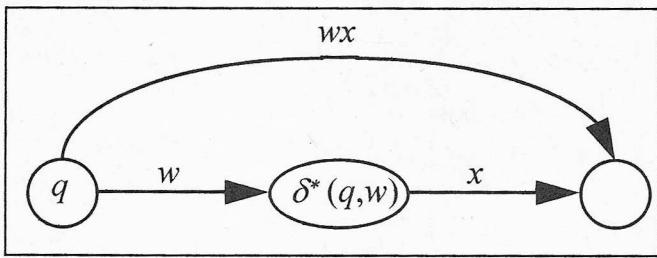


Figura 6.2

**Definizione 6.3** (Parola accettata o riconosciuta da un FSA)

Sia  $M = (Q, \delta, q_0, F)$  un FSA con alfabeto di ingresso  $X$ .

Una parola  $w \in X^*$  è *accettata* (o *riconosciuta*) da  $M$  se, partendo dallo stato iniziale  $q_0$ , lo stato  $q$  in cui l'automa si porta alla fine della sequenza di ingresso  $w$  è uno stato finale.

$$w \text{ accettata} \stackrel{\text{def}}{\Leftrightarrow} \delta^*(q_0, w) \in F$$

■

**Definizione 6.4** (Linguaggio accettato o riconosciuto da un FSA)

Sia  $M = (Q, \delta, q_0, F)$  un FSA con alfabeto di ingresso  $X$ .

Il *linguaggio accettato* o *riconosciuto* da  $M$  è il seguente sottoinsieme di  $X^*$ :

$$T(M) = \left\{ w \in X^* \mid \delta^*(q_0, w) \in F \right\}$$

(l'insieme delle parole accettate da  $M$ ). ■

**Definizione 6.5** (FSA equivalenti)

Sia  $M_1 = (Q_1, \delta_1, q_1, F_1)$  ed  $M_2 = (Q_2, \delta_2, q_2, F_2)$  due FSA di alfabeto di ingresso  $X$ .  $M_1$  ed  $M_2$  si dicono *equivalenti* se:

$$T(M_1) = T(M_2)$$

■

## 6.2 Linguaggi a stati finiti.

**Definizione 6.6** (Linguaggio a stati finiti, classe dei linguaggi a stati finiti)

Dato un alfabeto  $X$ , un linguaggio  $L$  su  $X$  è un *linguaggio a stati finiti* (o FSL - Finite State Language) se esiste un FSA  $M$  con alfabeto di ingresso  $X$  tale che  $L = T(M)$ .

Risulta così definita la *Classe dei Linguaggi a Stati Finiti*:

$$\mathcal{L}_{FSL} = \left\{ L \in 2^{X^*} \mid \exists M, M \text{ è un FSA: } L = T(M) \right\}$$

■

### **Proposizione 6.1**

I linguaggi a stati finiti sono chiusi rispetto al complemento.

### **Dimostrazione 6.1**

Sia  $L \in \mathcal{L}_{FSL}$  un linguaggio a stati finiti sull'alfabeto  $X$ .

Dalla definizione di linguaggio a stati finiti,  $L = T(M)$ , ove  $M = (Q, \delta, q_0, F)$ .<sup>1</sup>

Consideriamo il complemento di  $L$ :  $\bar{L} = X^* - L$  e l'automa a stati finiti  $\bar{M} = (Q, \delta, q_0, Q - F)$ . Si ha:  $\bar{L} = T(\bar{M})$ .

(Per esercizio, dimostrare  $\bar{L} = T(\bar{M})$  per induzione sulla lunghezza di una parola  $w$ ). *c.v.d.*

---

<sup>1</sup>  $M$  è un automa stati finiti con funzione di transizione  $\delta$  **definita totalmente** sul proprio dominio.

### 6.3 Automi a stati finiti non deterministici.

Definiamo ora una nuova classe di macchine a stati finiti.

**Definizione 6.7** (Automa a stati finiti non deterministico o accettore a stati finiti non deterministico)

Un *automa a stati finiti non deterministico (NDA)* con alfabeto di ingresso  $X$  è una quadrupla:

$$M = (Q, \delta, q_0, F)$$

ove:

- per  $Q$ ,  $q_0$  ed  $F$  valgono le definizioni date per gli FSA;
- $\delta: Q \times X \rightarrow 2^Q$  è la funzione di transizione che assegna ad ogni coppia (stato-simbolo di ingresso)  $(q, x)$  un insieme  $\delta(q, x) \subseteq Q$  di possibili stati successivi. ■

Si può osservare che un NDA è un FSA con l'unica eccezione che, in corrispondenza di una coppia (stato-simbolo di ingresso)  $(q, x)$ , vi è un insieme di stati in cui l'automa può transitare (*stati successivi possibili*).

Inoltre, come per gli automi a stati finiti deterministici (FSA), possiamo estendere la funzione di transizione  $\delta$  al dominio  $2^Q \times X^*$  attraverso la funzione  $\delta^*$  definita come segue:

**Definizione 6.8** ( $\delta^*$  per NDA)

Dato un NDA  $M = (Q, \delta, q_0, F)$  con alfabeto di ingresso  $X$ , definiamo per induzione la funzione:

$$\delta^*: 2^Q \times X^* \rightarrow 2^Q$$

$$\begin{cases} \delta^*(p, \lambda) = p \\ \delta^*(p, wx) = \bigcup_{q \in \delta^*(p, w)} \delta(q, x) \text{ per ogni } p \in 2^Q \ (p \subset Q), x \in X, w \in X^* \end{cases}$$

Analogamente a quanto fatto per gli FSA, si dovrebbero riformulare le definizioni di parola accettata e di linguaggio accettato da un NDA.

La complicazione, rinveniente dalla computazione non deterministica dello stato successivo in cui un NDA transita, comporta che una stessa parola può indurre cammini multipli attraverso un NDA, alcuni che terminano in stati di accettazione, altri che terminano in stati di non accettazione.

### Esempio 6.1

Sia dato il seguente NDA  $M = (Q, \delta, q_0, F)$  (Figura 6.3):

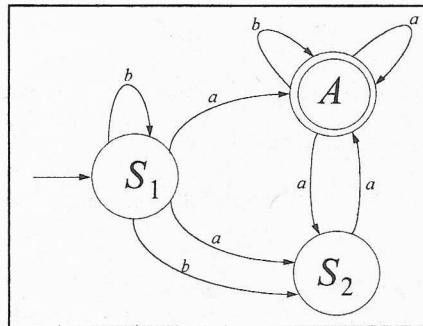


Figura 6.3

$$\delta(\{S_1\}, a) = \{A, S_2\}$$

$$\delta(\{S_1\}, b) = \{S_1, S_2\}$$

...

...

...

...

Supponiamo di avere la parola  $w = aba$ . Vogliamo stabilire se  $w$  è riconosciuta dall'automa (si veda la Figura 6.4).

Partiamo da  $S_1$ :

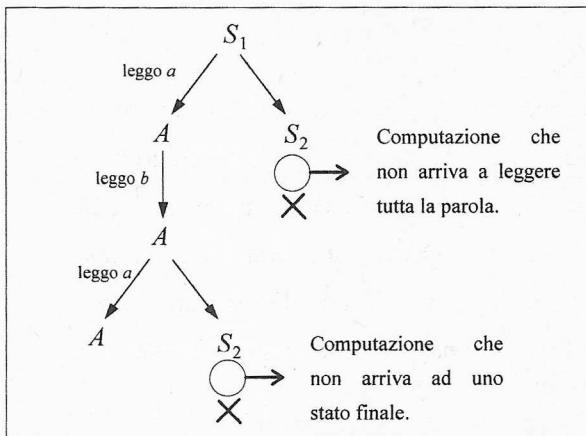


Figura 6.4

Quanto visto in Figura 6.4 è l'albero di tutte le possibili computazioni per la parola  $w = aba$ . Poiché almeno una computazione arriva ad uno stato finale, la stringa è riconosciuta.<sup>2</sup>

Possiamo ora formulare le definizioni di parola accettata e di linguaggio accettato da un NDA.

**Definizione 6.9** (Parola accettata o riconosciuta da un NDA)

Sia  $M = (Q, \delta, q_0, F)$  un NDA con alfabeto di ingresso  $X$ .

---

<sup>2</sup> Può succedere che si abbiano computazioni che non leggono tutta la parola in ingresso (nel caso  $\delta$  sia definita parzialmente) e computazioni che, pur leggendo per intero la parola in ingresso, non arrivano ad uno stato finale (non determinismo).

Una parola  $w \in X^*$  è accettata (o riconosciuta) da  $M$  se, partendo dallo stato iniziale  $q_0$ , esiste almeno un modo per  $M$  di portarsi in uno stato di accettazione alla fine della sequenza di ingresso  $w$ . In formule:

$$w \text{ accettata} \stackrel{\text{def}}{\Leftrightarrow} \exists p : p \in \delta^*(\{q_0\}, w) \cap F \Leftrightarrow \delta^*(\{q_0\}, w) \cap F \neq \emptyset \quad \blacksquare$$

### Esempio 6.2

Sia dato l'NDA dell'esempio 6.1. Calcoliamo  $\delta^*(\{S_1\}, aba)$ .

$$\delta^*(\{S_1\}, aba) = \bigcup_{q \in \delta^*(\{S_1\}, ab)} \delta(q, a)$$

$$\delta^*(\{S_1\}, ab) = \bigcup_{q' \in \delta^*(\{S_1\}, a)} \delta(q', b)$$

$$\delta^*(\{S_1\}, a) = \bigcup_{q'' \in \delta^*(\{S_1\}, \lambda)} \delta(q'', a)$$

$$\delta^*(\{S_1\}, \lambda) = \{S_1\}$$

$$\delta^*(\{S_1\}, a) = \delta(S_1, a) = \{A, S_2\}$$

$$\delta^*(\{S_1\}, ab) = \delta(A, b) \cup \delta(S_2, b) = \{A\} \cup \emptyset = \{A\}$$

$$\delta^*(\{S_1\}, aba) = \delta(A, a) = \{A, S_2\}$$

Poiché  $F = \{A\}$ , si ha:

$$\delta^*(\{S_1\}, aba) \cap F = \{A\} \neq \emptyset$$

e  $w = aba$  è accettata da  $M_1$ .

## 6.4 Linguaggi accettati da automi non deterministici.

**Definizione 6.10** (Linguaggio accettato o riconosciuto da un NDA)

Sia  $M = (Q, \delta, q_0, F)$  un NDA con alfabeto di ingresso  $X$ .

Il *linguaggio accettato o riconosciuto* da  $M$  è l'insieme delle parole su  $X$  accettate da  $M$ :

$$T(M) = \left\{ w \in X^* \mid \delta^*(\{q_0\}, w) \cap F \neq \emptyset \right\}$$

(è l'insieme delle parole  $w$  per le quali esiste almeno un cammino, etichettato con lettere di  $w$  nell'ordine da sinistra a destra, attraverso il diagramma degli stati che porta  $M$  dallo stato iniziale ad uno degli stati di accettazione). ■

**Definizione 6.11** (NDA equivalenti)

Siano  $M_1 = (Q_1, \delta_1, q_1, F_1)$  ed  $M_2 = (Q_2, \delta_2, q_2, F_2)$  due NDA di alfabeto di ingresso  $X$ .  $M_1$  ed  $M_2$  si dicono *equivalenti* se:

$$T(M_1) = T(M_2)$$
 ■

Risulta così definita, a partire da un NDA, la seguente classe di linguaggi:

$$\mathcal{L}_{NDL} = \left\{ L \in 2^{X^*} \mid \exists M, M \text{ è un NDA: } L = T(M) \right\}$$

## 6.5 Equivalenza delle classi di linguaggi accettati da automi a stati finiti deterministici e non deterministici.

**Teorema 6.2**

Le classi dei linguaggi  $\mathcal{L}_{FSL}$  e  $\mathcal{L}_{NDL}$  sono equivalenti (1<sup>a</sup> formulazione).

Sia  $L$  un linguaggio su  $X$ .  $L$  è un linguaggio a stati finiti se e solo se  $L = T(M)$  per qualche NDA  $M$  (2<sup>a</sup> formulazione).

### Dimostrazione 6.2<sup>3</sup>

⇒) Sia  $L \in 2^X$  ed  $L \in \mathcal{L}_{FSL}$ . Dalla definizione di linguaggio a stati finiti, si ha:

$$\exists M_1 : M_1 \text{ è un FSA, } M_1 = (Q_1, \delta_1, q_1, F_1)$$

con alfabeto di ingresso  $X$ :  $L = T(M_1)$ .

Sulla base di  $M_1$ , definiamo il seguente NDA con alfabeto di ingresso  $X$ :

$$M_2 = (Q_2, \delta_2, q_2, F_2)$$

ove:

- $Q_2 = Q_1$ ;
- $\delta_2$  è così definita:

$$\delta_2 : Q_2 \times X \rightarrow 2^{Q_2},$$

$$\delta_2(q, x) = \{\delta_1(q, x)\} \quad \forall q \in Q_2 = Q_1, x \in X;$$

- $q_2 = q_1$ ;
- $F_2 = F_1$ .

$M_2$  è un NDA ed inoltre accetta lo stesso linguaggio accettato da  $M_1$ , ossia si ha:

$$T(M_2) = T(M_1)$$

(Dimostrarlo, per esercizio, per induzione sulla lunghezza di una parola  $w$ ).

<sup>3</sup> Fatta sulla base della 2<sup>a</sup> formulazione.

$\Leftrightarrow$  Sia  $L \in 2^X^*$  ed  $L \in \mathcal{L}_{NDL}$ . Dalla definizione della classe di linguaggi  $\mathcal{L}_{NDL}$ , si ha:

$$\exists M, M \text{ è un NDA, } M = (Q, \delta, q_0, F)$$

con alfabeto di ingresso  $X$ :  $L = T(M)$ .

Si può definire allora il seguente algoritmo per la costruzione di un FSA equivalente all'NDA  $M$ :

**Algoritmo 6.1** (Trasformazione di un automa a stati finiti non deterministico in un automa deterministico equivalente)

Sia  $M = (Q, \delta, q_0, F)$  un automa accettore a stati finiti non deterministico di alfabeto di ingresso  $X$ .  $M$  può essere trasformato in un automa deterministico  $M'$  di alfabeto di ingresso  $X$  come segue:

$$M' = (Q', \delta', q'_0, F')$$

ove:

- i)  $Q' = 2^Q$ ;
- ii)  $q'_0 = \{q_0\}$ ;
- iii)  $F' = \{p \subseteq Q \mid p \cap F \neq \emptyset\}$ ;
- iv)  $\delta': Q' \times X \rightarrow Q' \quad \exists'$

$$\forall q' = \{q_1, q_2, \dots, q_i\} \in Q', \forall x \in X:$$

$$\delta'(q', x) = \delta'(\{q_1, q_2, \dots, q_i\}, x) = \bigcup_{j=1}^i \delta(q_j, x) = \bigcup_{q \in q'} \delta(q, x)$$

Si può dimostrare che  $M'$  è equivalente a  $M$ , ossia che  $T(M') = T(M)$ .

Ossia, dobbiamo dimostrare che:

$$\forall w \in X^*: w \in T(M') \Leftrightarrow w \in T(M)$$

Procediamo per induzione sulla lunghezza della parola  $w$ :  $|w|$ .

Passo base:  $|w| = 0$ ,  $w = \lambda$

$$\lambda \in T(M') \Rightarrow \delta'^*(q'_0, \lambda) \in F'$$

Dobbiamo dimostrare che:  $\delta^*(q_0, \lambda) \cap F \neq \emptyset$ .

Calcoliamo:

$$\delta'^*(q'_0, \lambda) = \delta'^*(\{q_0\}, \lambda) = \{q_0\} = \delta(q_0, \lambda) = \delta^*(q_0, \lambda)$$

Ma  $\{q_0\} \in F' \stackrel{\text{def}}{\Leftrightarrow} \{q_0\} \cap F \neq \emptyset$ .

Passo induttivo:  $w' = wa$

L'ipotesi di induzione è che:

$$\delta'^*(\{q_0\}, w) = \delta^*(q_0, w)$$

$$\begin{aligned} \delta'^*(q'_0, wa) &\stackrel{\text{def}}{=} \delta'^*(\{q_0\}, wa) \stackrel{\text{def}}{=} \\ &= \delta'\left(\delta'^*(\{q_0\}, w), a\right) \stackrel{\substack{\text{per ipotesi} \\ \text{di}}}{} = \delta'\left(\delta^*(q_0, w), a\right) \stackrel{\text{induzione}}{=} \delta'\left(\delta^*(q_0, w), a\right) \stackrel{\text{def}}{=} \bigcup_{q' \in \delta^*(q_0, w)} \delta(q', a) \end{aligned}$$

Quindi, poiché:

$$\delta^*(q_0, wa) = \bigcup_{q' \in \delta^*(q_0, w)} \delta(q', a)$$

si ha che:

$$\delta^*(q_0, wa) = \delta'^*(q'_0, wa)$$

e poiché per ipotesi:

$$\delta'^*(q'_0, wa) \in F'$$

si ha:

$$\delta^*(q_0, wa) \cap F \neq \emptyset \quad c.v.d.$$

## 6.6 Esercizi.

### Esercizio 6.1

Sia dato il seguente linguaggio:

$$L = \left\{ w \mid w \in \{a,b\}^*, w \text{ ha un numero pari di } a \text{ ed un numero dispari di } b \right\}$$

Costruire l'automa accettore a stati finiti che riconosce  $L$ .

Dobbiamo sintetizzare l'automa accettore  $M$  (Figura 6.5) tale che:

$$L = T(M)$$

$$M = (Q, \delta, q_0, F) \text{ con alfabeto } X = \{a,b\}$$

e con:

i)  $Q = \{q_0, q_1, q_2, q_3\}$  dove:

- $q_0$  = numero pari di  $a$  e di  $b$ ;
- $q_1$  = numero pari di  $a$  e numero dispari di  $b$ ;
- $q_2$  = numero dispari di  $a$  e numero pari di  $b$ ;
- $q_3$  = numero dispari di  $a$  e di  $b$ ;

ii) la funzione di transizione  $\delta$  è definita come segue:

- $\delta(q_0, a) = \delta(q_3, b) = q_2;$
- $\delta(q_0, b) = \delta(q_3, a) = q_1;$
- $\delta(q_1, a) = \delta(q_2, b) = q_3;$
- $\delta(q_1, b) = \delta(q_2, a) = q_0;$

iii)  $q_0$  è lo stato iniziale;

iv) l'insieme degli stati finali o di accettazione è  $F = \{q_1\}$ .

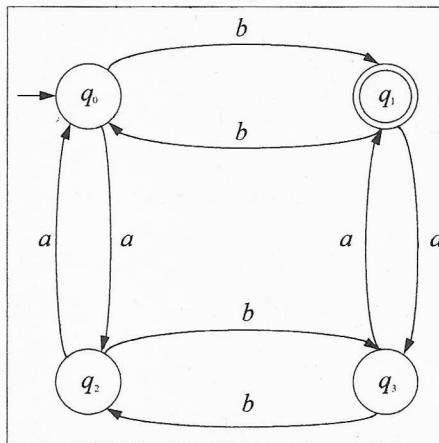


Figura 6.5

### Esercizio 6.2

Sia dato il seguente linguaggio:

$$L = \left\{ w \mid w \in \{a, b\}^*, w \neq \alpha a a \beta, \alpha, \beta \in \{a, b\}^* \right\}$$

(w non contiene due a consecutive)

Costruire l'automa accettore a stati finiti deterministico che riconosce  $L$ .

