9. LINGUAGGI FORMALI E GRAMMATICHE DI CHOMSKY

Linguaggi e informatica

@ Presenti in tutte le applicazioni e in tutte le fasi di uso di un computer:

- linguaggi di specifica, di programmazione, di scripting
- linguaggi di configurazione
- compilatori e interpreti
- linguaggi di marcatura (html, xml ecc)
- protocolli di comunicazione
- interazione uomo macchina (sequenze di click)

@ Paradigmatici nella teoria: molti importanti problemi informatici sono riconducibili a quello dell'appartenenza di una stringa a un linguaggio

@ Definibili in modo formale secondo diversi approcci

9.1 ALFABETI, STRINGHE, LINGUAGGI

Def. Un alfabeto è un insieme finito non vuoto di simboli (caratteri).

Esempi:
Alfabeto binario: {0,1}

Alfabeto decimale: {0,1,2,3,4,5,6,7,8,9}

Alfabeto italiano: {a,b,c,d,e,f,g,h,i,l,m,n,o,p,q,r,s,t,u,v,z}

Alfabeto di simboli utilizzati nel linguaggio Java: {a, … , z, A, … , Z, +, -, *, /, =, {, }, “, ...}

Def. Dato un alfabeto Σ definiamo stringa o parola una sequenza finita di caratteri di Σ. Data una stringa x il numero di caratteri che la costituiscono è chiamato lunghezza della stringa ed è indicato come segue: lxl. La stringa di lunghezza zero (cioè non costituita da alcun simbolo) è chiamata stringa vuota o nulla. L'insieme di tutte le stringhe definite sull'alfabeto Σ (inclusa la stringa vuota) è denotato Σ^*.

Esempi:
01000010 è una stringa definita sull'alfabeto binario {0, 1}. La sua lunghezza è 8. Essa appartiene quindi a {0, 1}*.
La sequenza infinita 010101 ... non appartiene a {0, 1}*.
abracadabra è una stringa definita sull'alfabeto italiano. La sua lunghezza è 11.

Def. Si definisce concatenazione l'operazione (denotata °) che consiste nel giustapporre due stringhe. La concatenazione è un'operazione associativa ma non commutativa.

Esempi:

salva ° gente = salvagente

abb ° bba = abbbba ≠ bba ° abb = bbaabb

((abb ° b) ° ba) = (abb ° (b ° ba)) = abbbba

Per ogni stringa x, $x \circ \varepsilon = \varepsilon \circ x = x$

Per indicare la ripetizione di simboli o più in generale la concatenazione di due o più stringhe uguali si usa il simbolo di potenza.

Esempi:

$ab^4a = abbbba$

$w^3 = www$

Se $x = cous$, con x^2 indichiamo la stringa couscous.

NOTA BENE

Non è difficile verificare che, dato un alfabeto Σ, la terna $<\Sigma^*, \circ, \varepsilon>$ è un monoide, cioè un insieme chiuso rispetto all'operazione di concatenazione $\circ$: $\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ e per il quale ε è l'elemento neutro. Esso è chiamato monoide sintattico definito su Σ poiché è alla base della definizione sintattica dei linguaggi.

Generalmente con Σ^+ denotiamo l'insieme delle stringhe non vuote. Pertanto $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Def. Un linguaggio è un sottoinsieme di Σ^*

Esempi:

Dato l'alfabeto {a, b}, l'insieme $L = \{a^n b^n \mid n \geq 0\}$
è il linguaggio di tutte le stringhe costituite da una sequenza di n a (n ≥ 0), seguite da altrettante b. aaabbb $\in \Lambda$; $\varepsilon \in L$; aaabb $\notin \Lambda$.

Dato l'alfabeto {I, V, X, L, C, D, M}, l'insieme di tutti i numeri da 1 a 3000 rappresentati come numeri romani è un linguaggio.

Dato l'alfabeto {0, 1} l'insieme di tutte le stringhe che contengono un numero pari di 1 è un linguaggio.

In particolare, dato l'alfabeto Σ:

- Σ stesso è un linguaggio,
- Σ^* è un linguaggio,
- l'insieme che non contiene nessuna stringa, denotato Λ o $\emptyset$ è un linguaggio (detto linguaggio vuoto)

NOTA BENE $\Lambda \neq \{\varepsilon\}$

Chiaramente non tutti i linguaggi su un dato alfabeto sono interessanti. Noi siamo interessati a linguaggi le cui stringhe hanno struttura particolare ovvero obbediscono a particolari regole.
Ad esempio:

- il linguaggio costituito da stringhe di parentesi bilanciate del tipo: (( )(( ))( ),
- il linguaggio costituito da espressioni aritmetiche contenenti identificatori di variabili e simboli delle quattro operazioni,
- il linguaggio costituito da tutti i programmi sintatticamente corretti (cioè accettati da un compilatore senza segnalazione di errore) scritti nel linguaggio C.

Per definire un linguaggio possiamo utilizzare diversi approcci:

- un approccio algebrico, mostrando cioè come il linguaggio è costruito a partire da linguaggi più elementari utilizzando operazioni su linguaggi
- un approccio generativo, definendo cioè mediante una grammatica le regole strutturali che devono essere soddisfatte dalle sue stringhe
- un approccio riconoscitivo, fornando cioè una 'macchina'(un algoritmo di riconoscimento) che ricevendo una stringa in input dice se essa appartiene o no al linguaggio.

Un importante fatto di cui bisogna essere consapevoli è che non tutti i linguaggi possono essere riconosciuti mediante programmi di calcolatore (o, equivalentemente, mediante algoritmi) e non tutti possono essere definiti mediante grammatiche.

In altre parole, supponiamo di chiederci: dato un linguaggio L, è sempre possibile scrivere un programma che "lo riconosce", cioè che, data una stringa, decide se essa appartiene ad L o no?
Come dimostreremo (in modo relativamente semplice) in una parte successiva del corso la risposta a tale domanda è NO.
Infatti la cardinalità dell'insieme (infinito) dei linguaggi definiti su un dato alfabeto Σ (cioè la cardinalità dell'insieme di tutti i sottoinsiemi di Σ^*), è maggiore della cardinalità dell'insieme (infinito) dei possibili programmi. In altre parole, esistono "più" linguaggi che programmi che possono riconoscerli e quindi esistono linguaggi per i quali non può esistere alcun algoritmo di riconoscimento. Tale considerazione vale anche per gli altri metodi di definizione di linguaggi e in particolare per le grammatiche.

9.2 OPERAZIONI SU LINGUAGGI

Operazioni insiemistiche. Siano L_1 ed L_2 linguaggi su Σ^*

unione

$L_1 \cup L_2 = \{x \in \Sigma^* | \; x \in L_1 \vee x \in L_2\}$

$L_1 \cup \Lambda = L_1$

intersezione

$L_1 \wedge L_2 = \{x \in \Sigma^* | \; x \in L_1 \wedge x \in L_2\}$

$L_1 \wedge \Lambda = \Lambda$

complementazione

$\underline{L_1} = \{x \in \Sigma^* | \; x \notin L_1\}$

Operazioni definite sul monoide sintattico.

Siano L_1 ed L_2 linguaggi su Σ^*

concatenazione (prodotto) di linguaggi

$L_1 \circ L_2 = \{x \in \Sigma^* | \text{ esistono } x_1 \text{ ed } x_2 \text{ tali che } x_1 \in L_1 \wedge x_2 \in L_2 \text{ e } x = x_1 \circ x_2\}$

$L \circ \{\varepsilon\} = \{\varepsilon\} \circ L = L$ $\qquad$ $L \circ \Lambda = \Lambda \circ L = \Lambda$

NOTA BENE L'operazione di concatenazione di linguaggi è associativa ma non è commutativa (come la concatenazione di stringhe).

Esempi:

Siano $L_1 = \{a^n b^n \mid n \geq 1\}$ e $L_2 = \{c^m \mid m \geq 1\}$

$L_1 \circ L_2 = \{a^n b^n c^m \mid n,m \geq 1\}$

Siano L_1= {Auguri., Condoglianze.}, L_2= {Tuo },
L_3= {Giorgio, Roberto}
$L_1 \circ L_2 \circ L_3$ = {Auguri.Tuo Giorgio,
Auguri.Tuo Roberto, Condoglianze.Tuo Giorgio,
Condoglianze.Tuo Roberto}

potenza di un linguaggio

Dato un linguaggio L
$L^h = L \circ L^{h-1}$, $h \geq 1$
$L^0 = \{\varepsilon\}$ per convenzione.

Quindi: L1 = L, L2 = L°L ecc.

Esempio:

Sia $L = \{a^n b^n \mid n \geq 1\}$
$L \circ L = \{a^n b^n a^m b^m \mid n,m \geq 1\}$

$aaabbbaabb \in L^2$, $aaabbbaaabbb \in L^2$, $aaabbaabb \notin L^2$

iterazione di un linguaggio

$$L^* = \bigcup_{h=0}^{\infty} L^h$$

NOTA BENE $\varepsilon \in L^*$ (per definizione)

Se si vuole indicare il linguaggio L^* escludendo la stringa vuota si usa il simbolo L^+ cioè

$$L^+ = \bigcup_{h=1}^{\infty} L^h \quad \text{e quindi} \quad L^* = L^+ \cup \{\varepsilon\}$$

Esempi:

L={ab, aab}, L^*={e, ab, aab, abab, abaab, aabab, aabaab, ecc.}

L={a, b}, L^* contiene tutte le stringhe definite sui due simboli a e b, inclusa la stringa vuota

L={aa, bb}, L^* contiene tutte le stringhe costituite da un numero pari di a e un numero pari di b, inclusa la stringa vuota

$L = \{a^n b^n \mid n \geq 1\}$, $L^* = \{a^{n1} b^{n1} a^{n2} b^{n2} \ldots a^{nk} b^{nk} \mid k \geq 0, n1, n2, \ldots nk \geq 1\}$ (ad esempio: aabbaaabbbabaabb $\in L^*$ e anche $\varepsilon \in L^*$)