Dobbiamo sintetizzare l'automa accettore  $M$  tale che:

$$L = T(M)$$

$$M = (Q, \delta, q_0, F)$$
 con alfabeto  $X = \{a, b\}$

con:

i)  $Q = \{q_0, q_1, q_2\}$  dove:

- $q_0$  = parole su  $X = \{a, b\}$  che non contengono due o più  $a$  consecutive e terminano per  $b$ ;
- $q_1$  = parole su  $X = \{a, b\}$  che non contengono due o più  $a$  consecutive e terminano per  $a$ ;

- $q_2$  = parole su  $X = \{a, b\}$  che contengono due o più  $a$  consecutive (*stato pozza*);

ii) la funzione di transizione  $\delta$  è definita come segue:

$$\delta: Q \times X \rightarrow Q$$

- $\delta(q_0, a) = q_1;$
- $\delta(q_0, b) = \delta(q_1, b) = q_0;$
- $\delta(q_1, a) = \delta(q_2, a) = \delta(q_2, b) = q_2;$

iii)  $q_0$  è lo stato iniziale;

iv) l'insieme degli stati finali o di accettazione è  $F = \{q_0, q_1\}$ .

La tavola di transizione è dunque:

$\delta$	$q_0$	$q_1$	$q_2$
$a$	$q_1$	$q_2$	$q_2$
$b$	$q_0$	$q_0$	$q_2$

Il grafo degli stati è indicato in Figura 6.6:

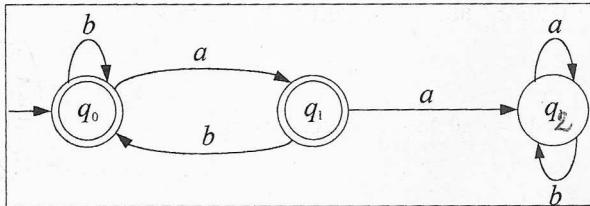


Figura 6.6

ed  $M$  è deterministico.

### Esercizio 6.3

Trasformare il seguente automa non deterministico  $M$  (Figura 6.7):

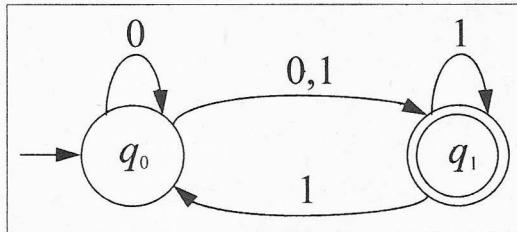


Figura 6.7

in un automa deterministico  $M'$  ad esso equivalente.

L'automa accettore a stati finiti deterministico  $M'$  tale che  $T(M') = T(M)$  si costruisce come segue.

Dato l'automa di Figura 6.7:

$$M = (Q, \delta, q_0, F) \text{ con } X = \{0,1\} \text{ come alfabeto di ingresso}$$

ove:

1)  $Q = \{q_0, q_1\}$ ;

2)  $F = \{q_1\}$

3) la funzione di transizione  $\delta: Q \times X \rightarrow 2^Q$  è definita dalla seguente tavola di transizione:

$\delta$	$q_0$	$q_1$
0	$\{q_0, q_1\}$	-
1	$\{q_1\}$	$\{q_0, q_1\}$

Definiamo  $M'$  come segue:

$$M' = (Q', \delta', q'_0, F') \text{ con } X = \{0,1\} \text{ come alfabeto di ingresso}$$

ove:

i)  $Q' = 2^Q = 2^{\{q_0, q_1\}}$ ;

ii)  $q'_0 = \{q_0\}$ ;

iii)  $F' = \{\{q_1\}, \{q_0, q_1\}\}$ ;

iv) la funzione di transizione  $\delta'$  è definita come segue:

$$\delta': Q' \times X \rightarrow Q'$$

- $\delta'(\{q_0\}, 0) = \delta(q_0, 0) = \{q_0, q_1\}$ ;
- $\delta'(\{q_0\}, 1) = \delta(q_0, 1) = \{q_1\}$ ;
- $\delta'(\{q_1\}, 0) = \delta(q_1, 0) = \text{non è definita}$ ;
- $\delta'(\{q_1\}, 1) = \delta(q_1, 1) = \{q_0, q_1\}$ ;
- $\delta'(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\}$ ;
- $\delta'(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0, q_1\}$ ;

La tavola di transizione che riassume la definizione della funzione  $\delta'$  è:

$\delta'$	$\{q_0\}$	$\{q_1\}$	$\{q_0, q_1\}$
0	$\{q_0, q_1\}$	-	$\{q_0, q_1\}$
1	$\{q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$

Il grafo degli stati di  $M'$  è indicato in Figura 6.8:

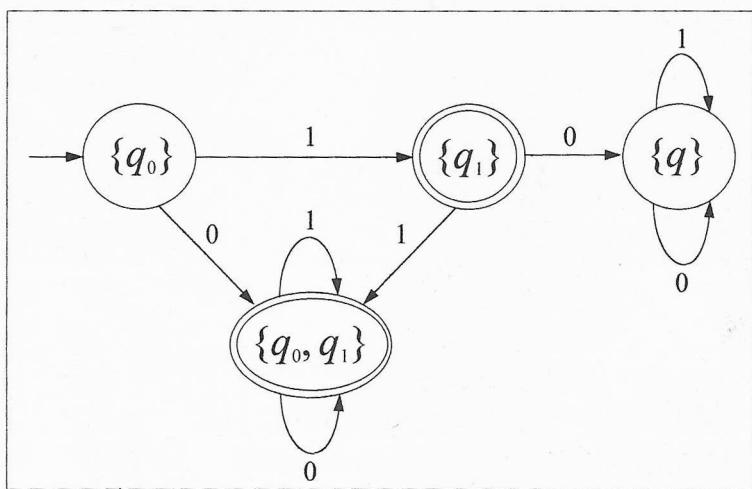


Figura 6.8

## 7. Linguaggi regolari, espressioni regolari e teorema di Kleene.

### *DESCRIZIONE ALGEBRICA DEI LINGUAGGI LINEARI DESTRI*

Grammatiche generative	Automi a stati finiti accettori	Espressioni regolari
<i>GENERATORI</i>	<i>RICONOSCITORI</i>	
procedure per		
<i>Costruire stringhe che appartengono al linguaggio</i>	<i>Stabilire se una stringa appartiene o no ad un linguaggio</i>	
<i>Definizioni operazionali</i>		<i>Definizione denotazionale</i>

### 7.1 Linguaggi regolari ed espressioni regolari.

Definiamo una nuova classe di linguaggi:

#### Definizione 7.1 (Linguaggio regolare)

Sia  $X$  un alfabeto . Un linguaggio  $L \subseteq X^*$  è regolare se:

- $L$  è finito;

oppure

- $L$  può essere ottenuto per induzione utilizzando una delle seguenti operazioni:
  - (1)  $L = L_1 \cup L_2$ , con  $L_1, L_2$  regolari;
  - (2)  $L = L_1 \cdot L_2$ , con  $L_1, L_2$  regolari;
  - (3)  $L = L_1^*$ , con  $L_1$  regolare.

■

Si noti che  $\emptyset$  e  $\{\lambda\}$  sono linguaggi regolari.

Denotiamo con  $\mathcal{L}_{REG}$  la classe dei linguaggi regolari.

### Definizione 7.2 (Espressione regolare)

Sia  $X$  un alfabeto . Una stringa  $R$  di alfabeto  $X \cup \{\lambda, +, *, :, \emptyset, (, )\}$  (con  $X \cap \{\lambda, +, *, :, \emptyset, (, )\} = \emptyset$ ) è una *espressione regolare* di alfabeto  $X$  se e solo se vale una delle seguenti condizioni:

- (i)  $R = \emptyset$ ;
- (ii)  $R = \lambda$ ;
- (iii)  $R = a$ , per ogni  $a \in X$ ;
- (iv)  $R = (R_1 + R_2)$ , con  $R_1, R_2$  espressioni regolari di alfabeto  $X$ ;
- (v)  $R = (R_1 \cdot R_2)$ , con  $R_1, R_2$  espressioni regolari di alfabeto  $X$ <sup>1</sup>;
- (vi)  $R = (R_1)^*$ , con  $R_1$  espressione regolare di alfabeto  $X$ .

■

---

<sup>1</sup> Invece di scrivere  $(R_1 \cdot R_2)$  si può anche scrivere  $(R_1 R_2)$ .

Ad ogni espressione regolare  $R$  corrisponde un linguaggio regolare  $S(R)$  definito nel modo seguente:

<i>ESPRESSIONE REGOLARE</i>	<i>LINGUAGGIO REGOLARE CORRISPONDENTE</i>
$\emptyset$	$\emptyset$
$\lambda$	$\{\lambda\}$
$a$	$\{a\}$
$(R_1 + R_2)$	$S(R_1) \cup S(R_2)$
$(R_1 \cdot R_2)$	$S(R_1) \cdot S(R_2)$
$(R_1)^*$	$(S(R_1))^*$

Da una espressione regolare si possono eliminare le coppie di parentesi superflue, tenuto conto che le operazioni ai punti (iv), (v) e (vi) sono elencate in ordine crescente di priorità.

### Proposizione 7.1

Un linguaggio su  $X$  è regolare se e solo se corrisponde ad una espressione regolare su  $X$ .

Quindi, denotato con  $\mathcal{R}$  l'insieme delle espressioni regolari di alfabeto  $X$ , definiamo la funzione:

$$S: \mathcal{R} \rightarrow 2^{X^*}$$

che ad ogni espressione regolare  $R$  associa il corrispondente linguaggio regolare  $S(R)$ .

Si ha dunque:

$$\mathcal{L}_{REG} = \left\{ L \in 2^{X^*} \mid \exists R \in \mathcal{R} : L = S(R) \right\}$$

### Esempi 7.1

$$ba^* \quad S(ba^*) = S(b) \cdot S(a^*) = S(b) \cdot (S(a))^* = \{b\} \cdot \{a\}^* = \\ = \text{insieme di tutte le parole su } X = \{a, b\} \text{ che} \\ \text{cominciano con una } b \text{ seguita eventualmente} \\ \text{soltanto da delle } a.$$

$$a^* \cdot b \cdot a^* \cdot b \cdot a^* \quad S(a^* \cdot b \cdot a^* \cdot b \cdot a^*) = S(a^*) \cdot S(b) \cdot S(a^*) \cdot S(b) \cdot S(a^*) = \\ = (S(a))^* \cdot S(b) \cdot (S(a))^* \cdot S(b) \cdot (S(a))^* = \\ = \{a\}^* \cdot \{b\} \cdot \{a\}^* \cdot \{b\} \cdot \{a\}^* = \\ = \text{insieme di tutte le parole su } X = \{a, b\} \\ \text{che contengono esattamente due } b.$$

$$(a+b)^* \quad S((a+b)^*) = (S(a+b))^* = (S(a) \cup S(b))^* = (\{a\} \cup \{b\})^* = \\ = \{a, b\}^* = \text{insieme di tutte le parole su } X = \{a, b\}.$$

$$(b+abb)^* \quad S((b+abb)^*) = (S(b+abb))^* = (S(b) \cup S(abb))^* = \\ = (S(b) \cup (S(a) \cdot S(b) \cdot S(b)))^* = \\ = (\{b\} \cup (\{a\} \cdot \{b\} \cdot \{b\}))^* = \\ = (\{b\} \cup \{abb\})^* = \{b, abb\}^* = \\ = \text{insieme di tutte le parole su } X = \{a, b\} \text{ in cui} \\ \text{ogni } a \text{ è immediatamente seguita da} \\ \text{almeno due } b.$$

### Osservazione 7.1

Un linguaggio regolare può essere descritto da più di una espressione regolare, ossia  $S : \mathcal{R} \rightarrow 2^{X^*}$  non è una funzione iniettiva.

### Esempi 7.2

Il linguaggio costituito da tutte le parole su  $X = \{a, b\}$  con  $a$  e  $b$  che si alternano (e che cominciano e terminano con  $b$ ) può essere descritto sia dall'espressione regolare

$$b \cdot (ab)^*$$

sia da

$$(ba)^* b$$

### Definizione 7.3 (Espressioni regolari equivalenti)

Due espressioni regolari  $R_1$  e  $R_2$  su  $X$  sono *equivalenti* (per abuso di notazione, scriviamo  $R_1 = R_2$ ) se e solo se  $S(R_1) = S(R_2)$ . ■

Si ha, pertanto:

$$b \cdot (ab)^* = (ba)^* b$$

## 7.2 Proprietà delle espressioni regolari.

Siano  $R_1$ ,  $R_2$  ed  $R_3$  espressioni regolari di alfabeto  $X$ , risulta:

$$1) \quad (R_1 + R_2) + R_3 = R_1 + (R_2 + R_3) = R_1 + R_2 + R_3 \quad \text{Proprietà associativa}$$

$$2) \quad R_1 + R_2 = R_2 + R_1 \quad \text{Proprietà commutativa}$$

- 3)  $R_1 + \emptyset = \emptyset + R_1$   $\emptyset$  è l'elemento neutro rispetto all'operazione "+" definita sulle espressioni regolari
- 4)  $R_1 + R_1 = R_1$  Idempotenza
- 5)  $(R_1 \cdot R_2) \cdot R_3 = R_1 \cdot (R_2 \cdot R_3) = R_1 \cdot R_2 \cdot R_3$  Proprietà associativa
- 6)  $R_1 \cdot R_2 \neq R_2 \cdot R_1$  In generale
- 7)  $R_1 \cdot \lambda = \lambda \cdot R_1 = R_1$   $\lambda$  è l'elemento neutro rispetto all'operazione ":" definita sulle espressioni regolari
- 8)  $R_1 \cdot \emptyset = \emptyset \cdot R_1 = \emptyset$   $\emptyset$  è l'elemento assorbente rispetto all'operazione ":".
- 9)  $R_1 \cdot (R_2 + R_3) = (R_1 \cdot R_2) + (R_1 \cdot R_3)$  Proprietà distributiva di ":" rispetto all'operazione "+" (distributività sinistra).
- 10)  $(R_1 + R_2) \cdot R_3 = (R_1 \cdot R_3) + (R_2 \cdot R_3)$  Proprietà distributiva di ":" rispetto all'operazione "+" (distributività destra).
- 11)  $(R_1)^* = (R_1)^* \cdot (R_1)^* = ((R_1)^*)^* = (\lambda + R_1)^*$
- 12)  $(\emptyset)^* = (\lambda)^* = \lambda$

$$13) \quad (R_1)^* = \lambda + R_1 + R_1^2 + \dots + R_1^n + (R_1^{n+1} \cdot R_1^*)$$

Caso particolare:

$$(R_1)^* = \lambda + R_1 \cdot R_1^* = \lambda + R_1^* \cdot R_1$$

$$14) \quad (R_1 + R_2)^* = (R_1^* + R_2^*)^* = (R_1^* \cdot R_2^*)^* = \\ = (R_1^* \cdot R_2)^* \cdot R_1^* = R_1^* \cdot (R_2 \cdot R_1^*)^*$$

$$15) \quad (R_1 + R_2)^* \neq R_1^* + R_2^* \quad \text{In generale}$$

$$16) \quad R_1^* \cdot R_1 = R_1 \cdot R_1^*$$

$$17) \quad R_1 \cdot (R_2 \cdot R_1)^* = (R_1 \cdot R_2)^* \cdot R_1$$

$$18) \quad (R_1^* \cdot R_2)^* = \lambda + (R_1 + R_2)^* \cdot R_2$$

$$19) \quad (R_1 \cdot R_2^*)^* = \lambda + R_1 \cdot (R_1 + R_2)^*$$

20) Supponiamo che  $R_2 \neq \lambda$ :

$$R_1 = R_2 \cdot R_1 + R_3 \text{ se e solo se } R_1 = R_2^* \cdot R_3$$

$$R_1 = R_1 \cdot R_2 + R_3 \text{ se e solo se } R_1 = R_3 \cdot R_2^*$$

Per esercizio: dimostrare le 1) - 20).

Aiuto: le 1) - 5) e le 7) - 14) si dimostrano ricorrendo alla funzione  $S$ ; comunque la maggior parte delle 1) - 20) può essere provata con una tecnica generale, detta *dimostrazione mediante riparsificazione*.

Illustriamo questa tecnica dimostrando la 17):

$$R_1 \cdot (R_2 \cdot R_1)^* = (R_1 \cdot R_2)^* \cdot R_1$$

Si consideri una qualunque parola:

$$w \in S\left(R_1 \cdot (R_2 \cdot R_1)^*\right), w = r_1^0 \left(r_2^{-1} \cdot r_1^{-1}\right) \cdot \left(r_2^{-2} \cdot r_1^{-2}\right) \cdots \left(r_2^{-n} \cdot r_1^{-n}\right), n \geq 0,$$

ove:

$$r_1^i \in S(R_1), i = 0, 1, 2, \dots, n$$

$$r_2^j \in S(R_2), j = 1, 2, \dots, n$$

Riparsificando  $w$  ed utilizzando la proprietà associativa della concatenazione, si ha:

$$w = \left(r_1^0 \cdot r_2^{-1}\right) \cdot \left(r_1^1 \cdot r_2^{-2}\right) \cdots \left(r_1^{n-1} \cdot r_2^{-n}\right) \cdot r_1^n$$

dunque:

$$w \in S\left(\left(R_1 \cdot R_2\right)^* \cdot R_1\right)$$

Da cui:

$$S\left(R_1 \cdot (R_2 \cdot R_1)^*\right) \subseteq S\left(\left(R_1 \cdot R_2\right)^* \cdot R_1\right)$$

In modo analogo si dimostra:

$$S\left(R_1 \cdot (R_2 \cdot R_1)^*\right) \supseteq S\left(\left(R_1 \cdot R_2\right)^* \cdot R_1\right)$$

Un'altra tecnica comune per dimostrare tali proprietà è semplicemente quella di utilizzare proprietà già note.

Mostreremo ora come si usano le proprietà delle espressioni regolari per provare l'equivalenza di espressioni regolari.

### Esempio 7.3

Dimostrare la seguente equivalenza:

$$(b + aa^*b) + (b + aa^*b) \cdot (a + ba^*b)^* \cdot (a + ba^*b) = a^*b \cdot (a + ba^*b)^*$$

$$\begin{aligned}
 (b + aa^*b) + (b + aa^*b) \cdot (a + ba^*b)^* \cdot (a + ba^*b) &= (b + aa^*b) \cdot \left[ \lambda + (a + ba^*b)^* (a + ba^*b) \right] = \\
 &\stackrel{13)}{=} (b + aa^*b) \cdot (a + ba^*b)^* = \\
 &\stackrel{10)}{=} (\lambda + aa^*) \cdot b \cdot (a + ba^*b)^* = \\
 &\stackrel{13)}{=} a^*b \cdot (a + ba^*b)^*
 \end{aligned}$$

### 7.3 Teorema di Kleene.

Teorema 7.1 (di Kleene)

$$\mathcal{L}_3 \equiv \mathcal{L}_{FSL} \equiv \mathcal{L}_{REG}$$

#### Dimostrazione 7.1

Lo schema della dimostrazione è il seguente:

- 1)  $\mathcal{L}_3 \subset \mathcal{L}_{FSL}$   $(\mathcal{L}_{FSL} \subset \mathcal{L}_3)$
- 2)  $\mathcal{L}_{FSL} \subset \mathcal{L}_{REG}$
- 3)  $\mathcal{L}_{REG} \subset \mathcal{L}_3$

- 1)  $\mathcal{L}_3 \subset \mathcal{L}_{FSL}$

Sia  $L \in \mathcal{L}_3 \overset{\text{def}}{\iff} \exists G = (X, V, S, P), G$  di tipo '3':  $L(G) = L$ .