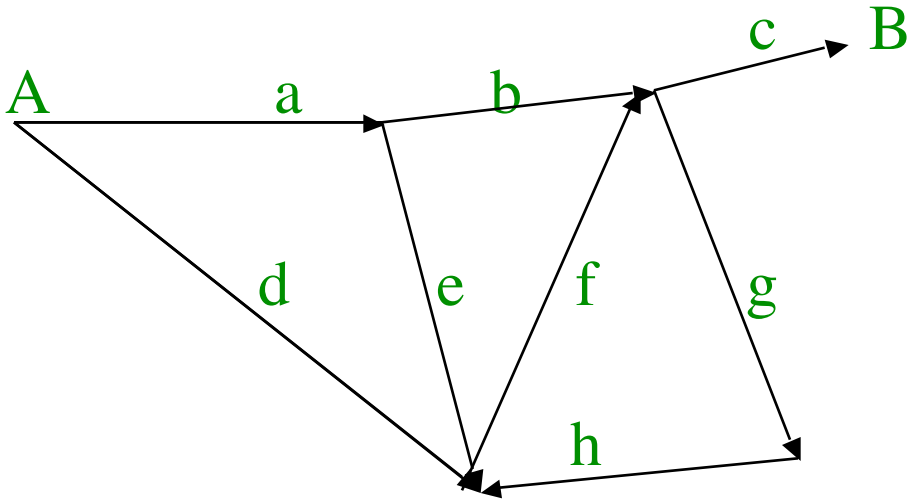
Infine si noti che $\Lambda^*=\{\varepsilon\}$

9.3 ESPRESSIONI REGOLARI

Le espressioni regolari sono un metodo per rappresentare linguaggi. Ad ogni espressione regolare ‘e’ corrisponde il linguaggio L(e) che essa rappresenta. La rappresentazione è basata sulle operazioni che ci permettono di definire L(e) a partire da linguaggi elementari. In una espressione regolare usiamo il simbolo + per indicare l’unione di linguaggi, il simbolo · (o semplicemente nessun simbolo) per indicare la concatenazione e il simbolo * per indicare l’iterazione.

Esempio intuitivo: se vogliamo rappresentare il linguaggio costituito da tutte le stringhe che cominciano con a, proseguono con un numero arbitrario (anche nullo) di b e terminano con la stringa bab o con la stringa aba possiamo scrivere ab*(bab+aba)

Un altro esempio intuitivo che ci permette di capire la struttura e l'uso delle espressioni regolari è il seguente. Sia data la rete stradale in figura.



Tutti i percorsi da A a B (inclusi i cicli) sono rappresentati dalla espressione regolare (ab + aef + df)(ghf)*c

Def. Dato un alfabeto Σ, chiamiamo espressione regolare una stringa r sull'alfabeto

$$(\Sigma \cup \{+,*,(,),\cdot,\emptyset\})$$

tale che:

1. $r = \emptyset$ oppure
2. $r \in \Sigma$ oppure
3. $r = (s+t)$ oppure $r = (s \cdot t)$ oppure $r = s^*$,
   dove s e t sono essere stesse espressioni regolari

Il significato di una espressione regolare e (cioè il linguaggio L(e) da essa rappresentato) è definito induttivamente come segue:

espressione	linguaggio
$\emptyset$	Λ
a	$\{a\}$ (per ogni simbolo $a \in \Sigma$)
$(s+t)$, $(s \cdot t)$, s^*	rispettivamente $L(s) \cup L(t)$, $L(s) \circ L(t)$, $L(s)^*$

Esempio:

Data l'espressione $r = (((a \cdot a) + b) \cdot c^*)$ abbiamo

$L(r) = L(((a \cdot a) + b) \cdot c^*) = L((a \cdot a) + b) \circ L(c^*)$

in cui $L((a \cdot a) + b) = L(a \cdot a) \cup L(b)$ e $L(c^*) = (L(c))^*$.
Inoltre $L(a \cdot a) = L(a) \circ L(a)$

Quindi $L(r) = ((L(a) \circ L(a)) \cup L(b)) \circ (L(c))^*$
dove $L(a) = \{a\}$, $L(b) = \{b\}$, $L(c) = \{c\}$ sono i linguaggi elementari costituiti da una sola stringa di un solo carattere.

In definitiva $L(r) = \{aa, b, aac, bc, aacc, bcc, ...\}$ cioè il linguaggio costituito da tutte le stringhe che iniziano con aa o b e proseguono con un numero arbitrario (anche nullo) di c.

Per semplificare la scrittura delle espressioni regolari possiamo sfruttare:

- l'eliminazione del simbolo della concatenazione, che ci permette di scrivere (st) anzichè (s·t)
- le precedenze tra operatori: * > · > + che ci permettono di usare meno parentesi.

Esempi:

(a+(b·(c·d))) = a +bcd

(((a·a) + b) · c*) = (aa+b)c*

(((((a·b) + ((a·e)·f)) + (d·f))((g·h)·f)*)·c) = (ab + aef + df)(ghf)*c

Quali linguaggi possono essere rappresentati mediante espressioni regolari?

Come vedremo, i linguaggi rappresentabili con espressioni regolari sono una sottoclasse piccola (ma utile ed interessante) di tutti i possibili linguaggi, sottoclasse che non a caso è chiamata classe dei linguaggi regolari.

9.4 GRAMMATICHE

Approccio generativo alla descrizione dei linguaggi
Metodo di costruzione delle stringhe del linguaggio basato sulla riscrittura. Il concetto di grammatica (in matematica e informatica) ha una lunga storia:

1914 Axel Thue studia i primi problemi di riscrittura.
1920-40 Emil Post definisce sistemi di produzione.
1947 A.A. Markov definisce algoritmi basati su regole di riscrittura.
1956 N. Chomsky introduce le grammatiche formali nell'ambito degli studi sul linguaggio naturale.
1960 J. W. Backus e P. Naur introducono la BNF per descrivere la struttura sintattica dei programmi.
1990 T. Berners Lee introduce i linguaggi di marcatura per definire la struttura sintattica di pagine web e documenti.