Vogliamo costruire un automa a stati finiti  $M = (Q, \delta, q_0, F)$  tale che  $T(M) = L(G)$ .

Allo scopo si fornisce il seguente algoritmo per la costruzione di un automa a stati finiti non deterministico che riconosce il linguaggio generato da una fissata grammatica lineare destra.

**Algoritmo 7.1** (Costruzione di un automa a stati finiti non deterministico equivalente ad una grammatica lineare destra).

Data una grammatica lineare destra:

$$G = (X, V, S, P)$$

l'automa accettore a stati finiti equivalente ( $T(M) = L(G)$ ) viene costruito come segue:

$$M = (Q, \delta, q_0, F)$$

I.  $X$  come alfabeto di ingresso;

II.  $Q = V \cup \{q\}$ ,  $q \notin V$ ;

III.  $q_0 = S$ ;

IV.  $F = \{q\} \cup \{B \mid B \rightarrow \lambda \in P\}$ ;

V.  $\delta: Q \times X \rightarrow 2^Q$  

V.a.  $\forall B \rightarrow aC \in P, C \in \delta(B, a)$ ;

V.b.  $\forall B \rightarrow a \in P, q \in \delta(B, a)$ .

L'algoritmo può generare un automa *non deterministico* per effetto dei passi

V.a. e V.b. Si può facilmente constatare che, se:

$$w = x_1 x_2 \dots x_n \in L(G)$$

$w$  può essere generata da una derivazione del tipo:

$$S \Rightarrow x_1 X_2 \Rightarrow x_1 x_2 X_3 \Rightarrow \dots \Rightarrow x_1 x_2 \dots x_{i-1} X_i \Rightarrow x_1 x_2 \dots x_i$$

Dalla definizione data, l'automa  $M$ , esaminando la stringa  $w = x_1 x_2 \dots x_n$ , compie una serie di mosse (o transizioni) che lo portano dallo stato  $S$  ad  $X_1, X_2, \dots, X_i$  e  $q$ ; pertanto  $L(G) \subseteq T(M)$ .

In modo del tutto analogo, ogni  $w$  in  $T(M)$  comporta una sequenza di mosse dell'automa a cui corrisponde una derivazione in  $G$ , e pertanto  $T(M) \subseteq L(G)$ .

Se ne deduce che:

$$L(G) = T(M) \quad c.v.d.$$

Sebbene non sia strettamente necessario per la dimostrazione del Teorema di Kleene, per il suo interesse pratico si riporta di seguito l'algoritmo per la costruzione di una grammatica lineare destra che genera il linguaggio accettato da un automa a stati finiti.

Tale algoritmo costituisce una dimostrazione costruttiva del seguente risultato:

$$\mathcal{L}_{FSL} \subset \mathcal{L}$$

**Algoritmo 7.2** (Costruzione di una grammatica lineare destra equivalente ad un automa accettore a stati finiti).

Sia dato un automa accettore a stati finiti:

$$M = (Q, \delta, q_0, F)$$

con alfabeto di ingresso  $X$ .

La grammatica lineare destra  $G$  equivalente a  $M$ , ossia tale che  $L(G) = T(M)$ , si costruisce come segue:

$$G = (X, V, S, P)$$

I.  $X = \text{alfabeto di ingresso di } M$ ;

II.  $V = Q$ ;

III.  $S = q_0$ ;

IV.  $P = \{q \rightarrow xq' \mid q' \in \delta(q, x)\} \cup \{q \rightarrow x \mid \delta(q, x) \in F\}$ .

Si lascia al lettore il compito di dimostrare che  $L(G) = T(M)$ . c.v.d.

2)  $\mathcal{L}_{FSL} \subset \mathcal{L}_{REG}$

Sia  $L \in \mathcal{L}_{FSL} \stackrel{\text{def}}{\Leftrightarrow} \exists M = (Q, \delta, q_0, F)$  automa a stati finiti con alfabeto di ingresso  $X$  tale che  $T(M) = L$ .

Supponiamo  $Q = \{q_0, q_1, \dots, q_n\}$ .

Definiamo il seguente linguaggio:

$$R_{ij} = \left\{ w \in X^* \mid \delta^*(q_i, w) = q_j \right\}$$

costituito da tutte e sole quelle parole che fanno transitare  $M$  dallo stato  $q_i$  allo stato  $q_j$ .

Dalla definizione di linguaggio accettato da un automa a stati finiti (Definizione 6.4), segue immediatamente:

$$L = \bigcup_{q_j \in F} R_{0,j}$$

ossia  $L$  può essere riguardato come l'insieme delle parole che fanno transitare  $M$  dallo stato iniziale  $q_0$  ad uno qualunque degli stati finali.

Se dimostriamo che ciascuno degli  $R_{ij}$ ,  $0 \leq i, j \leq n$ , è un linguaggio regolare, allora risulterà dimostrato che  $L$  è regolare, in quanto unione finita di linguaggi regolari.

Sia:

$$R_{ij}^k = \left\{ w \in X^* \mid \begin{array}{l} w \text{ fa transitare } M \text{ da } q_i \text{ a } q_j \text{ senza passare per nessuno} \\ \text{degli stati } q_k, q_{k+1}, \dots, q_n \end{array} \right\}$$

Si noti che:

$$R_{ij}^{n+1} = R_{ij}. \quad (*)$$

Dimostriamo:

$$R_{ij}^k \in \mathcal{L}_{REG} \quad \forall i, j: 0 \leq i, j \leq n$$

per induzione su  $k$ .

Passo base

$k=0$

$$R_{ij}^0 = \left\{ x \in X \mid \delta(q_i, x) = q_j \right\}$$

è finito (poiché è un sottoinsieme di  $X$ ) e quindi è per definizione regolare.

Passo induttivo

$R_{ij}^k$  regolare  $\forall i, j: 0 \leq i, j \leq n$

Si intende dimostrare che  $R_{ij}^{k+1}$  è regolare.

Sia  $w \in R_{ij}^{k+1}$  (Figura 7.1); per definizione la lettura di  $w$  non fa transitare  $M$  in nessuno degli stati  $q_{k+1}, q_{k+2}, \dots, q_n$ .

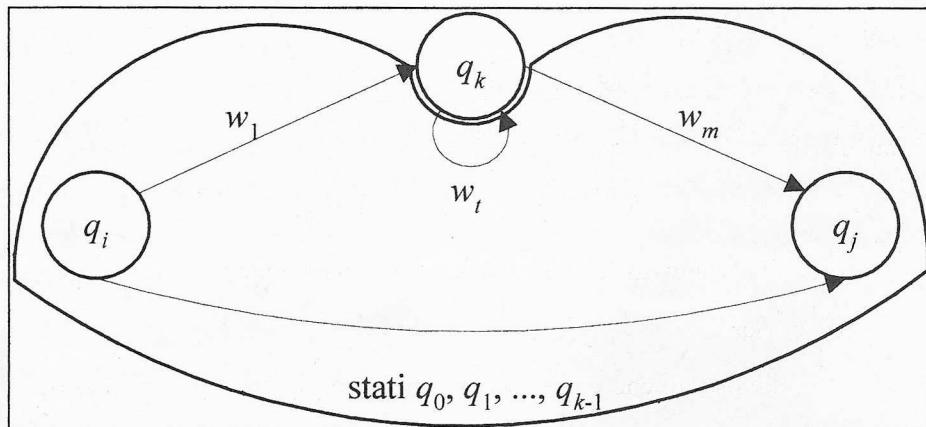


Figura 7.1

Si possono verificare due casi:

- a)  $w$  non fa transitare  $M$  in  $q_k$ , ossia  $w \in R_{ij}^k$  che, per ipotesi di induzione, è regolare.

Ciò implica che  $R_{ij}^{k+1}$  è regolare.

b)  $w$  fa transitare  $M$  in  $q_k$ .

Ci resta da dimostrare che  $R_{ij}^{k+1}$  è regolare anche nel caso b).

In questo caso,  $w$  può essere riguardata come la concatenazione di  $m$  segmenti,  $m \geq 2$ :

$$w = w_1 w_2 \dots w_{m-1} w_m \quad m \geq 2$$

ove (con riferimento a Figura 7.1):

$$w_1 \in R_{ik}^k \quad w_t \in R_{kk}^k \quad \forall t: 2 \leq t \leq m \quad w_m \in R_{kj}^k$$

Si ha dunque:

$$w \in R_{ik}^k \cdot (R_{kk}^k)^{m-2} \cdot R_{kj}^k \quad \forall m \geq 2$$

da cui:

$$w \in R_{ik}^k \cdot (R_{kk}^k)^* \cdot R_{kj}^k$$

Dunque:

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

e  $R_{ij}^{k+1}$  risulta espresso come unione, concatenazione ed iterazione di linguaggi regolari.

Risulta così dimostrato che  $R_{ij}^k$  è regolare, per ogni  $i, j$ ,  $0 \leq i, j \leq n$  e per ogni  $k$ ,  $0 \leq k \leq n$ .

Si ha dunque:

$$L = \bigcup_{q_j \in F} R_{0j} = \text{per la } (*)$$

$$= \bigcup_{q_j \in F} R_{0j}^{n+1}$$

è un linguaggio regolare.

c.v.d.

3)  $\mathcal{L}_{REG} \subset \mathcal{L}_3$

Sia  $L \in \mathcal{L}_{REG}$ . Per definizione di linguaggio regolare (Definizione 7.1), o  $L$  è finito oppure  $L$  può essere ottenuto induttivamente come segue:

- a)  $L = L_1 \cup L_2$ ,  $L_1, L_2$  regolari
- b)  $L = L_1 \cdot L_2$ ,  $L_1, L_2$  regolari
- c)  $L = L_1^*$   $L_1$  regolare

Procediamo per *induzione sulla costruzione* di  $L$ .

Passo Base (induzione sulla costruzione di  $L$ )

$L$  è finito

$$\begin{aligned} L &= \{w_1, w_2, \dots, w_n\} \quad n \geq 0 \\ L &= L_1 \cup L_2 \cup \dots \cup L_n = \bigcup_{i=1}^n L_i \end{aligned}$$

ove  $L_i = \{w_i\}$ ,  $1 \leq i \leq n$ .

Abbiamo ricondotto la dimostrazione a provare che  $L_i$  è lineare destro, per ogni  $i$ ,  $1 \leq i \leq n$ .

Procediamo per *induzione* su  $n$ .

Passo Base (induzione su  $n$ )

$n = 0$

Esiste sicuramente una grammatica lineare destra che genera il linguaggio  $L = \emptyset$  (si consideri, per esempio, la grammatica  $G: S \rightarrow aA$ ).

Passo induttivo (induzione su  $n$ )

Sia  $L = \{w_1, w_2, \dots, w_n, w_{n+1}\}$  un linguaggio regolare ( $n \geq 0$ ).

$$L = L_1 \cup L_2 \cup \dots \cup L_{n+1} = \bigcup_{i=1}^{n+1} L_i$$

ove  $L_i = \{ w_i \}$ ,  $1 \leq i \leq n+1$ .

Se dimostriamo che ogni  $L_i$ ,  $1 \leq i \leq n+1$ , è un linguaggio lineare destro, allora anche  $L$  risulterà lineare destro (per la proprietà di chiusura di  $\mathcal{L}_3$  rispetto all'unione - Teorema 5.3).

Consideriamo  $L_i = \{ w_i \}$ ,  $1 \leq i \leq n+1$ .

Poiché  $(X^*, \cdot)$  è il monoide libero generato da  $X$  (si veda Osservazione 2.1), si ha:

$$w_i = x_{i1} \cdot x_{i2} \cdot \dots \cdot x_{im_i} \quad x_{ij} \in X \quad 1 \leq i \leq n+1, 1 \leq j \leq m_i, m_i \geq 0$$

$$L_i = \{ w_i \} = \{ x_{i1} \} \cdot \{ x_{i2} \} \cdot \dots \cdot \{ x_{im_i} \} = L_{i1} \cdot L_{i2} \cdot \dots \cdot L_{im_i}$$

e  $G_{ij}: S \rightarrow x_{ij}$  è la grammatica lineare destra che genera  $L_{ij}$ ,  $1 \leq i \leq n+1$ ,  $1 \leq j \leq m_i$ ,  $m_i \geq 0$  ( $L_{ij} = L(G_{ij})$ ).

Dunque  $L$  è ottenuto per unione e concatenazione di linguaggi lineari destri. Per le proprietà di chiusura di  $\mathcal{L}_3$  rispetto alle suddette operazioni (Teorema 5.3),  $L$  risulta essere un linguaggio lineare destro.

Passo induttivo (induzione sulla costruzione di  $L$ )

$L$  è ottenuto induttivamente come segue:

- a)  $L = L_1 \cup L_2$ ,  $L_1, L_2$  regolari
- b)  $L = L_1 \cdot L_2$ ,  $L_1, L_2$  regolari
- c)  $L = L_1^*$   $L_1$  regolare

In tutti e tre i casi, la tesi  $L$  è *lineare destro* segue immediatamente dalla ipotesi di induzione ( $L_1, L_2$  *lineari destri*) e dal Teorema 5.3. *c.v.d.*

## 7.4 Pumping Lemma per i linguaggi regolari.

**Teorema 7.2** (Pumping Lemma per i linguaggi regolari)

Sia  $M = (Q, \delta, q_0, F)$  un automa accettore a stati finiti con  $n$  stati ( $|Q| = n$ ) e sia  $z \in T(M)$ ,  $|z| \geq n$ .

Allora  $z$  può essere scritta come  $uvw$ , e  $uv^*w \subset T(M)$  (ossia  $\forall i, i \geq 0 : uv^i w \in T(M)$ ).



### Dimostrazione 7.2

Sia  $z = x_1 x_2 \dots x_k$ ,  $z \in T(M)$ .

Possiamo rappresentare il comportamento dell'automa  $M$ , con ingresso  $z$ , come segue (Figura 7.2):

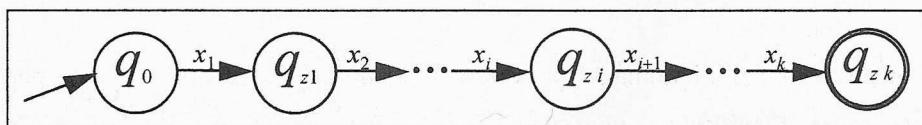


Figura 7.2

Se si ha:  $|z| \geq n$ , nella Figura 7.2 devono comparire almeno  $n+1$  stati ma, poiché  $M$  ha solo  $n$  stati distinti ( $|Q| = n$ ) almeno uno stato tra  $q_0, q_{z1}, q_{z2}, \dots, q_{zk}$  deve comparire due volte.

Supponiamo che si abbia  $q_{zi} = q_{zj}$ ,  $i < j$ .

Si ha dunque la situazione rappresentata in Figura 7.3.

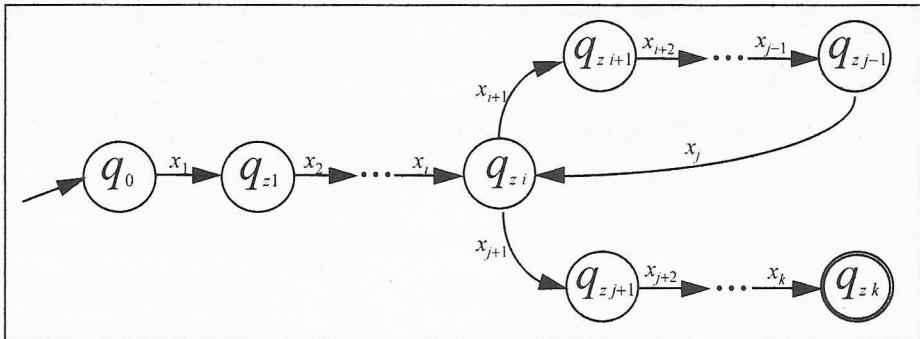


Figura 7.3

Possiamo scrivere  $z$  nella forma:

$$z = uvw$$

ove:

$$u = x_1 \ x_2 \dots \ x_i,$$

$$v = x_{i+1} \ x_{i+2} \dots \ x_j,$$

$$w = x_{j+1} \ x_{j+2} \dots \ x_k.$$

Poiché  $z \in T(M)$ , l'automa  $M$ , per effetto dell'ingresso di  $z = uvw$ , si porta per definizione in uno stato finale ( $\delta^*(q_0, z) \in F$ ).

Ma è immediato osservare che tale stato è lo stesso in cui  $M$  si porta per effetto dell'ingresso delle parole  $uw, uv^2w, \dots, uv^iw$ ,  $i \geq 0$ .

Dunque si ha:  $uv^iw \in T(M)$ ,  $i \geq 0$ .

c.v.d.

## 7.5 Esercizi.

### Esercizio 7.1

Determinare una grammatica lineare destra che genera il linguaggio descritto dalla seguente espressione regolare:

$$b^* + (ab)^*$$

Il linguaggio relativo all'espressione regolare è:

$$\begin{aligned} S(b^* + (ab)^*) &= S(b^*) \cup S((ab)^*) = (S(b))^* \cup (S(ab))^* = \{b\}^* \cup (S(a) \cdot S(b))^* = \\ &= \{b\}^* \cup (\{a\} \cdot \{b\})^* = \{b\}^* \cup \{ab\}^* \end{aligned}$$

Costruiamo dapprima la grammatica  $G_1$  tale che

$$L(G_1) = S(b^*) = \{b\}^*$$

$$G_1 = (X_1, V_1, S_1, P_1)$$

ove:

$$X_1 = \{b\} \quad V_1 = \{S_1\} \quad P_1 = \{S_1 \rightarrow bS_1 \mid \lambda\}.$$

Costruiamo ora la grammatica  $G_2$  tale che

$$L(G_2) = S(ab) = \{ab\}$$

$$G_2 = (X_2, V_2, S_2, P_2)$$

ove:

$$X_2 = \{a, b\} \quad V_2 = \{S_2, B_2\} \quad P_2 = \{S_2 \rightarrow aB_2, B_2 \rightarrow b\}.$$

La grammatica  $G_3$  tale che

$$L(G_3) = S((ab)^*) = \{ab\}^*$$

è dunque (si veda Capitolo 5, Tavola 1):

$$G_3 = (X_3, V_3, S_3, P_3)$$

ove:

$$X_3 = X_2 = \{a, b\} \quad V_3 = V_2 \cup \{S_3\}$$

$$\begin{aligned} P_3 &= \{S_3 \rightarrow \lambda\} \cup (P_2 - \{S_2 \rightarrow \lambda\}) \cup \{S_3 \rightarrow w \mid S_2 \rightarrow w \in P_2\} \cup \{A \rightarrow bS_3 \mid A \rightarrow b \in P_2\} = \\ &= \{S_3 \rightarrow \lambda\} \cup P_2 \cup \{S_3 \rightarrow aB_2\} \cup \{B_2 \rightarrow bS_3\} = \\ &= \{S_3 \rightarrow \lambda, S_2 \rightarrow aB_2, B_2 \rightarrow b, S_3 \rightarrow aB_2, B_2 \rightarrow bS_3\} = \\ &= \{S_3 \rightarrow \lambda \mid aB_2, S_2 \rightarrow aB_2, B_2 \rightarrow b \mid bS_3\} \end{aligned}$$

Si osservi che il nonterminale  $S_2$  non compare nella parte destra di nessuna produzione in  $P_3$  ( $S_2$  è un nonterminale inutile - si veda Definizione 8.12), dunque l'intera produzione ( $S_2 \rightarrow aB_2$ ) può essere rimossa da  $P_3$  senza alterare il linguaggio generato da  $G_3$ .