Le Grammatiche di Chomsky

Def. Una grammatica formale è un sistema $G = \langle V_T, V_N, P, S \rangle$ caratterizzato da:

- V_T (Σ) alfabeto finito di simboli detti terminali,
- V_N alfabeto di simboli non terminali,
- P, detto insieme di produzioni, è un insieme di coppie del tipo $\langle \alpha, \beta \rangle$ (indicate con $\alpha \rightarrow \beta$) in cui $\alpha \in (V_T \cup V_N)^* \circ V_N \circ (V_T \cup V_N)^*$ e $\beta \in (V_T \cup V_N)^*$ (cioè α è una stringa che contiene almeno un non terminale e β è una stringa qualunque di terminali e non terminali, anche vuota),
- $S \in V_N$ è detto assioma.

Le regole di produzione sono regole di riscrittura. Ogni regola $\alpha \rightarrow \beta$ dice che la stringa α può essere rimpiazzata dalla stringa β. Le regole di produzione definiscono come, a partire dall'assioma, si possono generare via via stringhe di terminali e non terminali fino a che, eventualmente, si giunge ad una stringa di soli terminali.

Informalmente**:** Il linguaggio generato da una grammatica è l'insieme delle stringhe di terminali ottenibili applicando una sequenza finita di passi di riscrittura consistenti nell'applicazione delle regole di produzione.

Esempio. G=<{a,b},{S},P,S>
in cui P è l'insieme delle seguenti produzioni

S → aSb

S → ab

può generare tutte le stringhe del tipo $a^n b^n$ con n≥1. Infatti abbiamo, ad esempio, che applicando la prima regola all'assioma (cioè scrivendo aSb al posto di S) otteniamo aSb, applicandola ancora otteniamo aaSbb e applicando infine la seconda regola otteniamo aaabbb.

La notazione per le produzioni può essere resa più sintetica scrivendo $\alpha \rightarrow \beta_1 \mid ..... \mid \beta_n$ anziché:

$\alpha \rightarrow \beta_1$

.....

$\alpha \rightarrow \beta_n$

Questo ci consente ad esempio di scrivere $S \rightarrow aSb \mid ab$ anziché

$S \rightarrow aSb$

$S \rightarrow ab$

9.5 DERIVAZIONE DI STRINGHE E GENERAZIONE DI LINGUAGGI

Per poter dare una definizione più precisa del linguaggio generato da una grammatica e passare ad esempi più complessi abbiamo bisogno di spiegare formalmente come si applicano le produzioni di una grammatica. Definiamo innanzitutto i concetti di derivazione diretta e di derivazione.

Nel seguito useremo il simbolo V per intendere $V_T \cup V_N$

Def. Data una grammatica $G = \langle V_T, V_N, P, S \rangle$ una stringa ψ si ottiene da una stringa ϕ per derivazione diretta se esistono due stringhe γ e δ (eventualmente vuote) tali che $\phi = \gamma\alpha\delta$, $\psi = \gamma\beta\delta$ e P contiene la produzione $\alpha \rightarrow \beta$. In tal caso si scrive $\phi \Rightarrow \psi$.

Def. Data una grammatica $G = \langle V_T, V_N, P, S \rangle$ una stringa ψ si ottiene da una stringa ϕ per derivazione se esiste una sequenza di stringhe $\xi 1, \xi 2, \ldots, \xi n$ tali che $\xi 1 = \phi$ e $\xi n = \psi$ e ognuna delle stringhe è ottenuta per derivazione diretta dalla precedente. In tal caso si scrive $\phi \Rightarrow^* \psi$.

Esempi

1) Data la grammatica G=<{a,b},{S,B,C},P,S>
con P consistente delle seguenti produzioni

S → aS | B
B → bB | bC
C → cC | c

abbiamo che la stringa aaabbC si ottiene per derivazione diretta da aaabB applicando la quarta produzione, cioè aaabB ⇒ aaabbC e la stringa aabbC si ottiene per derivazione da aS, cioè aS ⇒* aabbC.

2) Data la grammatica G=<{a,b},{S,B,C},P,S>
con P consistente delle produzioni $S \rightarrow aSb \mid ab$
abbiamo che la stringa aaSbb si ottiene per derivazione diretta da aSb applicando la prima produzione, cioè $aSb \Rightarrow aaSbb$ e la stringa aaabbb si ottiene per derivazione da S, cioè $S \Rightarrow^{*} aaabbb$**.**

Def. Data una grammatica G, una forma di frase è una qualunque stringa x tale che $x \in V^*$ e $S \Rightarrow^* x$. Il linguaggio generato da G è l'insieme di particolari forme di frase, costituite di soli terminali.

Def. Il linguaggio generato da G è il linguaggio
$L(G) = \{x \mid x \in V_T^* \wedge S \Rightarrow^* x\}$

Esempi

1) La grammatica G=<{a,b},{S, A},P,S> con le produzioni:
$S \rightarrow aAb \mid ab$
genera il linguaggio $\{a^n b^n \mid n \geq 1\}$.

2) La grammatica G=<{a,b},{S, A},P,S> con le produzioni:
$S \rightarrow aB \mid b$
$B \rightarrow aS$

genera il linguaggio $\{a^n b \mid n \geq 0$ e pari$\}$.

Def. Due grammatiche G_1 e G_2 si dicono equivalenti se $L(G_1) = L(G_2)$.