Quindi  $P_3$  diventa:

$$P_3 = \{S_3 \rightarrow \lambda \mid aB_2, B_2 \rightarrow b \mid bS_3\}$$

La grammatica  $G$  tale che:

$$L(G) = S(b^*) \cup S((ab)^*) = S(b^* + (ab)^*)$$

è dunque:

$$G = (X, V, S, P)$$

ove:

$$X = X_1 \cup X_3 = \{a, b\} \quad V = V_1 \cup V_3 = V_1 \cup V_2 \cup \{S_3\} \cup \{S\} = \{S, S_1, B_2, S_3\}$$

$$\begin{aligned} P &= \{S \rightarrow w \mid S_1 \rightarrow w \in P_1\} \cup \{S \rightarrow w \mid S_3 \rightarrow w \in P_3\} \cup P_1 \cup P_3 = \\ &= \{S \rightarrow bS_1 \mid \lambda\} \cup \{S \rightarrow aB_2 \mid \lambda\} \cup \{S_1 \rightarrow bS_1 \mid \lambda\} \cup \{S_3 \rightarrow \lambda \mid aB_2, B_2 \rightarrow b \mid bS_3\} = \\ &= \{S \rightarrow bS_1 \mid aB_2 \mid \lambda, S_1 \rightarrow bS_1 \mid \lambda, S_3 \rightarrow aB_2 \mid \lambda, B_2 \rightarrow bS_3 \mid b\} \end{aligned}$$

### Esercizio 7.2

Data la seguente grammatica lineare destra:

$$G = (X, V, S, P)$$

ove:

$$X = \{a, b, c\} \quad V = \{S, A, B\}$$

$$P = \{(1) S \rightarrow bA \mid aS \mid b, (2) A \rightarrow aB \mid cS \mid a, (3) B \rightarrow bA \mid cB \mid c\}$$

Determinare un'espressione regolare che denota il linguaggio  $L(G)$ .

Senza rischio di confusione, denotiamo con  $A$ ,  $B$  ed  $S$  gli insiemi delle stringhe derivabili in  $G$  dai nonterminali  $A$ ,  $B$  ed  $S$ , rispettivamente.

Dalla produzione (2) risulta:

$$A = \{a\} \cdot B \cup \{c\} \cdot S \cup \{a\}$$

che, per brevità, scriviamo nella forma:

$$A = aB \cup cS \cup a \tag{2'}$$

Sostituendo in (1) la (2'), si ha:

$$S = b \cdot (aB \cup cS \cup a) \cup aS \cup b$$

che, per la proprietà distributiva della concatenazione rispetto all'unione, è uguale a:

$$S = baB \cup bcS \cup ba \cup aS \cup b$$

Applicando la proprietà commutativa per l'unione, la proprietà associativa per la concatenazione e la distributività della concatenazione rispetto all'unione si ha:

$$S = baB \cup (bc \cup a)S \cup (ba \cup b) \tag{1'}$$

Sostituendo in (3) la (2'), si ottiene:

$$B = b \cdot (aB \cup cS \cup a) \cup cB \cup c = (ba \cup c)B \cup (bcS \cup ba \cup c) \tag{3'}$$

Osserviamo la (1') e la (3').

Se denotiamo con  $A$ ,  $B$  ed  $S$  tre espressioni regolari (oltre che gli insiemi di stringhe derivabili dai nonterminali  $A$ ,  $B$  ed  $S$ ), si ha che l'identità tra espressioni regolari corrispondente alla (1') è:

$$S = baB + (bc + a)S + (ba + b) \quad (1'')$$

mentre l'identità tra espressioni regolari corrispondente alla (3') è:

$$B = (ba + c)B + (bcS + ba + c) \quad (3'')$$

In entrambi i casi ci troviamo di fronte ad una identità del tipo:

$$R_1 = R_2 \cdot R_1 + R_3 \text{ con } R_2 \neq \lambda$$

Per la proprietà 20) sulle espressioni regolari, si ha:

$$R_1 = R_2^* \cdot R_3$$

Dunque la (3'') diventa:

$$B = (ba + c)^* \cdot (bcS + ba + c) \quad (3''')$$

Sostituendo la (3''') nella (1''), si ottiene:

$$\begin{aligned} S &= \left[ ba \cdot (ba + c)^* \cdot (bcS + ba + c) \right] + (bc + a)S + (ba + b) = \\ &= ba \cdot (ba + c)^* bcS + (a + bc)S + ba \cdot (ba + c)^* \cdot (ba + c) + ba + b = \\ &= \left( ba \cdot (ba + c)^* bc + a + bc \right) S + ba \cdot (ba + c)^* \cdot (ba + c) + ba + b = \end{aligned}$$

per la proprietà 20) sulle espressioni regolari

$$= \left( ba \cdot (ba + c)^* bc + a + bc \right)^* \cdot \left( ba \cdot (ba + c)^* (ba + c) + ba + b \right)$$

### Esercizio 7.3

Sia  $L$  il linguaggio denotato dalla seguente espressione regolare:

$$(aa + aaa)^*$$

- 1) Trovare un automa a stati finiti che riconosce  $L$ .

- 2) Trasformare l'automa non deterministico trovato al punto 1) in un automa deterministico equivalente.
- 1) Determiniamo innanzitutto due grammatiche lineari destre  $G_1$  e  $G_2$  tali che:

$$L(G_1) = S(aa) = \{aa\}; \quad L(G_2) = S(aaa) = \{aaa\}$$

$$G_1 = (X, V_1, S_1, P_1) \quad G_2 = (X, V_2, S_2, P_2)$$

$$X = \{a\}$$

$$V_1 = \{S_1, A\} \quad V_2 = \{S_2, B, C\}$$

$$P_1 = \{S_1 \rightarrow aa, A \rightarrow a\} \quad P_2 = \{S_2 \rightarrow aaa, B \rightarrow aC, C \rightarrow a\}$$

Determiniamo poi la grammatica lineare destra  $G_3$  tale che:

$$L(G_3) = L(G_1) \cup L(G_2) = S(aa) \cup S(aaa) = S(aa + aaa)$$

$$G_3 = (X, V_3, S_3, P_3)$$

dove:

$$V_3 = V_1 \cup V_2 \cup \{S_3\} = \{S_1, S_2, S_3, A, B, C\}$$

Le produzioni di  $G_3$  si ottengono come segue:

$$P_3 = \{S_3 \rightarrow w \mid S_1 \rightarrow w \in P_1\} \cup \{S_3 \rightarrow w \mid S_2 \rightarrow w \in P_2\} \cup P_1 \cup P_2$$

Dunque si ha:

$$P_3 = \{S_3 \rightarrow aA \mid aB, S_1 \rightarrow aA, A \rightarrow a, S_2 \rightarrow aB, B \rightarrow aC, C \rightarrow a\}$$

da cui possiamo eliminare le produzioni che presentano  $S_1$  o  $S_2$  nella parte sinistra ( $S_1$  ed  $S_2$  sono nonterminali inutili perché non sono presenti nella parte destra di nessuna produzione e né  $S_1$  né  $S_2$  sono il simbolo distintivo di  $G_3$ ). Per cui  $P_3$  diventa:

$$P_3 = \{S_3 \rightarrow aA \mid aB, A \rightarrow a, B \rightarrow aC, C \rightarrow a\}$$

Determiniamo ora la grammatica lineare destra  $G$  tale che:

$$L(G) = (L(G_3))^* = S((aa + aaa)^*)$$

$$G = (X, V, S, P)$$

dove:

$$V = V_3 \cup \{S\} = \{S, S_3, A, B, C\}$$

Le produzioni di  $G$  si ottengono come segue (si veda Tavola 1):

$$\begin{aligned} P = \{S \rightarrow \lambda\} &\cup \left(P_3 - \{S_3 \rightarrow \lambda\}\right) \cup \{S \rightarrow w \mid S_3 \rightarrow w \in P_3\} \cup \\ &\cup \{A \rightarrow bS \mid A \rightarrow b \in P_3\} \cup \{A \rightarrow bS \mid A \rightarrow bB \in P_3, b \neq \lambda, B \rightarrow \lambda \in P_1\} \end{aligned}$$

Dunque si ha:

$$\begin{aligned} P = \{S \rightarrow \lambda\} &\cup \{S_3 \rightarrow aA \mid aB, A \rightarrow a, B \rightarrow aC, C \rightarrow a\} \cup \\ &\cup \{S \rightarrow aA \mid aB\} \cup \{A \rightarrow aS, C \rightarrow aS\} \end{aligned}$$

da cui possiamo eliminare le produzioni che presentano  $S_3$  nella parte sinistra, ottenendo:

$$P = \underbrace{\{S \rightarrow aA \mid aB\}}_{\lambda}, \underbrace{A \rightarrow aS}_{a}, \underbrace{B \rightarrow aC}_{a}, \underbrace{C \rightarrow aS}_{a}$$

L'automa non deterministico che riconosce  $L(G)$  si costruisce come segue:

$$M = (Q, \delta, q_0, F) \quad \text{con alfabeto di ingresso } X$$

ove, per il relativo algoritmo di trasformazione (Algoritmo 7.1), si ha:

- 1)  $Q = V \cup \{q\} = \{S, A, B, C, q\};$
- 2)  $q_0 = S;$
- 3)  $F = \{q, S\};$
- 4)  $\delta$  è definita dalla seguente tavola di transizione:

$\delta$	$S$	$A$	$B$	$C$	$q$
$a$	$\{A, B\}$	$\{S, q\}$	$\{C\}$	$\{S, q\}$	-

Quindi il grafo degli stati è come in Figura 7.4.

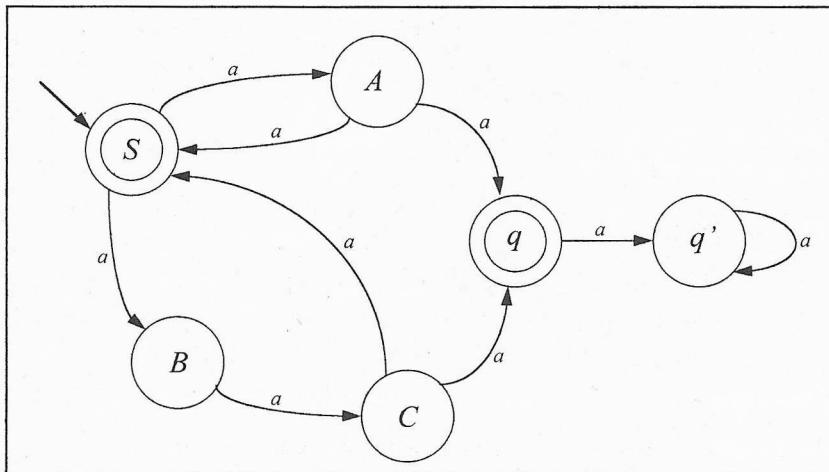


Figura 7.4

- 2) L'automa accettore deterministico  $M'$  equivalente ad  $M$  (ossia tale che  $T(M') = T(M)$ ) si costruisce, secondo l'Algoritmo 6.1, come segue:

$$M' = (Q', \delta', q'_0, F')$$

ove:

- a)  $Q' = 2^Q$ ;
- b)  $q'_0 = \{q_0\}$ ;
- c)  $F' = \{p \subseteq Q \mid p \cap F \neq \emptyset\}$ ;
- d)  $\delta': Q' \times X \rightarrow Q' \quad \exists'$   
 $\forall q' = \{q_1, q_2, \dots, q_i\} \in Q', \forall x \in X:$

$$\begin{aligned}\delta'(q', x) &= \delta'\left(\{q_1, q_2, \dots, q_i\}, x\right) = \\ &= \delta(q_1, x) \cup \delta(q_2, x) \cup \dots \cup \delta(q_i, x) = \bigcup_{j=1}^i \delta(q_j, x)\end{aligned}$$

Dunque si ha:

$$M' = (Q', \delta', q'_0, F')$$

con:

- a)  $Q' = 2^Q = 2^{\{S, A, B, C, q\}}$ ,  $|Q'| = 32$  ;
- b)  $q'_0 = \{S\}$  ;
- c)  $F' = \{\{q\}, \{S\}, \{q, A\}, \{q, B\}, \{q, C\}, \dots, \{S, A\}, \{S, B\}, \{S, C\}, \dots, \{q, S\}\}$ .

Nella pratica, non è necessario considerare tutti gli stati di  $Q'$ , in quanto molti sono irraggiungibili. Per questo motivo, iniziamo a definire  $\delta'$  dal nuovo stato iniziale e proseguiamo definendo  $\delta'$  per un nuovo stato di  $Q'$  non appena esso viene generato:

Per cui si ha:

$$\delta'(\{S\}, a) = \delta(S, a) = \{A, B\}$$

$$\delta'(\{A, B\}, a) = \delta(A, a) \cup \delta(B, a) = \{S, q\} \cup \{C\} = \{S, C, q\}$$

$$\begin{aligned}\delta'(\{S, C, q\}, a) &= \delta(S, a) \cup \delta(C, a) \cup \delta(q, a) = \\ &= \{A, B\} \cup \{S, q\} \cup \emptyset = \{S, A, B, q\}\end{aligned}$$

$$\begin{aligned}\delta'(\{S, A, B, q\}, a) &= \delta(S, a) \cup \delta(A, a) \cup \delta(B, a) \cup \delta(q, a) = \\ &= \{A, B\} \cup \{S, q\} \cup \{C\} \cup \emptyset = \\ &= \{S, A, B, C, q\}\end{aligned}$$

$$\delta'(\{S, A, B, C, q\}, a) = \delta(\{S, A, B, q\}, a) \cup \delta(C, a) = \{S, A, B, C, q\}$$

Il grafo degli stati di  $M'$  è dato in Figura 7.5.

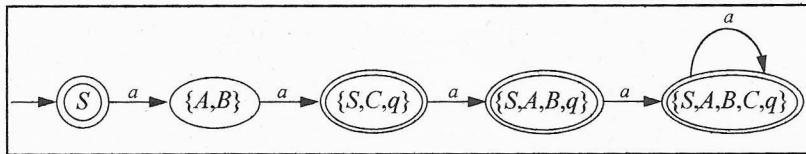


Figura 7.5

Quanto illustrato in precedenza costituisce il modo “standard” di svolgere l’esercizio.

Un modo alternativo è il seguente.

$$((aa + aaa)^*)$$

Osserviamo che:

$$S((aa + aaa)^*) = \{\lambda, aa, aaa, aaaa, aaaaa, \dots\} = \{a\}^* - \{a\}$$

L’automa (minimo e deterministico) che riconosce  $\{a\}^* - \{a\}$  è (Figura 7.6):

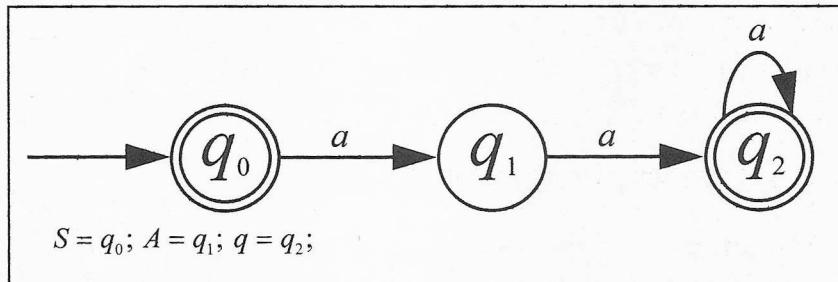


Figura 7.6

corrispondente alla grammatica:

$$S \rightarrow aA \mid \lambda$$

$$A \rightarrow aA \mid a$$

Si osservi che, data la grammatica  $G$ , le cui produzioni sono:

$$S \rightarrow aA \mid \lambda$$

$$A \rightarrow aA \mid a$$

l'automa (non deterministico) che riconosce  $L(G)$  è (Figura 7.7):

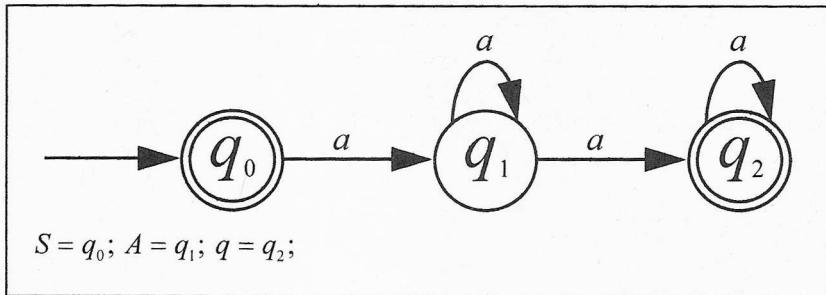


Figura 7.7

che può essere comunque trasformato nell'automa deterministico equivalente in modo molto più semplice:

$$\delta'(\{q_0\}, a) = \delta(q_0, a) = q_1$$

$$\delta'(\{q_1\}, a) = \delta(q_1, a) = \{q_1, q_2\}$$

$$\delta'(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a) = \{q_1, q_2\}$$

per cui l'automa equivalente è (Figura 7.8):

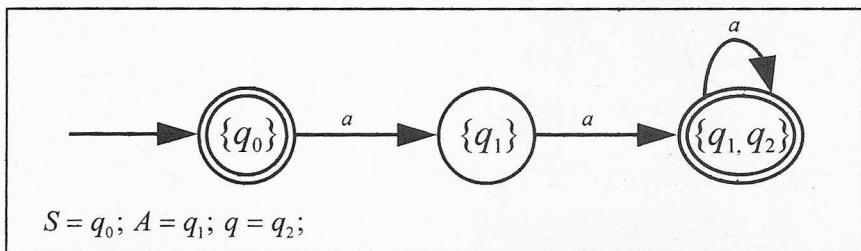


Figura 7.8

Un terzo modo di risolvere l'esercizio:

$$\{a\}^* - \{a\} = \overline{\{a\}}$$

L'automa che riconosce  $L = \{a\}$  è dato in Figura 7.9.

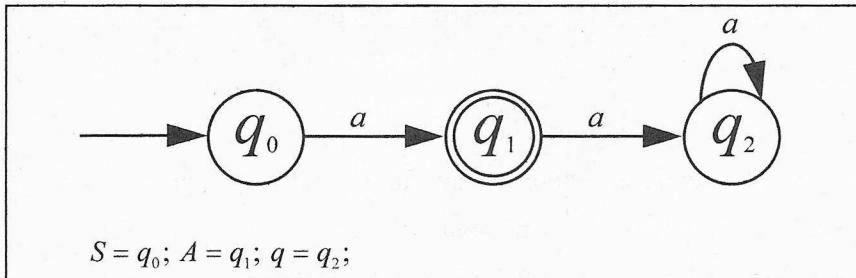


Figura 7.9

L'automa che riconosce  $\overline{L} = \overline{\{a\}}$  si costruisce dall'automa che riconosce  $L$  considerando come stati di accettazione il complementare di  $F$  (Figura 7.10).