Esempio Le grammatiche
G1 =<{a,b},{S, A},P,S> con le produzioni:
S → Ab | b
A → aAa | aa

e G2 =<{a,b},{S, A},P,S> con le produzioni:
S → Ab
A → Aaa | ε

sono equivalenti perchè generano entrambe il linguaggio
$\{a^n b \mid n \geq 0 \text{ e pari}\}$.

Esempio. Mostriamo come possiamo generare il linguaggio $\{a^n b^n c^n \mid n \geq 1\}$

Grammatica: G=<{a,b,c},{S,A,B,C},P,S>
con le produzioni

1 S → aSBC
2 S → aBC
3 CB → BC
4 aB → ab
5 bB → bb
6 bC → bc
7 cC → cc

Per generare 'aaabbbccc' si effettua la seguente derivazione (la stringa che viene riscritta è sottolineata, il risultato della riscrittura è in viola, il numero rappresenta la produzione applicata):

S (1) ⇒ aSBC
(1) ⇒ aaSBCBC
(2) ⇒ aaaBCBCBC
(3) ⇒ aaaBCBBCC
(3) ⇒ aaaBBCBCC
(3) ⇒ aaaBBBCCC
(4) ⇒ aaabBBCCC
(5) ⇒ aaabbBCCC
(5) ⇒ aaabbbCCC
(6) ⇒ aaabbbcCC
(7) ⇒ aaabbbccC
(7) ⇒ aaabbbccc

NOTA BENE. Non è detto che una sequenza di derivazioni porti ad una stringa del linguaggio; essa può portare ad una forma di frase in cui non si può più applicare alcuna produzione.

Esempio. Con la grammatica precedente può accadere quanto segue:

S (1) ⇒ aSBC
(1) ⇒ aaSBCBC
(2) ⇒ aaaBCBCBC
(4) ⇒ aaabCBBCC
(6) ⇒ aaabcCBCC
(3) ⇒ aaabcBCCC

A questo punto non è più applicabile alcuna produzione e la forma di frase ottenuta non darà luogo ad alcuna stringa del linguaggio.

Altri esempi.

1) La grammatica
G=<{a, b, c}, {S,A}, P, S> con

$S \rightarrow aSc \mid A$

$A \rightarrow bAc \mid \varepsilon$

genera il linguaggio $\{a^n b^m c^{n+m} \mid n \geq 1, m \geq 0\}$

2) La grammatica G=<{a,b,c},{S,A},P,S> con

$S \rightarrow Ab$

$A \rightarrow Sa$

non genera alcuna stringa (cioè genera Λ, il linguaggio vuoto).

9.6 TIPI DI LINGUAGGI

Le grammatiche di Chomsky si classificano in base al tipo di produzio
che utilizzano:

- tipo 0, produzioni di tipo generale (non limitate)
- tipo 1, produzioni contestuali (context sensitive: CS)
- tipo 2, produzioni non contestuali (context free: CF),
- tipo 3, produzioni regolari

Def. Le grammatiche di Chomsky di tipo 0, sono basate sulle produzion più generali, del tipo:

$$\alpha \rightarrow \beta \ , \ \alpha \in V^* \circ V_N \circ V^* , \ \beta \in V^*$$

NOTA BENE. Le grammatiche di tipo 0 ammettono l'uso di produzion che hanno il lato destro più corto del lato sinistro (in particolare possonc anche avere il lato destro vuoto) e quindi possono dar luogo a derivazio che accorciano stringhe.

Def. I linguaggi di tipo 0 sono i linguaggi generati da grammatiche di ti 0 (cioè sono tutti i linguaggi generabili con grammatiche di Chomsky).

Def. Le grammatiche di Chomsky di tipo 1, (dette context sensitive o contestuali) sono basate su produzioni del tipo:

$$\alpha \rightarrow \beta \ , \ \alpha \in V^* \circ V_N \circ V^*, \ \beta \in V^+$$
$$\text{con } |\alpha| \leq |\beta|$$

NOTA BENE. Le produzioni di tipo 1 non riducono la lunghezza delle forme di frase.

Def. I linguaggi di tipo 1 (context sensitive o contestuali) sono i linguag generati da grammatiche di tipo 1.

Esempio. Il linguaggio $\{a^n b^n c^n \mid n \geq 1\}$ è di tipo 1, infatti può essere generato dalla grammatica vista precedentemente, con le produzioni:

$S \rightarrow aSBC \mid aBC$
$CB \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$

che soddisfano il vincolo che il lato destro è almeno lungo quanto il latc sinistro.

Def. Le grammatiche di Chomsky di tipo 2, (dette context free o non contestuali) sono basate su produzioni del tipo:

$$A \rightarrow \beta \quad \text{in cui } A \in V_N,\ \beta \in V^+$$

Def. I linguaggi di tipo 2 (context free o non contestuali) sono i linguag generati da grammatiche di tipo 2.

Esempio. Il linguaggio $\{a^n b^n \mid n \geq 1\}$ è di tipo 2 in quanto può essere generato dalla grammatica con le produzioni:

$$S \rightarrow aSb \mid ab$$

Def. Le grammatiche di Chomsky di tipo 3, (dette regolari) sono basate produzioni del tipo:

$$A \rightarrow aB \text{ oppure } A \rightarrow a,$$

con $A, B \in V_N$, $a \in V_T$

Def. I linguaggi di tipo 3 (regolari) sono i linguaggi generati da grammatiche di tipo 3.