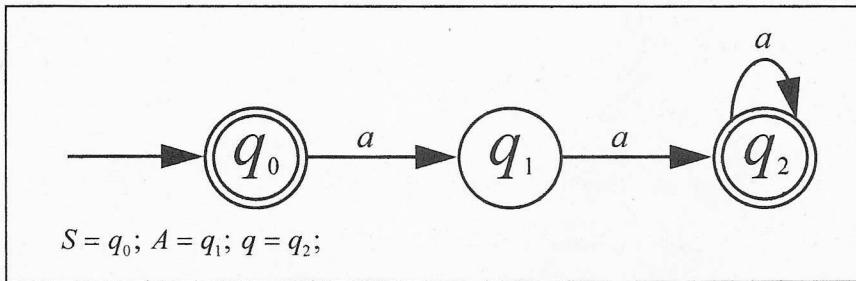


Figura 7.10

Cosa accade se consideriamo l'automa che riconosce  $L = \{a\}$  con funzione di transizione  $\delta$  definita parzialmente?

#### Esercizio 7.4

Con riferimento all'Esercizio 6.1, determinare una grammatica lineare destra che genera il linguaggio:

$$L = \left\{ w \mid w \in \{a, b\}^*, w \text{ ha un numero pari di } a \text{ ed un numero dispari di } b \right\}$$

Una grammatica lineare destra che genera il linguaggio  $L$  può essere costruita a partire dall'automa accettore a stati finiti determinato nell'Esercizio 6.1 utilizzando l'Algoritmo 7.2, come segue:

$$G = (X, V, S, P)$$

dove:

- 1)  $X = \{a, b\}$ ;
- 2)  $V = Q = \{q_0, q_1, q_2, q_3\}$ ;
- 3)  $S = q_0$ ;
- 4)  $P = \{ q_0 \rightarrow aq_2 \mid bq_1 \mid b, q_1 \rightarrow aq_3 \mid bq_0, q_2 \rightarrow aq_0 \mid bq_3, q_3 \rightarrow aq_1 \mid bq_2 \mid a \}$ .

#### Esercizio 7.5

Sia  $L$  il linguaggio denotato dalla seguente espressione regolare:

$$ab(bb)^*c$$

- 1) Trovare un automa a stati finiti che riconosce  $L$ .
- 2) Trasformare l'automa non deterministico al punto 1) in un automa deterministico equivalente.
  
- 1) Determiniamo una grammatica lineare destra che genera  $L$ .

Poiché:

$$\begin{aligned}
 S(ab(bb)^*c) &= S(a) \cdot S(b) \cdot S((bb)^*) \cdot S(c) = \\
 &= \{a\} \cdot \{b\} \cdot \{bb\}^* \cdot \{c\} = \\
 &= \{ab\} \cdot \{bb\}^* \cdot \{c\}
 \end{aligned}$$

possiamo determinare una grammatica che genera  $L$  sfruttando le proprietà di chiusura dei linguaggi di tipo ‘3’.

Una grammatica che genera  $\{ab\}$  è:

$$G_1 = (X, V_1, S_1, P_1)$$

dove:

- $X = \{a, b, c\}$  ;
- $V_1 = \{S_1, A\}$  ;
- $P_1 = \{S_1 \rightarrow aA, A \rightarrow b\}$  .

Una grammatica  $G_3$  che genera  $\{bb\}^*$  si ottiene per iterazione dalla grammatica  $G_2$  che genera  $\{bb\}$ :

$$G_2 = (X, V_2, S_2, P_2)$$

dove:

- $X = \{b\}$  ;
- $V_2 = \{S_2, B\}$  ;
- $P_2 = \{S_2 \rightarrow bB, B \rightarrow b\}$  .

$$G_3 = (X, V_3, S_3, P_3)$$

dove:

- $X = \{b\}$  ;
- $V_3 = V_2 \cup \{S_3\} = \{S_2, B, S_3\}$  ;

- $P_3 = \{S_3 \rightarrow bB | \lambda, B \rightarrow bS_3 | b, S_2 \not\rightarrow bB\}.$ <sup>1</sup>

Una grammatica che genera  $\{c\}$  è:

$$G_4 = (X, V_4, S_4, P_4)$$

dove:

- $X = \{c\};$
- $V_4 = S_4;$
- $P_4 = \{S_4 \rightarrow c\}.$

Una grammatica che genera  $\{ab\} \cdot \{bb\}^*$  è:

$$G_5 = (X, V_5, S_5, P_5)$$

dove:

- $X = \{a, b\};$
- $V_5 = V_1 \cup V_3 = \{S_1, A, S_3, B\};$
- $S_5 = S_1;$
- $P_5 = \{S_1 \rightarrow aA, A \rightarrow bS_3, S_3 \rightarrow bB | \lambda, B \rightarrow bS_3 | b\}.$

Infine, una grammatica che genera il linguaggio  $L = \{ab\} \cdot \{bb\}^* \cdot \{c\}$  è:

$$G = (X, V, S, P)$$

dove:

- $X = \{a, b, c\};$
- $V = V_4 \cup V_5 = \{S_4, S_1, A, S_3, B\};$
- $S = S_1;$
- $P = \{S_1 \rightarrow aA, B \rightarrow bS_3 | bS_4, A \rightarrow bS_3 | bS_4, S_3 \rightarrow bB, S_4 \rightarrow c\}.$

<sup>1</sup> La produzione  $S_2 \rightarrow bB$  viene cancellata in quanto  $S_2$  è un nonterminale inutile.

L'automa che riconosce  $L(G)$  si costruisce come segue, in base all'Algoritmo 7.1:

$$M = (Q, \delta, q_0, F) \text{ con alfabeto di ingresso } X$$

- $Q = V \cup \{q\}$ ,  $q \notin V$ ;
- $q_0 = S_1$ ;
- $F = \{q\}$ ;

e il diagramma di transizione è dato in Figura 7.11.

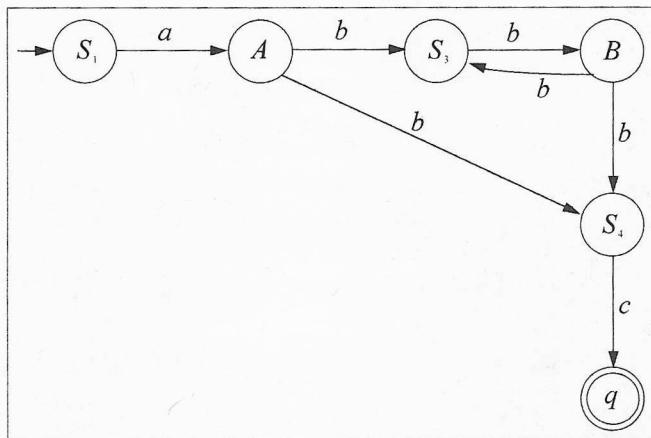


Figura 7.11

2) L'automa deterministico  $M'$  equivalente ad  $M$  si costruisce come segue, in base all'Algoritmo 6.1:

$$M' = (Q', \delta', q'_0, F')$$

dove:

- $Q' = 2^Q$ ;
- $q'_0 = \{S_1\}$ ;
- $F' = \{p \subseteq Q \mid p \cap F \neq \emptyset\}$  con  $|F'| = 32$ ;

- $\delta'$  è definita come segue:

$$\delta' : Q' \times X \rightarrow Q'$$

- ◆  $\delta'(\{S_1\}, a) = \delta(S_1, a) = \{A\};$
- ◆  $\delta'(\{S_1\}, b) = \delta(S_1, b) = \text{non è definita};$
- ◆  $\delta'(\{S_1\}, c) = \delta(S_1, c) = \text{non è definita};$
- ◆  $\delta'(\{A\}, a) = \delta(A, a) = \text{non è definita};$
- ◆  $\delta'(\{A\}, b) = \delta(A, b) = \{S_3, S_4\};$
- ◆  $\delta'(\{A\}, c) = \delta(A, c) = \text{non è definita};$
- ◆  $\delta'(\{S_3, S_4\}, a) = \delta(S_3, a) \cup \delta(S_4, a) = \text{non è definita};$
- ◆  $\delta'(\{S_3, S_4\}, b) = \delta(S_3, b) \cup \delta(S_4, b) = \{B\};$
- ◆  $\delta'(\{S_3, S_4\}, c) = \delta(S_3, c) \cup \delta(S_4, c) = \{q\};$
- ◆  $\delta'(\{B\}, a) = \delta(B, a) = \text{non è definita};$
- ◆  $\delta'(\{B\}, b) = \delta(B, b) = \{S_3, S_4\};$
- ◆  $\delta'(\{B\}, c) = \delta(B, c) = \text{non è definita};$

e il diagramma degli stati è dato in Figura 7.12.

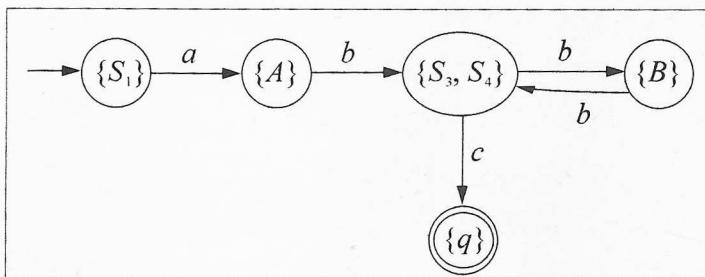


Figura 7.12

### Esercizio 7.6

Si consideri la seguente grammatica lineare destra:

$$G = (X, V, S, P)$$

con

$$X = \{a, b\} \quad V = \{S, B\}$$

$$P = \{S \rightarrow aB, \quad B \rightarrow aB \mid bS \mid a\}$$

Determinare un automa deterministico  $M$  tale che:

$$L(G) = T(M)$$

Facciamo riferimento all'algoritmo per la costruzione dell'automa accettore a stati finiti non deterministico equivalente ad una grammatica lineare destra (Algoritmo 7.1).

$$M = (Q, \delta, q_0, F) \text{ con}$$

- 1)  $X = \{a, b\}$  alfabeto di ingresso;
- 2)  $Q = \{S, B, q\}$ ;
- 3)  $q_0 = S$ ;
- 4)  $F = \{q\}$ ;

$\delta$  è definita come segue:

- 5)  $\delta: Q \times X \rightarrow 2^Q$ 
  - 5.a)  $S \rightarrow aB$  dà origine a:  $B \in \delta(S, a)$ ;  
 $B \rightarrow aB$  dà origine a:  $B \in \delta(B, a)$ ;  
 $B \rightarrow bS$  dà origine a:  $S \in \delta(B, b)$ ;
  - 5.b)  $B \rightarrow a$  dà origine a:  $q \in \delta(B, a)$ .

Per cui, la tavola di transizione che riassume la definizione della funzione  $\delta$  è la seguente:

$\delta$	$S$	$B$	$q$
$a$	$\{B\}$	$\{B, q\}$	-
$b$	-	$\{S\}$	-

Il grafo degli stati di  $M$  è dato in Figura 7.13.

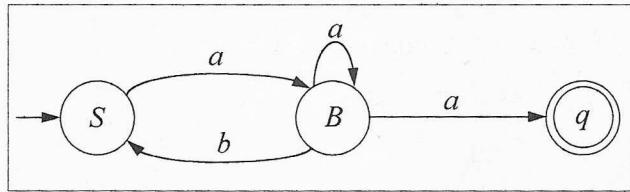


Figura 7.13

L'automa accettore a stati finiti deterministico  $M'$  equivalente ad  $M$  si costruisce come segue (Algoritmo 6.1):

$$M' = (Q', \delta', q'_0, F') \quad X = \{a, b\} \text{ alfabeto di ingresso}$$

con:

- i)  $Q' = 2^Q = 2^{\{S, B, q\}}$ ;
- ii)  $q'_0 = \{S\}$ ;
- iii)  $F' = \{\{q\}, \{q, S\}, \{q, B\}, \{q, S, B\}\}$ ;
- iv)  $\delta'$  è definita come segue:

$$\delta': Q' \times X \rightarrow Q'$$

- $\delta'(\{S\}, a) = \delta(S, a) = \{B\}$ ;
- $\delta'(\{S\}, b) = \delta(S, b) = \text{non definita}$ ;
- $\delta'(\{B\}, a) = \delta(B, a) = \{B, q\}$ ;
- $\delta'(\{B\}, b) = \delta(B, b) = \{S\}$ ;

- $\delta'(\{B, q\}, a) = \delta(B, a) \cup \delta(q, a) = \{B, q\};$
- $\delta'(\{B, q\}, b) = \delta(B, b) \cup \delta(q, b) = \{S\}.$

La tavola di transizione che riassume la definizione della funzione  $\delta'$  è la seguente:

$\delta'$	$\{S\}$	$\{B\}$	$\{B, q\}$
$a$	$\{B\}$	$\{B, q\}$	$\{B, q\}$
$b$	—	$\{S\}$	$\{S\}$

Il grafo degli stati è indicato in Figura 7.14.

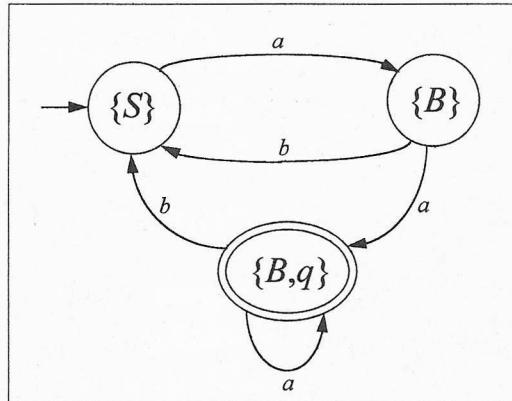


Figura 7.14

### Esercizio 7.7

Con riferimento all'Esercizio 6.2, determinare una grammatica lineare destra che genera il linguaggio:

$$L = \left\{ w \mid w \in \{a, b\}^*, w \neq \alpha aa \beta, \alpha, \beta \in \{a, b\}^* \text{ (} w \text{ non contiene due } a \text{ consecutive)} \right\}$$

Una grammatica lineare destra che genera il linguaggio  $L$  può essere costruita a partire dall'automa accettore a stati finiti determinato nell'Esercizio 6.2, utilizzando l'Algoritmo 7.2, come segue:

$$G = (X, V, S, P)$$

con:

- $X = \{a, b\}$ ;
- $V = Q = \{q_0, q_1, q_2\}$ ;
- $S = q_0$ ;
- $P = \left\{ q \rightarrow xq' \mid q' \in \delta(q, x) \right\} \cup \left\{ q \rightarrow x \mid \delta(q, x) \in F \right\} =$   
 $= \{ q_0 \rightarrow bq_0 \mid aq_1 \mid a \mid b \mid \lambda, q_1 \rightarrow bq_0 \mid aq_2 \mid b, q_2 \rightarrow bq_2 \mid aq_2 \} =$   
 $= \{ q_0 \rightarrow bq_0 \mid aq_1 \mid a \mid b \mid \lambda, q_1 \rightarrow bq_0 \mid b \}.$

### Esercizio 7.8

Si utilizzi il Pumping Lemma per i linguaggi regolari (Teorema 7.2) per dimostrare che i seguenti linguaggi non sono regolari:

- 1)  $L = \{a^k b^k \mid k > 0\}$
- 2)  $L = \{a^n b^m c^k \mid n > k, m, n, k \geq 1\}$

- 1) Supponiamo, per assurdo, che  $L$  sia regolare. Questo implica che  $\exists M = (Q, \delta, q_0, F) : T(M) = L$ .

Supponiamo che  $|Q| = n$  con  $n > 0$ . Consideriamo la seguente parola di  $L$ :

$$a^n b^n$$

$$\overbrace{aaaa...a}^n bbbb...b$$

L'automa  $M$  parte dallo stato  $q_0$  e legge una  $a$  per volta. Dopo aver letto la prima  $a$  si porta in  $q_1$ , dopo la seconda  $a$  in  $q_2$ , ..., dopo la  $n$ -esima  $a$  si porta in  $q_n$ . Abbiamo dunque  $n+1$  stati  $q_0, q_1, \dots, q_n$ , in cui  $M$  transita.

Si ha:

$$q_0 \xrightarrow{a} q_1$$

$$q_1 \xrightarrow{a} q_2$$

.....

$$q_{n-1} \xrightarrow{a} q_n$$

Poiché  $M$  ha solo  $n$  stati, due tra gli stati  $q_0, q_1, \dots, q_n$  devono coincidere.

Siano, ad esempio,  $q_i$  e  $q_j$  gli stati coincidenti:

$$q_i = q_j, \quad i < j$$

Si ha dunque un ciclo nel grafo degli stati di  $M$  (Figura 7.15).

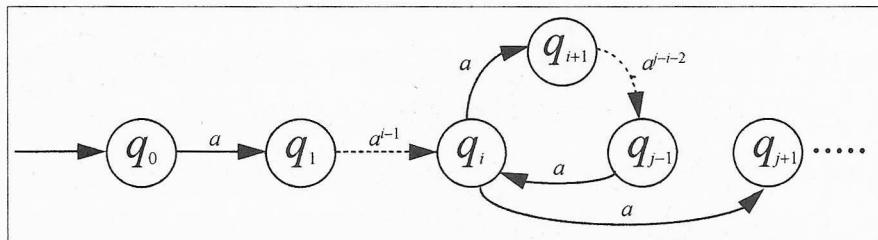


Figura 7.15

Tale ciclo ha lunghezza  $j - i$ .

Poiché esiste tale ciclo, possiamo aggiungere indefinitamente un numero arbitrario di  $a$  nella parola in ingresso, senza modificare l'esito del riconoscimento (vale a dire che la parola viene comunque accettata) se tale numero è un multiplo di  $j - i$ .

Dunque anche  $a^{n+k(j-i)}b^n \in T(M)$ ,  $k = 0, 1, 2, \dots$

Ma ciò è in contraddizione con l'ipotesi  $T(M) = L$ . Dunque  $L$  non è regolare.

2) 1° Modo.

Supponiamo per assurdo che:

$$L = \{a^n b^m c^k \mid n > k, m, n, k \geq 1\}$$

sia regolare.

Allora esiste un automa accettore a stati finiti  $M = (Q, \delta, q_0, F)$  tale che  $L = T(M)$ . Supponiamo  $|Q| = n$  con  $n > 0$ . Consideriamo le sottostringhe:

$$a^2, a^3, \dots, a^n, a^{n+1}, a^{n+2}$$

della stringa  $a^{n+2}bc$ , e gli stati in cui  $M$  si porta quando ha in ingresso una delle suddette sottostringhe.

Si ha:

$$a^2 \rightarrow q_{a^2}$$

$$a^3 \rightarrow q_{a^3}$$

.....

$$a^n \rightarrow q_{a^n}$$

$$a^{n+1} \rightarrow q_{a^{n+1}}$$

$$a^{n+2} \rightarrow q_{a^{n+2}}$$

Se  $q_{a^2}, \dots, q_{a^{n+1}}, q_{a^{n+2}}$  fossero tutti distinti tra loro, avremmo  $n + 1$  stati nell'automa (contro l'ipotesi fatta).

Dunque almeno due stati devono coincidere. Siano  $a^i$  e  $a^j$  le stringhe per cui  $M$  si porta in uno stesso stato e supponiamo  $i < j$ . Si ha dunque:

$$q_{a^i} = q_{a^j} = q$$

Consideriamo ora la parola di  $L$ :  $a^j b c^{j-1}$ . Poiché  $L = T(M)$ , tale parola deve essere accettata da  $M$ . Ma allora la parola  $a^i b c^{j-1}$ ,  $i < j$ , viene accettata da  $M$ .

Ma questo è un assurdo in quanto  $a^i b c^{j-1} \notin L$ ,  $i < j$ .

2° Modo.

Supponiamo, per assurdo, che  $L$  sia regolare. Questo implica che  $\exists M = (Q, \delta, q_0, F) : T(M) = L$ .