Esempio. Il linguaggio $\{a^n b \mid n \geq 0\}$ è di tipo 3 in quanto si può generare con la grammatica con le produzioni:

$S \rightarrow aS$
$S \rightarrow b$

Si noti che ogni tipo di produzioni e di grammatiche determina una restrizione delle possibilità, quindi le grammatiche di tipo 2 sono anche tipo 1 e di tipo 0 e quelle di tipo 3 sono anche di tipo 2 e di tipo 1. Analogamnete i linguaggi di tipo 1 sono un sottoinsieme dei linguaggi d tipo 0, quelli di tipo 2 sono un sottoinsieme di quelli di tipo 2 e così via.

Per caratterizzare un linguaggio siamo interessati a capire quali sono le grammatiche più semplici che lo generano, quindi diremo che un linguaggio è strettamente di tipo i se non esiste una grammatica di tipo j (con $j > i$) che lo genera. Come vedremo potremo dimostrare che il linguaggio $\{a^n b^n \mid n \geq 1\}$ è strettamente di tipo 2, perché non esiste una grammatica di tipo 3 che lo genera, e che il linguaggio $\{a^n b^n c^n \mid n \geq 1\}$ è strettamente di tipo 1, perché non può essere generato con grammatiche tipo 2 o di tipo 3.

9.7 ESEMPI ED ESERCIZI

Nel seguito forniremo ulteriori approfondimenti sui vari tipi di grammatiche e mostreremo altri esempi. Inoltre presenteremo alcuni esercizi consistenti nel derivare stringhe di un linguaggio, data la grammatica, o nel definire una grammatica che genera un dato linguaggio.

Esempio. Generazione di espressioni aritmetiche con somma e prodotto
il simbolo a indica una generica variabile. L'assioma è il non terminale

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow a \mid (E)$

La grammatica specifica che una generica espressione è costituita da un
somma di termini (o eventualmente da un unico termine), ogni termine
costituito dal prodotto di più fattori (o da un unico fattore), ed ogni fatto
è il nome di una variabile o una espressione tra parentesi. Mostriamo la
derivazione dell'espressione a+ a*a:

$E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow T+T*F \Rightarrow F+T*F \Rightarrow F+F*F$
$\Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$

Come si deriva l'espressione (a+a)*(a+a) ?

Esempio. Generazione di sequenze di parentesi ben bilanciate:

$S \rightarrow ( ) \mid SS \mid (S)$

La grammatica indica che una sequenza di parentesi ben bilanciate è costituita

- o da una semplice coppia di parentesi
- o da due sequenze corrette poste una di seguito all'altra
- o da una seuenza corretta racchiusa tra due parentesi bilanciate.

Mostriamo la generazione della stringa ( ) ( ) (( )):

$S \Rightarrow SS \Rightarrow SSS \Rightarrow ( )SS \Rightarrow ( ) ( ) S \Rightarrow ( ) ( ) (S) \Rightarrow ( ) ( ) (( ))$

Da quale sequenza di produzioni è generata la sequenza di parentesi: ( )((( )( ))) ?

Def. Data una stringa w su un dato alfabeto Σ, definiamo stringa simmetrica (o inversa) la stringa w^R così definita:

se $w = \varepsilon$, allora $w^R = \varepsilon$

se $w = \sigma x$ con σ in Σ e x in Σ^*,

allora $w^R = x^R\sigma$

Chiaramente, $w = (w^R)^R$

Esempio. Se w = ROMA, w^R = AMOR

Esercizio. Definire la grammatica che consente di generare il linguaggi
{ x | esiste w tale che $x = ww^R$}

Def. Stringhe palindrome sono le stringhe che si possono leggere indifferentemente da sinistra verso destra o da destra verso sinistra.

Esempio. Le stringhe ANNA, ALA, BOB, AMOROMA sono stringhe palindrome.

Esercizio. Definire la grammatica che consente di generare tutte le stringhe palindrome sull'alfabeto {a,b}.

Si noti che le stringhe del tipo $x = ww^R$ sono anch'esse palindrome (so palindrome di lunghezza pari).

Esercizio. Chiamando L il linguaggio delle stringhe palindrome di lunghezza pari, definire una grammatica per generare il linguaggio L*. Il linguaggio L* contiene ad esempio la stringa: aabbaaabbbababbbab.

Esempio. Generare il linguaggio $\{w\ w \mid w \in (a+b)^*\}$ cioè il linguaggio delle stringhe ripetute.

1 $S \rightarrow T \mid \varepsilon$
$T \rightarrow aAT \mid bBT \mid A_0a \mid B_0b$

2 $Aa \rightarrow aA$
$Ab \rightarrow bA$
$Ba \rightarrow aB$
$Bb \rightarrow bB$

3 $AA_0 \rightarrow A_0a$
$BA_0 \rightarrow A_0b$
$AB_0 \rightarrow B_0a$
$BB_0 \rightarrow B_0b$

4 $A_0 \rightarrow a$
$B_0 \rightarrow b$

Le produzioni 1 generano insieme caratteri della prima e della seconda stringa; A_0 (B_0) è l'ultimo carattere della prima stringa.

Le produzioni 2 e 3 separano la prima stringa dalla seconda. Le produzioni 4 chiudono la generazione, se sono applicate troppo presto il processo non genera una stringa di soli terminali.

Perché le grammatiche di tipo 1 sono chiamate "contestuali" e quelle di tipo 2 "non contestuali"?

Nella originaria formulazione data da Chomsky le grammatiche di tipo sono definite come quelle grammatiche le cui produzioni hanno la form

$\alpha \rightarrow \gamma$ con $\alpha=\phi_1 A \phi_2$ e $\gamma=\phi_1 \beta \phi_2$, e con $A \in V_N$, $\phi_1,\phi_2 \in V^*$, $\beta \in V^+$

Quindi, se $|\phi_1| \geq 1$ o $|\phi_2| \geq 1$, la produzione esprime il fatto che il non terminale A viene rimpiazzato dalla stringa β solo se appare nel contest delle stringhe ϕ_1 e ϕ_2. Per tale motivo quelle produzioni (e le relative grammatiche) sono chiamate "contestuali". Nel caso che le stringhe ϕ_1 e ϕ_2 siano entrambe vuote le produzioni di tipo contestuale divengon di tipo $A \rightarrow \beta$ cioè "non contestuali", infatti in tal caso il non terminale può essere rimpiazzato dalla stringa β indipendentemente dal contesto i cui esso si trova.

Esempio. Nella grammatica per il linguaggio $\{a^n b^n c^n \mid n \geq 1\}$ abbiamo alcune produzioni contestuali secondo la definizione di Chomsky:

$$aB \rightarrow ab,\ bB \rightarrow bb,\ bC \rightarrow bc,\ cC \rightarrow cc$$

mentre la produzione $CB \rightarrow BC$ è di tipo 1 ma non è contestuale secon la definizione di Chomsky. Infine le produzioni

$$S \rightarrow aSBC \quad S \rightarrow aBC$$

sono non contestuali.

Teorema. Le grammatiche di tipo 1 e le grammatiche contestuali secon Chomsky consentono di generare la stessa classe di linguaggi.

Dim. i) Le produzioni contestuali di Chomsky sono di tipo 1.
ii) Le produzioni di tipo 1 possono essere facilmente trasformate in produzioni contestuali di Chomsky. (**Esercizio**)

Ad esempio la produzione CB →BC può essere trasformata nella sequenza di produzioni (contestuali secondo Chomsky):

CB → CX
CX → BX
BX → BC

9.8 GRAMMATICHE LINEARI

Def. Diciamo che una grammatica di tipo 2 (e quindi anche il linguaggio che essa genera) è lineare se in ogni produzione $A \rightarrow \beta$, la stringa β contiene (al più) un solo simbolo non terminale.

Esempio.
La grammatica seguente, che genera $\{a^n c b^n \mid n \geq 0\}$ è lineare:

$S \rightarrow aSb \mid c$

Un altro esempio di grammatica lineare è quella per le stringhe palindrome. Al contrario la grammatica per le espressioni aritmetiche e quella per le parentesi ben bilanciate non sono lineari.

Si noti che il concetto di grammatica lineare ha una affinità con quello d
equazione lineare, un'equazione in cui ogni termine contiene una sola
variabile.

I linguaggi lineari sono un sottoinsieme di quelli strettamente CF e un
soprainsieme di quelli regolari. Infatti il linguaggio delle stringhe
palindrome è lineare ma non regolare.

Le grammatiche regolari sono un caso particolare di grammatiche linear
Esse vengono chiamate lineari destre (Ld) perchè il non terminale
compare a destra del terminale.

Si possono anche definire grammatiche lineari sinistre (Ls) con
produzioni del tipo:

$A \rightarrow Ba$ oppure $A \rightarrow a$, con $A, B \in V_N$, $a \in V_T$

Esempio. Il linguaggio $\{a^n b \mid n \geq 0\}$ è un linguaggio lineare destro perch si può definire con la grammatica lineare destra seguente:

$S \rightarrow aS \mid b$

Il linguaggio $\{a^n b \mid n \geq 0\}$ può essere anche generato da una grammatica lineare sinistra con le seguenti produzioni:

$S \rightarrow Tb \mid b$
$T \rightarrow a \mid Ta$

Teorema. Le classi di linguaggi lineari destri e lineari sinistri (cioè generabili con grammatiche Ld e Ls) coincidono.

9.9 ε-PRODUZIONI

Le ε-produzioni sono produzioni che generano la stringa vuota. Se la grammatica è di tipo 0 le ε-produzioni rientrano nella definizione ma se grammatica è di tipo 1, 2, 3 in teoria non le ε-produzioni non sarebbero ammesse.

Supponiamo però di voler generare un linguaggio di tipo 1, 2, 3 contenente la stringa vuota. E' indispensabile utilizzare ε-produzioni e quindi dobbiamo estendere le definizioni di grammatica di tipo 1, 2, 3 ammettendo anche ε-produzioni.

Una soluzione che possiamo comunque adottare è la seguente.

Teorema. Supponiamo di avere una grammatica G che genera un linguaggio L(G) che non contiene la stringa vuota. Se L(G) è di tipo 1 o è possibile modificare G in modo che generi $L(G) \cup \{\varepsilon\}$, usando ε-produzioni e senza modificare le altre istruzioni della grammatica.

Dim. Sia data $G=<V_T,V_N,P,S>$; per generare $L'=L(G) \cup \{\varepsilon\}$ basta usare nuova grammatica $G'=<V_T,V_N \cup \{S'\},P',S'>$ con $P' = P \cup \{S' \rightarrow S, S' \rightarrow \varepsilon$

Quindi le ε-produzioni si possono aggiungere all'assioma di grammatich di tipo 1 o 2 senza modificare il resto della grammatica, purché l'assiom non appaia a destra di nessuna produzione.