Supponiamo che  $|Q| = n$  con  $n > 0$ . Consideriamo la seguente parola di  $L$ :

$$a^{n+1} b c^n$$

$aaaa\dots ab \overbrace{cccc\dots c}^n$

L'automa  $M$  parte dallo stato  $q_0$ , legge le  $n+1$   $a$  e si porta in  $q$ , poi legge l'unica  $b$  e si porta in  $q^1$ . Di seguito, legge la prima  $c$  e si porta in  $q^2$ , legge la seconda  $c$  e si porta in  $q^3$ , ..., legge la  $n$ -esima  $c$  e si porta in  $q^{n+1}$ . Abbiamo dunque  $n+1$  stati  $q^1, q^2, \dots, q^{n+1}$  in cui  $M$  transita. Poiché  $M$  ha solo  $n$  stati, due tra gli stati  $q^1, q^2, \dots, q^{n+1}$  devono coincidere.

Siano  $q^i$  e  $q^j$  tali stati:

$$q^i = q^j = q, \quad i < j$$

Nel grafo degli stati di  $M$  esiste dunque un ciclo di lunghezza  $j-i$ . L'esistenza di tale ciclo nel grafo di  $M$  ci permette di aggiungere altre  $c$  nella parola in ingresso, ottenendo ancora parole accettate da  $M$  se il

numero di  $c$  aggiunte è un multiplo di  $j-i$ . Dunque anche:  
 $a^{n+1}bc^{n+k(j-i)} \in T(M)$ . Ma  $a^{n+1}bc^{n+k(j-i)} \notin L$ ,  $k = 0, 1, 2, \dots$

Da cui l'assurdo. Dunque  $L$  non è regolare.

### Esercizio 7.9

Dimostrare che

$$L = \{a^n b^m c^k \mid m > k, n, m, k \geq 1\};$$

non è un linguaggio regolare.

Supponiamo per assurdo che  $L$  sia un linguaggio regolare.

Allora esiste un automa accettore a stati finiti  $M = (Q, \delta, q_0, F)$  di alfabeto di ingresso  $X = \{a, b, c\}$  tale che  $L = T(M)$ . Supponiamo che  $|Q| = n$  con  $n > 0$ .

Consideriamo le seguenti sottostringhe:

$$ab^2, ab^3, \dots, ab^{n+1}, ab^{n+2}$$

della parola  $ab^{n+2}c^{n+1}$  e gli stati in cui  $M$  si porta quando ha in ingresso una delle suddette sottostringhe:

$$ab^2 \rightarrow q_{ab^2}$$

$$ab^3 \rightarrow q_{ab^3}$$

.....

$$ab^n \rightarrow q_{ab^n}$$

$$ab^{n+1} \rightarrow q_{ab^{n+1}}$$

$$ab^{n+2} \rightarrow q_{ab^{n+2}}$$

Se  $q_{ab^2}, q_{ab^3}, \dots, q_{ab^{n+2}}$  fossero tutti stati distinti di  $M$ , avremmo  $n+1$  stati nell'automa (contro l'ipotesi che  $|Q|=n$ ). Dunque almeno due stati devono coincidere.

Siano  $ab^i$  e  $ab^j$  le stringhe per cui  $M$  si porta in uno stesso stato e supponiamo  $2 \leq i < j \leq n+2$ . Dunque si ha:

$$q_{ab^i} = q_{ab^j} = q$$

Consideriamo ora la stringa:

$$ab^j c^{j-1}$$

Evidentemente:

$$ab^j c^{j-1} \in L$$

Poiché  $L = T(M)$ , tale stringa deve essere accettata da  $M$ .

Ma allora anche la stringa  $ab^i c^{j-1}$ ,  $i < j$ , viene accettata da  $M$  (lo stato in cui  $M$  si porta quando ha in ingresso  $ab^j c^{j-1}$  e  $ab^i c^{j-1}$  è lo stesso).

Siamo dunque giunti ad una contraddizione:

$$ab^i c^{j-1} \in T(M) \text{ e } ab^i c^{j-1} \notin L$$

Ne consegue che  $L$  non è regolare.

## 8. Automi a pila e grammatiche libere.

### 8.1 Automi a pila.

Problema Può un automa a stati finiti (FSA) riconoscere un linguaggio non contestuale?

NO!

#### Esempio 8.1

Si consideri il linguaggio:

$$L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$$

e si supponga che l'automa a stati finiti  $M_1$  che riconosce  $L_1$  abbia  $n$  stati.

Cioè:

$$M_1 = (Q, \delta, q_0, F), |Q| = n$$

per cui si ha:

$$T(M_1) = L_1$$

Si consideri la parola:  $a^n b^n \in L_1$ . Per riconoscere  $a^n b^n$ ,  $M_1$  deve transitare per  $n+1$  stati. Poiché gli stati sono  $n$ , i vertici del cammino non sono tutti distinti, ma si hanno dei cicli (Figura 8.1).

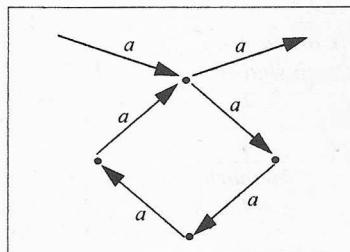


Figura 8.1

Se il ciclo ha lunghezza  $k < n$ , allora la macchina riconosce anche  $a^{n+k}b^n$ , che non è una parola di  $L_1$ .

$M_1$  non ha "stati" a sufficienza per riconoscere  $L_1$ . Gli stati sono l'unico modo di *memorizzare*, di *ricordare il passato* per un automa a stati finiti.

Gli automi a stati finiti hanno memoria finita.

Molti linguaggi, invece, hanno bisogno di memorie potenzialmente infinite.

Gli automi a stati finiti falliscono laddove la procedura di riconoscimento di un linguaggio non stabilisce un limite superiore al numero di stati per il calcolo (è il caso di  $L_1$ , perché non si conosce a priori il numero massimo di  $a$  in ingresso).

Per estendere la memoria di un automa, si possono pensare due soluzioni:

- 1) aumentare il numero dei suoi stati (interni);
- 2) dotare l'automa di una memoria esterna (*memoria ausiliaria*).

La soluzione 1) non risolve il problema di riconoscere linguaggi come  $L_1$ .

La soluzione 2) definisce un nuovo tipo di automa riconoscitore.

Lo schema di un automa riconoscitore, nella sua forma più generale, è illustrato in Figura 8.2.

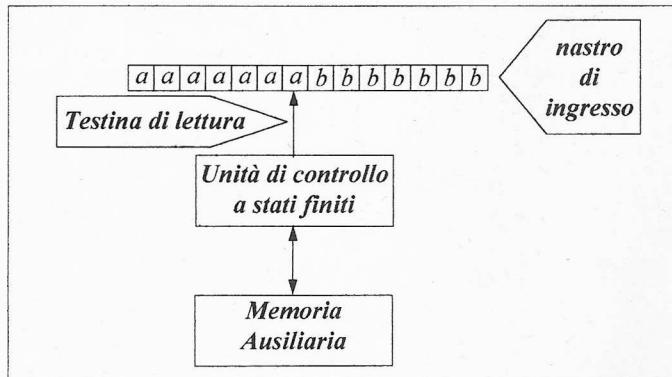


Figura 8.2

Le tre unità che compongono il dispositivo sono (Figura 8.3):

- 1) il *nastro di ingresso*, suddiviso in caselle adiacenti, ciascuna contenente un simbolo dell'alfabeto di ingresso.

L'automa legge il carattere sottostante la *testina di lettura*, che può essere fatta scorrere lungo il nastro in entrambe le direzioni;

- 2) la *memoria ausiliaria* è un'unità, opportunamente organizzata, in generale di capacità illimitata, in grado di accumulare informazioni composte da simboli di un *alfabeto di memoria o di lavoro*.

Tali informazioni, che costituiscono il risultato di un'elaborazione parziale della stringa, vengono successivamente fornite quando richieste.

La memoria è quindi utilizzabile sia in lettura sia in scrittura.

Se la memoria ausiliaria è organizzata a *pila* (*stack*), l'automa prende il nome di *automa a pila* o *automa push-down* (*PDA*);

- 3) l'*unità di controllo a stati finiti* è il cuore dell'automa riconoscitore. In essa è fisicamente *inciso* il programma invariante che costituisce l'algoritmo di riconoscimento.

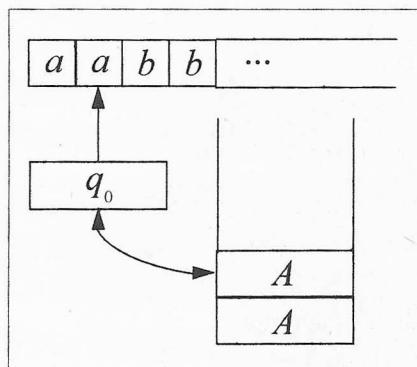


Figura 8.3

Formalmente, possiamo quindi definire un automa a pila non deterministico come segue:

**Definizione 8.1** (Automa a pila non deterministico o automa push-down o PDA)

Un *automa a pila* (PDA) non deterministico è una settupla:

$$M = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q$  è un insieme finito non vuoto di *stati*;
- $X$  è un insieme finito non vuoto di simboli, detto *alfabeto di ingresso*;
- $\Gamma$  è un insieme finito non vuoto di simboli, detto *alfabeto di pila*;
- $\delta$  è una funzione

$$\delta: Q \times (X \cup \{\lambda\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$$

detta *funzione di transizione*.

$\delta$  può anche essere vista come l'insieme delle transizioni  $\langle q, x, Z, q', \sigma \rangle$ , che possiamo anche scrivere nella notazione  $(q', \sigma) \in \delta(q, x, Z)$ ;

- $q_0 \in Q$  è lo *stato iniziale*;
- $Z_0 \in \Gamma$  è il *simbolo inizialmente posto nella pila*;
- $F \subseteq Q$  è l'*insieme degli stati finali*. ■

Il funzionamento di un automa push-down può essere descritto anche attraverso una sequenza di *descrizioni istantanee* (ID).

### Definizione 8.2 (Descrizione istantanea o ID)

Una *descrizione istantanea* (ID) per un automa push-down è una terna:

$$(q, w, \sigma) \in Q \times X^* \times \Gamma^*$$

dove:

- $q \in Q$  è lo stato corrente dell'unità di controllo;
- $w \in X^*$ ,  $w = x_1 x_2 \dots x_n$ , è la porzione di stringa di simboli dell'alfabeto di ingresso da esaminare sul cui primo carattere,  $x_1$ , è posizionata la testina di lettura;
- $\sigma = Z_1 Z_2 \dots Z_m$  è il contenuto della pila con  $Z_1$  al top e  $Z_m$  al bottom. ■

### Definizione 8.3 (Descrizione iniziale)

Una *descrizione*  $(q_0, x, Z_0)$ , con  $x \in X$ , è detta *iniziale*. ■

### Definizione 8.4 (Descrizione finale)

Una *descrizione*  $(q, \lambda, \sigma)$ , con  $q \in F$ ,  $\sigma \in \Gamma^*$ , è detta *finale*. ■

Le *transizioni* fanno corrispondere (in maniera non deterministica) descrizioni istantanee a descrizioni istantanee in due modi:

se  $M$  è nello stato  $q$ , il simbolo di input corrente è  $x$  ed il simbolo al top dello stack è  $A$ , allora:

1) se  $\langle q, x, A, q', A_1 \dots A_k \rangle$ , per  $x \in X$ , è una quintupla dell'automa a pila, allora  $M$  può (non deterministicamente) causare la seguente transizione:

$$(q, xw, A\sigma) \Rightarrow (q', w, A_1 \dots A_k \sigma)$$

che può anche essere scritta nella forma:

$$\delta(q, x, A) = (q', A_1 \dots A_k)$$

2) se  $\langle q, \lambda, A, q', A_1 \dots A_k \rangle$  è una quintupla dell'automa a pila, allora la sua esecuzione può provocare la seguente transizione:

$$(q, xw, A\sigma) \xrightarrow{*} (q', xw, A_1 \dots A_k \sigma)$$

che può anche essere scritta nella forma:

$$\delta(q, \lambda, A) = (q', A_1 \dots A_k)$$

Il caso 2) denota una transizione non deterministica che avviene senza leggere simboli in ingresso. Dunque, al termine, la testina di lettura non viene fatta avanzare.

La notazione  $\xrightarrow{*}$  denota la chiusura riflessiva e transitiva *dell'operatore di transizione*, ossia tale simbolo indica una successione di transizioni che porta  $M$  da una data descrizione istantanea ad un'altra.

La notazione  $\langle terna1 \rangle \xrightarrow{*} \langle terna2 \rangle$  significa che la descrizione istantanea  $\langle terna2 \rangle$  può essere derivata dopo una sequenza finita di zero o più transizioni da  $\langle terna1 \rangle$ .

## 8.2 Linguaggi accettati da automi a pila.

**Definizione 8.5** (Stringa accettata in condizione di pila vuota)

Dato un PDA  $M$ , diremo che  $w \in X^*$  è una *stringa accettata da  $M$  in condizione di pila vuota* se:

$$(q_0, w, Z_0) \xrightarrow{*} (q, \lambda, \lambda) \quad q \in Q$$

**Definizione 8.6** (Linguaggio accettato in condizione di pila vuota)

Dato un PDA  $M$ , diremo che:

$$T(M) = \left\{ w \mid (q_0, w, Z_0) \xrightarrow{*} (q, \lambda, \lambda), q \in Q \right\}$$

è il *linguaggio accettato da  $M$  nella condizione di pila vuota*. ■

**Osservazione 8.1**

I valori di  $X$  o di  $\Gamma$  per cui non è definita la funzione  $\delta$  in uno stato  $q$  corrispondono a stringhe respinte dall'automa.

In alternativa alle definizioni precedenti di stringa accettata e di linguaggio accettato da un automa a pila nella condizione di pila vuota, si possono dare le seguenti definizioni:

**Definizione 8.7** (Stringa accettata in condizione di stato finale)

Dato un PDA  $M$ , diremo che  $w \in X^*$  è una *stringa accettata da  $M$  in condizione di stato finale* se:

$$(q_0, w, Z_0) \xrightarrow{*} (q_a, \lambda, \sigma), q_a \in F, \sigma \in \Gamma^*$$
 ■

**Definizione 8.8** (Linguaggio accettato in condizione di stato finale)

Dato un PDA  $M$ , diremo che:

$$T(M) = \left\{ w \mid (q_0, w, Z_0) \xrightarrow{*} (q_a, \lambda, \sigma), q_a \in F, \sigma \in \Gamma^* \right\}$$

è il *linguaggio accettato da  $M$  nella condizione di stato finale*. ■

### Teorema 8.1 (senza dimostrazione)

La classe dei linguaggi riconosciuti da un automa a pila nella condizione di stato finale coincide con quella riconosciuta nella condizione di pila vuota.

### 8.3 Esempi di linguaggi riconosciuti da automi a pila.

#### Esempio 8.2

Sia dato il seguente linguaggio  $L_2$ :

$$L_2 = \{w \mid w \text{ ha lo stesso numero di } a \text{ e di } b\}$$

Una grammatica che genera  $L_2$  è:

$$G_2 = (X, V, S, P_2)$$

dove:

$$X = \{a, b\}, \quad V = \{S\},$$

$$P_2 = \{S \rightarrow ab \mid ba \mid SS \mid aSb \mid bSa\},$$

che rispecchia la seguente definizione ricorsiva di  $L_2$ :

$$\begin{cases} ab, ba \in L_2 \\ \text{se } v, w \in L_2 \text{ allora: } vw \in L_2 \\ awb, bwa \in L_2 \end{cases}$$

Costruiamo l'automa a pila  $M_2$  che riconosce  $L_2$  nella condizione di pila vuota:

$$M_2 = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0\}$
- $X = \{a, b\}$
- $\Gamma = \{Z_0, A, B\}$   $Z_0$  indicatore del fondo dello stack

- $F = \emptyset$  (si intende accettare il linguaggio nella condizione di pila vuota)

La tavola di transizione di  $M_2$  è la seguente:

CARATTERE DI PILA	STATO	CARATTERE DI INGRESSO	
		<i>a</i>	<i>b</i>
<b><i>A</i></b>	$q_0$	<b><i>AA</i></b> (3)	$\lambda$ (5)
<b><i>B</i></b>	$q_0$	$\lambda$ (6)	<b><i>BB</i></b> (4)
<b><i>Z</i><sub>0</sub></b>	$q_0$	<b><i>AZ</i><sub>0</sub></b> (1)	<b><i>BZ</i><sub>0</sub></b> (2)

corrispondente al seguente algoritmo.

Inizialmente lo stack è vuoto; la stringa  $w$  viene esaminata da sinistra a destra e si effettuano le seguenti operazioni:

- (1) se lo stack è vuoto ed il simbolo corrente di  $w$  è *a*, pon *A* sullo stack;
- (2) se lo stack è vuoto ed il simbolo corrente di  $w$  è *b*, pon *B* sullo stack;
- (3) se il simbolo al top dello stack è *A* e il simbolo corrente di  $w$  è *a*, pon un'altra *A* sullo stack;
- (4) se il simbolo al top dello stack è *B* e il simbolo corrente di  $w$  è *b*, pon un'altra *B* sullo stack;
- (5) se il simbolo al top dello stack è *A* e il simbolo corrente di  $w$  è *b*, estrai un elemento dallo stack;
- (6) se il simbolo al top dello stack è *B* e il simbolo corrente di  $w$  è *a*, estrai un elemento dallo stack.

Si osservi che l'unità di controllo di  $M_2$  ha un unico stato, dunque nella casella della tavola di transizione non è necessario indicare lo stato.

Quindi:

$$T(M_2) = \left\{ w \mid (q_0, w, Z_0) \xrightarrow{*} (q, \lambda, \lambda), q \in Q \right\} = L_2$$

L'algoritmo visto sopra corrisponde al seguente *programma per PDA*:

- (1)  $\langle a, Z_0, AZ_0 \rangle;$
- (2)  $\langle b, Z_0, BZ_0 \rangle;$
- (3)  $\langle a, A, AA \rangle;$
- (4)  $\langle b, B, BB \rangle;$
- (5)  $\langle a, B, \lambda \rangle;$
- (6)  $\langle b, A, \lambda \rangle;$
- (7)  $\langle \lambda, Z_0, \lambda \rangle; \rightarrow$  condizione di pile vuota

in cui abbiamo introdotto la  $\lambda$ -regola (7) che ci consente di cancellare il marcitore del fondo dello stack  $Z_0$  senza leggere alcun carattere di ingresso.