Cosa accade se le ε-produzioni sono in posizione più generale?

- Nel caso di grammatiche di tipo 1 esse diventano strettamente più potenti, equivalenti alle grammatiche di tipo 0.

- Nel caso di grammatiche di tipo 2 e di tipo 3 non si amplia la classe linguaggi che possono essere generati con produzioni di tipo 2 e 3, rispettivamente, utilizzando eventualmente una ε-produzione applicata sull'assioma.

Teorema. Una grammatica CF estesa con ε-produzioni può generare o meno ε. Nel secondo caso si può costruire una grammatica equivalente senza ε-produzioni; nel primo caso si può costruire una grammatica equivalente con solo ε-produzioni sull'assioma.

Dim. Supponiamo per semplicità che all'inizio esista una sola ε-produzione $E \rightarrow \varepsilon$ e che l'assioma non appaia mai a destra di una produzione; se E è l'assioma il teorema è dimostrato. Supponiamo che E compaia a destra delle $k \geq 1$ produzioni $A_i \rightarrow \beta_i$.

Eliminiamo la produzione $E \rightarrow \varepsilon$ e aggiungiamo k produzioni $A_i \rightarrow \beta_i'$ dove β_i' e' ricavato da β_i sostituendo ε alle E.

Se non ci sono più ε-produzioni o se ci sono solo sull'assioma il procedimento termina, altrimenti il procedimento si ripete per tutte le nuove ε-produzioni. La terminazione è garantita dalla finitezza di V_N .

Si noti che nel caso che la grammatica sia in particolare una grammatica regolare la eliminazione di una ε-produzione non ne genera di nuove e quindi se abbiamo una ε-produzione del tipo $E \rightarrow \varepsilon$ o E è l'assioma, ed allora la grammatica genera la stringa vuota, o E non è l'assioma ed allo l'eliminazione della produzione porta immediatamente ad una grammati equivalente che non genera la stringa vuota.

Esempio. La grammatica regolare

$S \rightarrow bX \mid aB$
$B \rightarrow cX$
$X \rightarrow \varepsilon$

si può modificare in

$S \rightarrow b \mid aB$
$B \rightarrow c$

Esempio. Nella grammatica CF

$S \rightarrow AB \mid aB \mid B$

$A \rightarrow ab \mid aB$

$B \rightarrow cX \mid X$

$X \rightarrow \varepsilon$

la ε-produzione può migrare verso l'assioma

$S \rightarrow AB \mid aB \mid B$

$A \rightarrow ab \mid aB$

$B \rightarrow c \mid \varepsilon$

e successivamente si ottiene

$S \rightarrow AB \mid A \mid aB \mid a \mid B \mid \varepsilon$

$A \rightarrow ab \mid aB \mid a$

$B \rightarrow c$

Nel caso delle grammatiche di tipo 1 invece non si possono ammettere ε-produzioni in posizione arbitraria perchè si altera il potere generativo delle grammatiche.

Esempio. Sia G una grammatica di tipo 0 con una sola regola $\alpha \rightarrow \beta$ con $|\alpha| \geq |\beta|$; supponiamo sia $AB \rightarrow C$, si può costruire una G' di tipo 1 e con ε produzioni equivalente a G.

$G' = \langle V_T, V_N \cup \{H\}, P', S \rangle$

$P' = P - \{AB \rightarrow C\} \cup \{AB \rightarrow CH,\ H \rightarrow \varepsilon\}$

Teorema. Data una grammatica di tipo 0 esiste una grammatica di tipo con ε-produzioni equivalente.

Dim. Sia $G=<V_T,V_N,P,S>$ una grammatica di tipo 0, ricaviamo una grammatica equivalente

$$G'=<V_T,V_N',P',S>$$

con $V_N'=V_N \cup \{X\}$, $X \in V_N$ e P' uguale a P ma con in più la produzione $X \rightarrow \varepsilon$ e al posto di tutte le $\alpha \rightarrow \beta$ con $|\alpha|>|\beta|$, le produzioni $\alpha \rightarrow \beta X....X$ con X ripetuta $|\alpha|-|\beta|$ volte

9.10 RICONOSCIMENTO DI LINGUAGGI

Dato un linguaggio L, il problema del riconoscimento di L è il problema decisionale seguente: data una stringa x, stabilire se essa appartiene ad L. Tale problema è noto anche come problema dell'appartenenza (o membership).

- I linguaggi di tipo 3 sono riconosciuti da dispositivi con memoria costante in tempo lineare (automi a stati finiti).

- I linguaggi di tipo 2 sono riconosciuti da dispositivi nondeterministi con pila in tempo lineare (automi a pila non deterministici).

- I linguaggi di tipo 1 sono riconosciuti da dispositivi nondeterministi con memoria che cresce linearmente con la lunghezza della stringa da esaminare (automi non deterministici "linear bounded").

- Per alcuni linguaggi strettamente di tipo 0 è possibile che non esista un algoritmo di decisione ma esiste comunque un processo semidecisionale, in cui, se la stringa fa parte del linguaggio essa viene riconosciuta ma se non fa parte del linguaggio non è detto che la computazione termini. I dispositivi che consentono di riconoscere o di attuare un procedimento di semidecisione per i linguaggi di tipo 0 sono macchine di Turing. In generale non è possibile stabilire un limite alle quantità di risorsa tempo o memoria che si rende necessario per riconoscere un linguaggio di tipo 0.