### Esempio 8.3

Sia dato il seguente linguaggio  $L_3$ :

$$L_3 = \left\{ wcw^R \mid w \in \{a, b\}^* \right\}$$

Una grammatica che genera  $L_3$  è:

$$G_3 = (X, V, S, P_3)$$

dove:

$$X = \{a, b, c\}, \quad V = \{S\},$$

$$P_3 = \{S \rightarrow c \mid aSa \mid bSb\},$$

che rispecchia la seguente definizione ricorsiva di  $L_3$ :

$$\begin{cases} c \in L_3 \\ \text{se } w \in L_3 \text{ allora } awa, bwb \in L_3 \end{cases}$$

Costruiamo l'automa a pila  $M_3$  che riconosce  $L_3$  nella condizione di pila vuota:

$$M_3 = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0, q_1\}$        $q_0 = \text{read}, \quad q_1 = \text{match}$
- $X = \{a, b, c\}$
- $\Gamma = \{Z_0, A, B\}$
- $F = \emptyset$  (si intende accettare il linguaggio nella condizione di pila vuota)

La tavola di transizione di  $M_3$  è la seguente:

CARATTERE DI PILA	STATO	CARATTERE DI INGRESSO		
		$a$	$b$	$c$
$A$	$q_0$	$(q_0, AA)$ (2)	$(q_0, BA)$ (5)	$(q_1, A)$ (8)
	$q_1$	$(q_1, \lambda)$ (10)		
$B$	$q_0$	$(q_0, AB)$ (3)	$(q_0, BB)$ (6)	$(q_1, B)$ (9)
	$q_1$		$(q_1, \lambda)$ (11)	
$Z_0$	$q_0$	$(q_0, AZ_0)$ (1)	$(q_0, BZ_0)$ (4)	$(q_1, Z_0)$ (7)
	$q_1$			

corrispondente al seguente programma per PDA:

- (1)  $\langle q_0, a, Z_0, q_0, AZ_0 \rangle;$
- (2)  $\langle q_0, a, A, q_0, AA \rangle;$
- (3)  $\langle q_0, a, B, q_0, AB \rangle;$
- (4)  $\langle q_0, b, Z_0, q_0, BZ_0 \rangle;$
- (5)  $\langle q_0, b, A, q_0, BA \rangle;$
- (6)  $\langle q_0, b, B, q_0, BB \rangle;$
- (7)  $\langle q_0, c, Z_0, q_1, Z_0 \rangle;$
- (8)  $\langle q_0, c, A, q_1, A \rangle;$
- (9)  $\langle q_0, c, B, q_1, B \rangle;$
- (10)  $\langle q_1, a, A, q_1, \lambda \rangle;$
- (11)  $\langle q_1, b, B, q_1, \lambda \rangle;$
- (12)  $\langle q_1, \lambda, Z_0, q_1, \lambda \rangle; (\lambda\text{-regola})$

dove:

- (1), (2) e (3) si possono riassumere con  $\langle q_0, a, Z, q_0, AZ \rangle$  con  $Z = Z_0, A, B$
- (4), (5) e (6) si possono riassumere con  $\langle q_0, b, Z, q_0, BZ \rangle$  con  $Z = Z_0, A, B$
- (7), (8) e (9) si possono riassumere con  $\langle q_0, c, Z, q_1, Z \rangle$  con  $Z = Z_0, A, B$

Esempio di *trace*:

TRACE	
STRINGA DA ESAMINARE	STACK
<i>aabcbaa</i>	$Z_0$
<i>abcbaa</i>	$AZ_0$
<i>bcbaa</i>	$AAZ_0$
<i>cbaa</i>	$BAAZ_0$
<i>baa</i>	$BAAZ_0$
<i>aa</i>	$AAZ_0$
<i>a</i>	$AZ_0$
	$Z_0$

Si osservi che:

- $M_3$  nello stato  $q_1 = \text{match}$  si ferma se:
  - legge una *a* con *B* al top dello stack;
  - oppure:
  - legge una *b* con *A* al top dello stack;
  - ...
- $L_3$  è semplice da riconoscere per un PDA perché il centro *c* dei palindromi segnala al PDA quando cambiare stato ed iniziare a svuotare lo stack.

#### Esempio 8.4

Sia dato il seguente linguaggio  $L_4$ :

$$L_4 = \left\{ ww^R \mid w \in \{a, b\}^* \right\}$$

Una grammatica che genera  $L_4$  è:

$$G_4 = (X, V, S, P_4)$$

dove:

$$X = \{a, b\}, \quad V = \{S\},$$

$$P_4 = \{S \rightarrow aa \mid bb \mid aSa \mid bSb \mid \lambda\},$$

corrispondente alla seguente definizione ricorsiva di  $L_4$ :

$$\begin{cases} aa, bb \in L_4 \\ \text{se } w \in L_4 \text{ allora } awa, bwb \in L_4 \end{cases}$$

### Osservazione

La sostanziale differenza tra  $L_3$  ed  $L_4$  è la mancanza del carattere  $c$  che in  $L_3$  fungeva da separatore fra la stringa  $w$  e la speculare  $w^R$ . L'automa riconoscitore, in questo caso, non è in grado di capire se il carattere letto appartiene a  $w$  o a  $w^R$ ; quindi deve operare come se fosse contemporaneamente nella fase di accumulo di informazioni nella pila oppure di scarico della pila.

Questo implica che solo un automa a pila non deterministico può riconoscere  $L_4$ .

Costruiamo l'automa a pila  $M_4$  che riconosce  $L_4$  nella condizione di pila vuota:

$$M_4 = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0, q_1\}$

- $X = \{a, b\}$
- $\Gamma = \{Z_0, A, B\}$
- $F = \emptyset$

La tavola di transizione di  $M_4$  è la seguente:

CARATTERE DI PILA	STATO	CARATTERE DI INGRESSO	
		$a$	$b$
<b>A</b>	$q_0$	$\{(q_0, AA), (q_1, \lambda)\}$	$(q_0, BA)$
	$q_1$	$(q_1, \lambda)$	
<b>B</b>	$q_0$	$(q_0, AB)$	$\{(q_0, BB), (q_1, \lambda)\}$
	$q_1$		$(q_1, \lambda)$
<b>Z<sub>0</sub></b>	$q_0$	$(q_0, AZ_0)$	$(q_0, BZ_0)$
	$q_1$		

corrispondente al seguente programma per PDA:

- (1)  $< q_0, a, Z, q_0, AZ >$ ;  $Z = Z_0, A, B$
- (2)  $< q_0, b, Z, q_0, BZ >$ ;
- (3)  $< q_0, \lambda, Z, q_1, Z >$  (sostituisce la  $\lambda$ -regola dell'esempio precedente);
- (4)  $< q_1, a, A, q_1, \lambda >$ ;
- (5)  $< q_1, b, B, q_1, \lambda >$ ;
- (6)  $< q_1, \lambda, Z_0, q_1, \lambda >$ .

L'automa è non deterministico perché, ognqualvolta è nello stato  $q_0 = \text{read}$ , ha due possibili transizioni:

- leggere un simbolo di input;
- commutare nello stato  $q_1 = \text{match}$ .

### Osservazione

$L_3$  è accettato da un PDA deterministico.

$L_4$  è accettato da un PDA non deterministico,

e inoltre  $\exists M'_4, M'_4$  PDA deterministico tale che:

$$T(M'_4) = L_4$$

Dunque *la classe dei linguaggi accettati da automi a pila non deterministici contiene propriamente la classe dei linguaggi accettati da automi a pila deterministici*.

## 8.4. Forme normali per grammatiche non contestuali.

### Esempio 8.5

Sia dato il seguente automa a stati finiti non deterministico (Figura 8.4):

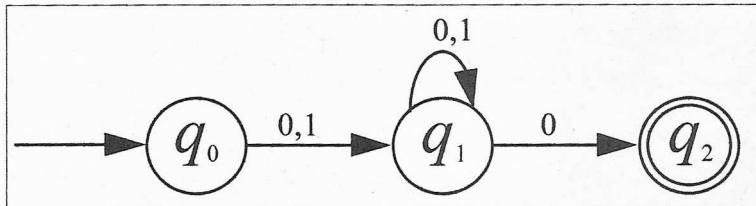


Figura 8.4

che riconosce il linguaggio:

$$L = \left\{ w0 \mid w \in \{0,1\}^+ \right\} = S((0+1)^+ 0)$$

Una grammatica che genera  $L$  è:

$$G = (X, V, S, P)$$

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

$$V = \{S, A\}$$

$$P = \left\{ S \rightarrow 0A \mid 1A, A \rightarrow 0A \mid 1A \mid 0 \right\}$$

Consideriamo la stringa  $w = 0110$ .

Esaminiamo come può essere riconosciuta; in particolare cerchiamo di identificare come, partendo dall'assioma  $S$ , può essere derivata  $w$  per successive applicazioni delle regole sintattiche.

La lettura del primo carattere, 0, permette di ricostruire univocamente la produzione che è stata usata per riscrivere  $S$ , cioè  $S \rightarrow 0A$ .

I successivi due caratteri 1 identificano entrambi la regola  $A \rightarrow 1A$ , mentre la lettura dell'ultimo carattere, 0, implica una scelta non deterministica tra la produzione  $A \rightarrow 0A$  e  $A \rightarrow 0$ .

Ovviamente, qualora venga effettuata la prima scelta, sarà necessario successivamente, a causa del suo fallimento, considerare la seconda, che determina un esito positivo nell'analisi della stringa.

Quanto è stato qui esposto in modo informale può essere espresso in modo più rigoroso, considerando il seguente automa a pila:

$$M = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0\}$
- $X = \{0,1\}$
- $\Gamma = \{S, A\}$
- $Z_0 = S$

- $F = \emptyset$
- $\delta$  è data da:
  - ◆  $\delta(q_0, 0, S) = \{(q_0, A)\};$
  - ◆  $\delta(q_0, 1, S) = \{(q_0, A)\};$
  - ◆  $\delta(q_0, 0, A) = \{(q_0, A), (q_0, \lambda)\};$
  - ◆  $\delta(q_0, 1, A) = \{(q_0, A)\}.$

### Osservazione 8.2

La pila dell'automa  $M$  contiene sempre un solo carattere al massimo. Ciò è dovuto al fatto che  $G$  è di tipo ‘3’.

Le operazioni che ad ogni istante vengono effettuate dall'automa  $M$  sono basate su una serie di *predizioni*; ad esempio, leggendo 0 sul nastro di ingresso, con  $S$  in cima alla pila, l'automa predice che il resto della stringa dovrà derivare da  $A$ .

### Esempio 8.6

Esaminiamo la seguente grammatica non contestuale:

$$G = (X, V, S, P)$$

$$X = \{0, 1, 2\} \quad V = \{S, A, B\}$$

$$P = \{S \rightarrow 0SAB \mid 1, A \rightarrow 1A \mid 1, B \rightarrow 2B \mid 2\}$$

Basandosi sulle considerazioni precedenti, anche in questo caso è possibile costruire un automa che riconosca il linguaggio generato da  $G$  in modo predittivo. Per esempio, la lettura di 0 quando in cima alla pila c'è  $S$  fa sì che il resto della stringa debba derivare da  $SAB$ .

Esaminando quindi il primo carattere di tale sequenza e leggendo ancora uno 0 sul nastro di ingresso, si identifica nuovamente la produzione  $S \rightarrow 0SAB$ , quindi il resto della stringa deve derivare da  $SABAB$ .

I risultati delle successive predizioni vengono messi in una memoria a pila, che in questo esempio può effettivamente contenere un numero illimitato di simboli.

In modo formale, il riconoscimento di  $L(G)$  può essere effettuato dal seguente automa a pila:

$$M = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0\}$
- $X = \{0, 1, 2\}$
- $\Gamma = \{S, A, B\}$
- $Z_0 = S$
- $F = \emptyset$  (linguaggio accettato nella condizione di pila vuota)
- $\delta$  è data da:
  - ◆  $\delta(q_0, 0, S) = \{(q_0, SAB)\}$
  - ◆  $\delta(q_0, 1, S) = \{(q_0, \lambda)\}$
  - ◆  $\delta(q_0, 1, A) = \{(q_0, A), (q_0, \lambda)\}$
  - ◆  $\delta(q_0, 2, B) = \{(q_0, B), (q_0, \lambda)\}$

Gli esempi precedenti trovano una giustificazione nel seguente:

### Teorema 8.2 (senza dimostrazione)

Sia  $G = (X, V, S, P)$  una grammatica non contestuale le cui produzioni sono del tipo  $A \rightarrow a\alpha$ ,  $\alpha \in V^*$ ,  $a \in X$ .

$L(G)$  è riconosciuto dall'automa a pila:

$$M = (Q, X, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- $Q = \{q_0\}$
- $\Gamma = V$
- $Z_0 = S$
- $F = \emptyset$  (linguaggio accettato nella condizione di pila vuota)
- $\forall A \rightarrow a\alpha \in P : (q_0, \alpha) \in \delta(q_0, a, A)$

### Definizione 8.9 (Forma normale di Chomsky)

Si dice che una grammatica non contestuale  $G = (X, V, S, P)$  è in *forma normale di Chomsky* se ogni produzione è in una delle seguenti forme:

- i)  $S \rightarrow \lambda$ ;
- ii)  $A \rightarrow BC$ , dove  $A, B, C \in V$ ; se  $S \rightarrow \lambda \in P$  allora  $B, C \in V - \{S\}$ ;
- iii)  $A \rightarrow a$ , dove  $A \in V$ ,  $a \in X$ .

■

### Definizione 8.10 (Forma normale priva di ricorsioni sinistre)

Si dice che una grammatica non contestuale  $G = (X, V, S, P)$  è in *forma normale priva di ricorsioni sinistre* (*NLR* = No Left Recursion) se non ha produzioni della forma:

$$A \rightarrow Av$$

dove  $A \in V$  e  $v \in (X \cup V)^*$  (dette *produzioni ricorsive a sinistra*). ■

### Definizione 8.11 (Forma normale di Greibach)

Si dice che una grammatica non contestuale  $G = (X, V, S, P)$  è in *forma normale di Greibach* (*GNF = Greibach Normal Form*) se ogni produzione è del tipo:

- i)  $S \rightarrow \lambda;$
- ii)  $A \rightarrow a\alpha$ , dove  $a \in X$  e  $\alpha \in V^*$ .

■

### Teorema 8.3

Sia  $G = (X, V, S, P)$  un'arbitraria grammatica non contestuale.

Esistono allora tre grammatiche  $G'$ ,  $G''$  e  $G'''$ , equivalenti a  $G$ , ossia tali che  $L(G) = L(G^i)$ , con  $i = ', '' ,'''$ ,

con       $G'$  in forma normale di Chomsky,  
 $G''$  in forma normale di Greibach,  
 $G'''$  in forma normale priva di ricorsioni sinistre.

### Dimostrazione 8.3

La dimostrazione di questo teorema è costruttiva.

L'algoritmo per la trasformazione di una grammatica non contestuale  $G$  in una grammatica equivalente  $G'$  in forma normale di Chomsky è illustrato nell'Esercizio 8.1, mentre quello per la trasformazione di  $G$  in una grammatica equivalente  $G''$  in forma normale di Greibach è dato nell'Esercizio 8.2. Un passo di questo algoritmo effettua la trasformazione di  $G$  in una grammatica equivalente  $G'''$  in forma normale priva di ricorsioni sinistre.

c.v.d.

### Esercizio 8.1

Convertire la seguente grammatica in forma normale di Chomsky:

$$G = (X, V, S, P)$$

$$X = \{0, 1, 2\} \quad V = \{S, A, B\}$$

$$P = \{S \rightarrow 00A \mid B \mid 1, A \rightarrow 1AA \mid 2, B \rightarrow 0\}$$

$G$  non presenta  $\lambda$ -regole. Se così non fosse, occorrerebbe applicare preventivamente il Passo 1 dell'Algoritmo 8.2.

Algoritmo 8.1 (Trasformazione in forma normale di Chomsky).

Passo 1 *Conversione dei terminali che compaiono nella parte destra di qualche produzione in nonterminali e aggiunta delle produzioni appropriate.*

$$S \rightarrow BBA \mid B \mid 1$$

$$A \rightarrow CAA \mid 2$$

$$C \rightarrow 1$$

$$B \rightarrow 0$$

Passo 2 *Suddivisione delle produzioni le cui parti destre hanno più di due nonterminali.*

$$S \rightarrow BD \mid B \mid 1$$

$$D \rightarrow BA$$

$$A \rightarrow CE \mid 2$$

$$E \rightarrow AA$$

$$C \rightarrow 1$$

$$B \rightarrow 0$$

Passo 3 *Sostituzione dei nonterminali che costituiscono, da soli, le parti destre di qualche produzione.*

$$S \rightarrow BD \mid 0 \mid 1$$

$$D \rightarrow BA$$

$$A \rightarrow CE \mid 2$$

$$E \rightarrow AA$$

$$C \rightarrow 1$$

$$B \rightarrow 0$$

### Esercizio 8.2

Convertire la seguente grammatica in forma normale di Greibach:

$$G = (X, V, S, P)$$

$$X = \{a, b, c, d\} \quad V = \{S, A, B, C, D\}$$

$$P = \left\{ S \rightarrow AaB, B \rightarrow CD \mid c, D \rightarrow ab \mid a, C \rightarrow BC \mid d, A \rightarrow \lambda \right\}$$

Algoritmo 8.2 (Trasformazione in forma normale di Greibach).

Passo 1 *Eliminazione delle  $\lambda$ -regole* (del tipo  $A \rightarrow \lambda$ ).

Dividiamo  $V$  in due sottoinsiemi  $V_1$  e  $V_2$ :

$$V_1 = \left\{ A \in V \mid A \stackrel{*}{\Rightarrow} \lambda \right\}, \quad V_2 = V - V_1$$

Applichiamo il seguente algoritmo (Algoritmo 8.2.1) che, per ogni nonterminale  $A$  di una grammatica  $G$ , calcola l'attributo  $e(A)$ , che è vero se e solo se  $A \stackrel{*}{\Rightarrow} \lambda$ , cioè:

$$A \in V: \quad e(A) \stackrel{\text{def}}{=} \text{true} \Leftrightarrow A \stackrel{*}{\Rightarrow} \lambda$$

**Algoritmo 8.2.1** (Calcolo dei nonterminali che generano  $\lambda$ ).

- 1)  $\forall A \in V: e(A) \leftarrow \text{false};$
- 2) se  $A \rightarrow \lambda \in P, e(A) \leftarrow \text{true};$   
e si marcano (con un apice) tutte le occorrenze di  $A$  che appaiono nelle parti destre delle produzioni di  $G$ ;
- 3)  $\forall A \rightarrow \alpha \in P$ , si cancellano da  $\alpha$  i nonterminali marcati; se la parte destra diventa vuota  $e(A) \leftarrow \text{true}$ , e si marcano (con un apice) tutte le occorrenze di  $A$  che appaiono nelle parti destre delle produzioni di  $G$ ;
- 4) se nel passo precedente è mutato qualche  $e(A)$  si riapplica 3), altrimenti l'algoritmo termina.

Nel caso della grammatica dell'Esercizio 8.2, si ha:

$e( )$	1)	2)	3)	4)	
$S$	0	0	0	0	
$A$	0	1	1	1	$0 = \text{false}$
$B$	0	0	0	0	$1 = \text{true}$
$C$	0	0	0	0	
$D$	0	0	0	0	

L'unico nonterminale per cui  $e( )$  è vero è proprio  $A$ . Per cui:

$$V_1 = \{A\}$$

Si trasforma la grammatica  $G = (X, V, S, P)$  contenente  $\lambda$ -regole in una grammatica  $G' = (X, V, S, P')$  (secondo il Lemma della Stringa Vuota - Lemma 5.2), attraverso le seguenti regole, che costituiscono la dimostrazione costruttiva del Lemma 5.2:

- i)  $S \rightarrow \lambda \in P' \Leftrightarrow \underset{G}{\overset{*}{S}} \Rightarrow \lambda \quad (\lambda \in L(G));$
- ii) Se  $A \rightarrow X_1 X_2 \dots X_r \in P$ ,  $r \geq 1$ , allora in  $P'$  si trovano tutte le produzioni del tipo:

$$A \rightarrow Y_1 Y_2 \dots Y_r$$

ove:

se  $X_i \in X \cup V_2$ , allora  $Y_i = X_i$ ;

altrimenti (se  $X_i \in V_1$ ), si pone  $Y_i = X_i$  oppure  $Y_i = \lambda$ , eliminando però il caso  $Y_1 Y_2 \dots Y_r = \lambda$ .

La grammatica  $G'$  è data da:

$$P' = \left\{ S \rightarrow aB \mid AaB, B \rightarrow CD \mid c, D \rightarrow ab \mid a, C \rightarrow BC \mid d \right\}$$

### Passo 2 Eliminazione dei nonterminali inutili.

Un'operazione di potatura alla quale si assoggetta normalmente una grammatica consiste nella eliminazione di eventuali *nonterminali inutili* in essa presenti.

#### Definizione 8.12 (Nonterminale inutile)

Un *nonterminale A* è *inutile* se:

- $A$  non genera alcuna stringa terminale;
- da  $S$  non deriva alcuna forma di frase contenente  $A$ . ■

L'algoritmo seguente calcola, per ogni nonterminale  $A$ , gli attributi  $t(A)$  e  $s(A)$ , così definiti:

$$\begin{aligned} A \in V: \quad t(A) &= \text{true} \stackrel{\text{def}}{\Leftrightarrow} A \xrightarrow{*} w, w \in X^*; \\ s(A) &= \text{true} \stackrel{\text{def}}{\Leftrightarrow} S \xrightarrow{*} \alpha A \beta, \alpha, \beta \in (X \cup V)^* \end{aligned}$$

**Algoritmo 8.2.2** (Eliminazione dei nonterminali inutili).

- 1)  $\forall A \in V: t(A) \leftarrow s(A) \leftarrow \text{false};$
- 2) se  $A \rightarrow x \in P, x \in X^*, t(A) \leftarrow \text{true},$   
e si marcano (con un apice ‘*t*’) tutte le occorrenze di  $A$  che appaiono nelle parti destre delle produzioni di  $G$ ;
- 3)  $s(S) \leftarrow \text{true}$ , e si marcano con apice ‘*s*’ tutte le occorrenze di  $S$  che appaiono come parte sinistra in  $P$ ;
- 4) si effettuano le procedure seguenti 4.1) e 4.2):
  - 4.1)  $\forall A \rightarrow \alpha \in P$ , se tutti i nonterminali di  $\alpha$  sono marcati con ‘*t*’,  
 $t(A) \leftarrow \text{true}$ , e si marcano con apice ‘*t*’ tutte le occorrenze di  $A$  che appaiono nelle parti destre delle produzioni di  $G$ ;
  - 4.2) se  $A$  è marcato con ‘*s*’, si pone  $s(B) \leftarrow \text{true} \quad \forall A \rightarrow \alpha B \beta$ , e si marcano con apice ‘*s*’ tutte le occorrenze di  $B$  che appaiono come parte sinistra in  $P$ ;
- 5) se nel passo precedente è mutato qualche  $t(A)$  o  $s(A)$  si riapplica 4), altrimenti
- 6) per ogni  $A$  per cui  $t(A) = \text{false}$  oppure  $s(A) = \text{false}$ , si cancellano da  $G$  tutte le produzioni in cui compare  $A$  (si cancellano gli  $A$  per cui  $\neg s(A) \vee \neg t(A)$ ).

Nella grammatica data si ha:

	1)		2)		3)		4.1)		4.2)		5)		5)	
	$t()$	$s()$												
$S$	0	0	0	1	1	1	1	1	1	1	1	1	1	1
$A$	0	0	0	0	0	1	0	1	0	1	0	1	0	1
$B$	0	0	1	0	1	1	1	1	1	1	1	1	1	1
$C$	0	0	1	0	1	0	1	1	1	1	1	1	1	1
$D$	0	0	1	0	1	0	1	0	1	1	1	1	1	1

$$S^s \rightarrow aB^t \mid \underline{AaB^t}$$

$$B^s \rightarrow C^t D^t \mid c$$

$$D^s \rightarrow ab \mid a$$

$$C^s \rightarrow B^t C^t \mid d$$

$A$  è l'unico nonterminale inutile. Dunque (al passo 6) si cancella la sola produzione  $S \rightarrow AaB$ .

### Passo 3 Eliminazione dei nonterminali ciclici.

Un'altra operazione di potatura alla quale si assoggetta una grammatica, prima di ogni ulteriore elaborazione, consiste nella eliminazione di eventuali *nonterminali ciclici*.

#### Definizione 8.13 (Nonterminale ciclico)

Un *nonterminale A* è *ciclico* se:

$$A \stackrel{+}{\Rightarrow} A$$

ossia se  $A$  genera se stesso in un numero finito di passi. ■

L'algoritmo seguente calcola, per ogni nonterminale  $A$ , l'insieme:

$$c(A) = \left\{ B \mid A \xrightarrow{+} B \right\}$$

**Algoritmo 8.2.3** (Calcolo dell'insieme  $c(A)$ ).

- 1)  $\forall A \in V: c(A) \leftarrow \emptyset;$
- 2)  $\forall A \rightarrow \alpha \in P$ , dove  $\alpha = \beta B \gamma$  e  $\beta \xrightarrow{*} \lambda$ ,  $\gamma \xrightarrow{*} \lambda$ ,  
si pone:

$$c(A) \leftarrow c(A) \cup \{B\};$$

- 3)  $\forall A \in V$ , se  $B \in c(A)$ ,  $c(A) \leftarrow c(A) \cup c(B);$
- 4) Se in 3) è mutata la composizione di almeno un insieme  $c$ , si riapplica 2),  
altrimenti l'algoritmo termina.

I nonterminali ciclici sono gli  $A$  per cui  $A \in c(A)$ .

Ricerchiamo gli eventuali nonterminali ciclici in:

$$\begin{array}{ll} S \rightarrow aB & D \rightarrow ab \mid a \\ B \rightarrow CD \mid c & C \rightarrow BC \mid d \end{array}$$

$c()$	1)	2)	3)
$S$	$\emptyset$	$\emptyset$	$\emptyset$
$B$	$\emptyset$	$\emptyset$	$\emptyset$
$C$	$\emptyset$	$\emptyset$	$\emptyset$
$D$	$\emptyset$	$\emptyset$	$\emptyset$

Dunque, non vi sono nonterminali ciclici.

Le eventuali ciclicità possono essere eliminate sopprimendo tutte le produzioni del tipo  $A \rightarrow B$ , senza alterare il linguaggio generato, mediante il seguente algoritmo (Algoritmo 8.2.4) che costruisce, a partire da  $P$ , un nuovo insieme  $P'$  di produzioni.

**Passo 4** *Eliminazione delle produzioni*  $A \rightarrow B$ .

**Algoritmo 8.2.4** (Eliminazione delle produzioni  $A \rightarrow B$ ).<sup>1</sup>

- 1) Si partiziona  $P$  in  $P'$  e  $P''$ , ove  $P''$  contiene tutte e sole le produzioni del tipo:  $A \rightarrow B$

$$P'' = \{ A \rightarrow B \in P \mid A, B \in V \};$$

- 2)  $P' \leftarrow P' \cup \{ A \rightarrow \alpha \mid B \in c(A) \text{ e } B \rightarrow \alpha \in P' \}$

**Passo 5** *Eliminazione delle produzioni del tipo*  $A \rightarrow B\beta$ .<sup>2</sup>

Supponendo che le produzioni che trascrivono  $B$  siano:

$$B \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

la  $A \rightarrow B\beta$  può essere sostituita dalle produzioni:

$$A \rightarrow \alpha_1\beta \mid \alpha_2\beta \mid \dots \mid \alpha_n\beta$$

senza che il linguaggio generato venga modificato.

Se tutti gli  $\alpha_i$  iniziano con un carattere terminale allora il procedimento di modifica innescato dalla  $A \rightarrow B\beta$  ha termine, altrimenti se  $\alpha_k$ ,  $1 \leq k \leq n$ , non inizia con un carattere terminale occorre ripetere il procedimento per  $A \rightarrow \alpha_k\beta$ .

Tale procedimento induce una relazione d'ordine sull'insieme dei nonterminali della grammatica ed ha termine soltanto se:

- a) i nonterminali inutili sono stati già eliminati;
- b) la grammatica non contiene *ricorsioni sinistre*, cioè non esiste alcuna derivazione del tipo:

$$A \xrightarrow{*} A\alpha$$

---

<sup>1</sup> Questo algoritmo può generare non terminali ciclici.

<sup>2</sup> Il seguente procedimento può generare nonterminali inutili.

Sotto le ipotesi a) e b),  $G$  viene trasformata in  $G'$  tale che  $L(G) = L(G')$  ed avente solo produzioni del tipo  $A \rightarrow a\alpha$ ,  $\alpha \in (X \cup V)^*$ ,  $a \in X$ .

A questo punto, è immediato introdurre, per ogni terminale di  $\alpha$ , un nuovo nonterminale e trasformare  $G'$  in forma normale di Greibach.

Tornando al nostro esempio,

$$S \rightarrow aB$$

$$B \rightarrow CD \mid c$$

$$D \rightarrow ab \mid a$$

$$C \rightarrow BC \mid d$$

non vi sono produzioni del tipo  $A \rightarrow B$  (Passo 4) mentre vi sono due produzioni del tipo  $A \rightarrow B\beta$  (Passo 5), vale a dire:

$$B \rightarrow CD \text{ e } C \rightarrow BC.$$

Imponiamo una relazione d'ordine sull'insieme dei nonterminali della grammatica:

$$V = \{S, B, C, D\} = \{A_1, A_2, A_3, A_4\}$$

$$\text{ord}(S) < \text{ord}(B) < \text{ord}(C) < \text{ord}(D)$$

Desideriamo che le produzioni di  $G$  siano di questo tipo:

(a)  $A_i \rightarrow A_j v$ , con  $i < j$ ;

oppure

(b)  $A_i \rightarrow av$ ,

con  $a \in X$ ,  $v \in (X \cup V)^*$ .

Esaminiamo le produzioni di  $G$ , tenendo presente l'ordine imposto sull'insieme dei nonterminali:

$S \rightarrow aB$	è di tipo (b)
$B \rightarrow CD$	è di tipo (a)
$B \rightarrow c$	è di tipo (b)
$D \rightarrow ab \mid a$	è di tipo (b)
$C \rightarrow d$	è di tipo (b)
$C \rightarrow BC$	non è né di tipo (a) (perché $ord(C) > ord(B)$ ) né di tipo (b).

La trasformiamo come segue:

le produzioni che trascrivono  $B$  sono:

$$B \rightarrow \underbrace{CD}_{\alpha_1} \mid \underbrace{c}_{\alpha_2}$$

La  $C \rightarrow B \underbrace{C}_{\beta}$  può dunque essere sostituita da:

$$C \rightarrow \underbrace{CD}_{\alpha_1} \mid \underbrace{C}_{\beta} \mid \underbrace{c}_{\alpha_2} \mid \underbrace{C}_{\beta}$$

ottenendo ancora una grammatica equivalente a  $G$ :

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow CD \mid c \\ D &\rightarrow ab \mid a \\ C &\rightarrow CDC \mid cC \mid d \end{aligned}$$

#### Passo 6 Eliminazione delle ricorsioni sinistre.

Supponiamo che  $G$  abbia una produzione del tipo

$$A \rightarrow Av \mid w$$

con  $v, w \in (X \cup V)^*$  e  $w \neq A\gamma$ ,  $\gamma \in (X \cup V)^*$ .<sup>3</sup>

<sup>3</sup> Cioè  $w$  non inizia per  $A$ .

È immediato osservare che:

$$A \xrightarrow{*} wv^*$$

Ma il linguaggio  $wv^*$  può essere generato per concatenazione di  $w$  e di  $v^*$ , che sono generati rispettivamente da:

$$A \rightarrow w \quad \text{e} \quad B \rightarrow vB \mid \lambda$$

Concatenando queste due grammatiche di tipo ‘3’ si ottiene:

$$A \rightarrow w \mid wB$$

$$B \rightarrow vB \mid v \mid \lambda$$

In  $G$ , l'unica ricorsione sinistra è data da:

$$C \rightarrow \underbrace{CDC}_v \mid \underbrace{cC}_{w_1} \mid \underbrace{d}_{w_2}$$

Dunque, questa produzione può essere trasformata in:

$$C \rightarrow cC \mid d \mid cCE \mid dE$$

$$E \rightarrow DCE \mid DC$$

ottenendo così:

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow CD \mid c \\ D &\rightarrow ab \mid a \\ C &\rightarrow cC \mid d \mid cCE \mid dE \\ E &\rightarrow DCE \mid DC \end{aligned}$$

Poiché abbiamo introdotto un nuovo nonterminale,  $E$ , dobbiamo ridefinire un ordinamento di  $V$ . Se consideriamo il seguente ordinamento:

$$V = \{S, B, C, E, D\} = \{A_1, A_2, A_3, A_4, A_5\}$$

tutte le produzioni di  $G$  sono del tipo (a) o (b).

Ritorniamo ad effettuare il passo 5.

$$B \rightarrow C \underbrace{D}_{\beta} \text{ diventa } B \rightarrow \underbrace{cC}_{\alpha_1} \underbrace{D}_{\beta} \mid \underbrace{d}_{\alpha_2} \underbrace{D}_{\beta} \mid \underbrace{cCE}_{\alpha_3} \underbrace{D}_{\beta} \mid \underbrace{dE}_{\alpha_4} \underbrace{D}_{\beta}$$

considerando le produzioni che trascrivono C:

$$C \rightarrow \underbrace{cC}_{\alpha_1} \mid \underbrace{d}_{\alpha_2} \mid \underbrace{cCE}_{\alpha_3} \mid \underbrace{dE}_{\alpha_4}$$

e

$$E \rightarrow DCE \mid DC \text{ diventa } E \rightarrow abCE \mid aCE \mid abC \mid aC$$

dando luogo a:

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow cCD \mid dD \mid cCED \mid dED \mid c \\ D &\rightarrow ab \mid a \\ C &\rightarrow cC \mid d \mid cCE \mid dE \\ E &\rightarrow abCE \mid aCE \mid abC \mid aC \end{aligned}$$

Poiché non vi sono ricorsioni sinistre, possiamo finalmente effettuare il:

### Passo 7 Introduzione di nuovi nonterminali.

Le produzioni sono ora tutte nella forma:

$$A \rightarrow a\alpha, \alpha \in (X \cup V)^*$$

Per ogni terminale in  $\alpha$ , introduciamo un nuovo nonterminale ed una opportuna produzione.

Si ottiene quindi la grammatica G:

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow cCD \mid dD \mid cCED \mid dED \mid c \\ D &\rightarrow aF \mid a \end{aligned}$$

$$\begin{aligned}C &\rightarrow cC \mid d \mid cCE \mid dE \\E &\rightarrow aFCE \mid aCE \mid aFC \mid aC \\F &\rightarrow b\end{aligned}$$

$G$  è in forma normale di Greibach.

La possibilità di trasformare in modo algoritmico una qualunque grammatica non contestuale in una grammatica in forma normale di Greibach è un risultato particolarmente importante.

Da esso discendono i seguenti due teoremi:

#### Teorema 8.4

Ogni linguaggio non contestuale è generabile da una grammatica in forma normale di Greibach; tale grammatica è costruibile in modo algoritmico.

#### Teorema 8.5

Ogni linguaggio non contestuale è riconoscibile da un automa a pila.

Si può anche dimostrare il seguente risultato:

#### Teorema 8.6

Ogni linguaggio riconoscibile da un automa a pila è generabile da una grammatica non contestuale.

Questo teorema con i due precedenti, stabilisce l'*equivalenza tra grammatiche non contestuali e automi a pila*.

### Esercizio 8.3

Convertire la seguente grammatica in forma normale di Greibach:

$$S \rightarrow 00A \mid B \mid 1$$

$$A \rightarrow 1AA \mid 2$$

$$B \rightarrow 0$$

Nell'Esercizio 8.1 abbiamo trasformato la grammatica data in forma normale di Chomsky:

$$S \rightarrow BD \mid 0 \mid 1$$

$$D \rightarrow BA$$

$$A \rightarrow CE \mid 2$$

$$E \rightarrow AA$$

$$C \rightarrow 1$$

$$B \rightarrow 0$$

I passi 1) - 4) della procedura di trasformazione di una CFG in una grammatica in forma normale di Greibach ad essa equivalente non modificano la grammatica.

Eliminiamo le produzioni che non sono in forma normale di Greibach, ossia quelle del tipo:

$$A \rightarrow B\beta$$

Esse sono:

$$S \rightarrow BD$$

$$D \rightarrow BA$$

$$A \rightarrow CE$$

$$E \rightarrow AA$$

$S \rightarrow BD$	diventa	$S \rightarrow 0D$	per mezzo di $B \rightarrow 0$
$D \rightarrow BA$	diventa	$D \rightarrow 0A$	per mezzo di $B \rightarrow 0$
$A \rightarrow CE$	diventa	$A \rightarrow 1E$	per mezzo di $C \rightarrow 1$
$E \rightarrow AA$	diventa	$E \rightarrow 1EA \mid 2A$	per mezzo di $A \rightarrow 1E \mid 2$

e la grammatica in forma normale di Greibach, equivalente a quella di partenza, è:

$$S \rightarrow 0D \mid 0 \mid 1$$

$$D \rightarrow 0A$$

$$A \rightarrow 1E \mid 2$$

$$E \rightarrow 1EA \mid 2A$$

$$\begin{array}{c} C \not\rightarrow 1 \\ B \not\rightarrow 0 \end{array}$$

ove le due ultime produzioni sono state eliminate in virtù del fatto che  $C$  e  $B$  sono nonterminali inutili.

## Bibliografia

### Capitolo 1.

Mauri, G., *Informatica teorica*, in Enciclopedia delle Scienze, Elettronica - Informatica - Comunicazioni, De Agostini, Novara, 1984.

Rayward-Smith, V.J., *A First Course in Formal Language Theory*, Blackwell Scientific Publications, Oxford, United Kingdom, 1983.

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

### Capitolo 2.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Di Martino, L., Tamburini, M.C., *Appunti di ALGEBRA*, CLUED, Milano, 1983.

Borzacchini, L., *Dispense del corso di Logica Matematica*, Bari, 1985.

### Capitolo 3.

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

### Capitolo 4.

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Hopcroft, J.E., Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Massachussetts, 1979.

Rayward-Smith, V.J., *A First Course in Formal Language Theory*, Blackwell Scientific Publications, Oxford, United Kingdom, 1983.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

Aho, A.V., Ullman, J.D., *The Theory of Parsing, Translation, and Compiling*, Volume I: Parsing, Prentice-Hall, Englewood Cliffs, N.J., 1972.

## **Capitolo 5.**

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

Bartezzaghi, S., *Accavallavacca - inventario di parole da gioco*, Bompiani, Milano, 1992.

## **Capitolo 6.**

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Rayward-Smith, V.J., *A First Course in Formal Language Theory*, Blackwell Scientific Publications, Oxford, United Kingdom, 1983.

Crespi-Reghizzi, S., *Sintassi, semantica e tecniche di compilazione*, Volume primo: metodi sintattici, CLUP, Milano, 1985.

## **Capitolo 7.**

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Manna, Z., *Teoria Matematica della Computazione*, Boringhieri, Torino, 1978.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

## **Capitolo 8.**

Moll, R.N., Arbib, M.A., Kfoury, A.J., *An Introduction to Formal Language Theory*, Springer-Verlag, New York, 1988.

Crespi-Reghizzi, S., Della Vigna, P.L., Ghezzi, C., *Linguaggi Formali e Compilatori*, ISEDI, Petrini Editore, Milano, 1985.

Brookshear, J.G., *Theory of Computation - Formal Languages, Automata, and Complexity*, The Benjamin/Cummings Publishing Company, Redwood City, California, 1989.