Trasparenze del corso di

Informatica Teorica

Parte Seconda

Linguaggi e Grammatiche

1

corrispondenza tra
linguaggi e problemi

Linguaggi e informatica

- ubiquitari nelle applicazioni
  - compilatori ed interpreti
  - protocolli
  - sequenze di operazioni
  - interfacce
- paradigmatici nella teoria
  - molti importanti problemi teorici sono riconducibili a quello dell'appartenenza di una stringa ad un linguaggio

2

Tre approcci diversi

tra loro complementari

non molto utile all'atto pratico

- approccio insiemistico

  come un frasario
  - utile per determinare le proprietà elementari dei linguaggi
- approccio generativo
  - grammatiche formali

    regole per generare le stringhe
- approccio riconoscitivo
  - automi riconoscitori

    meccanismi che riconoscono le stringhe di un linguaggio

3

Alfabeto

- un *alfabeto* è un insieme finito non vuoto di simboli (caratteri)
- esempi:

  {'M', 'C', 'L', 'X', 'I', 'V'}

  {'0', '1'}

  {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}

  {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'z'}

4

Concatenazione

- dato un alfabeto Σ, definiamo l'operazione binaria *concatenazione* (denotata con "∘")

  $\mathtt{a} \circ \mathtt{b} = \mathtt{ab}$ con $\mathtt{a}, \mathtt{b} \in \Sigma$
- indichiamo con $\mathtt{a}^n$ la concatenazione di **a** con se stessa n volte

  esempio: $\mathtt{a}^4 = \mathtt{a} \circ \mathtt{a} \circ \mathtt{a} \circ \mathtt{a} = \mathtt{aaaa}$
- l'operazione "∘" è associativa ma non commutativa
- dati Σ e "∘" definiamo Σ^+ come l'insieme delle stringhe (parole) di lunghezza finita
- se a Σ^+ aggiungiamo la stringa vuota ε = "" otteniamo Σ^*

5

Linguaggio

- un *linguaggio* è un sottoinsieme di Σ^*
- Σ^* è detto linguaggio universale
- il linguaggio vuoto Λ non contiene stringhe

  (nota che Λ coincide con l'insieme vuoto ∅)
- $\Lambda \neq \{\varepsilon\}$

6

Operazioni sui linguaggi

L_1 e L_2 linguaggi

- *unione*

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

  $L_1 \cup \Lambda = L_1$

tutte le stringhe di L1 + le stringhe di L2

- *intersezione*

  $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \wedge x \in L_2\}$

  $L_1 \cap \Lambda = \Lambda$

tutte le stringhe che appartengono a entrambi i linguaggi L1 ed L2

- *complementazione*

  $\underline{L}_1 = \{x \in \Sigma^* \mid x \notin L_1\}$

tutte le stringhe che non appartengono a L1

7

Operazioni sui linguaggi

L_1 e L_2 linguaggi

- *concatenazione* o *prodotto*

  $L_1 \circ L_2 = \{x \in \Sigma^* \mid$
  $\exists x_1 \in L_1 \wedge \exists x_2 \in L_2$ tali che $x = x_1 \circ x_2\}$

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

L1={a,b}, L2={b,c}
L1°L2={ac,ad,bc,bd,a,b}

  esempio:

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

- *potenza* L^h di un linguaggio L

  $L^0 = \{\varepsilon\}$

  $L^h = L \circ L^{h-1}$, per $h \geq 1$

L={a,b}, L^1=L°L^(1-1)
=L°L^0={a,b}

8

Operatore di Kleene

- *chiusura riflessiva e transitiva* di un linguaggio

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

$\varepsilon \in L^*$ $\Lambda^* = \{\varepsilon\}$

esempio: $L=\{aa\}$ $L^*=\{a^{2n} \mid n \geq 0\}$

- *chiusura transitiva* (non riflessiva) di un linguaggio

$L^+ = \bigcup_{h=1}^{\infty} L^h$

esempio: $L=\{aa\}$ $L^+=\{a^{2n} \mid n \geq 1\}$

$L^*=L^+ \cup \{\varepsilon\}$

9

Espressioni regolari

- è uno strumento per descrivere linguaggi (vedremo poi quali)
- dato un alfabeto Σ, si definisce *espressione regolare* ogni stringa r

$r \in (\Sigma \cup \{+, *, (,), \circ, \varnothing\})^+$ chiusura transitiva

- tale che:
  1. $r=\varnothing$ oppure
  2. $r \in \Sigma$ oppure
  3. $r=(s+t)$ oppure $r=(s \circ t)$ oppure $r=s^*$, con s e t espressioni regolari

semantica

espressione	linguaggio
$\varnothing$	Λ
$a \in \Sigma$	$\{a\}$
$(s+t)$	$L(s) \cup L(t)$
$(s \circ t)$	$L(s) \circ L(t)$
s^*	$L(s)^*$

10

un'espressione regolare può essere l'unione, la concatenazione o la chiusura transitiva e riflessiva di due espressioni regolari

(ab+cd)*={epsilon,ab,cd,abcd,abab,cdcd,cdab.....}
in pratica tutte le combinazioni possibili

Espressioni regolari

esempio:
(**a**+**b**)* rappresenta L=({**a**}∪{**b**})*

esempio:
(**a**+**b**)***a** rappresenta L={x|x∈{**a**,**b**}* ∧ "x termina con **a**"}

forma sintetica
st è forma sintetica di **s**∘**t**

forma sintetica
espressioni sintetiche si ottengno definendo delle precedenze tra gli operatori: * > ∘ > +

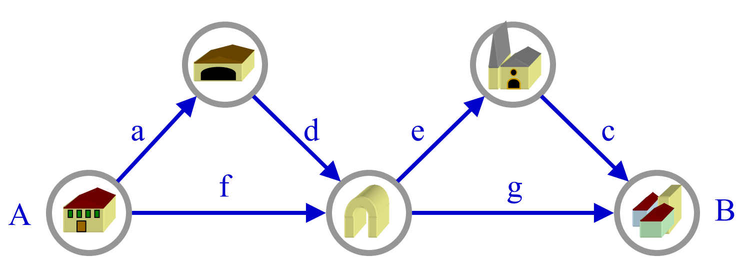
esempio:
(**a**+(**b**(**cd**))) = **a**+**bcd**

i linguaggi rappresentabili con espressioni regolari sono una interessante sottoclasse

non si possono quindi rappresentare tutti i linguaggi

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B



adec+**adg**+**fec**+**fg** =

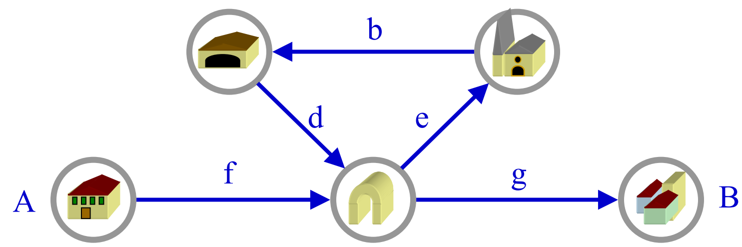
= **ad**(**ec**+**g**) + **f**(**ec**+**g**) =

= (**ad**+**f**)(**ec**+**g**)

12

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B



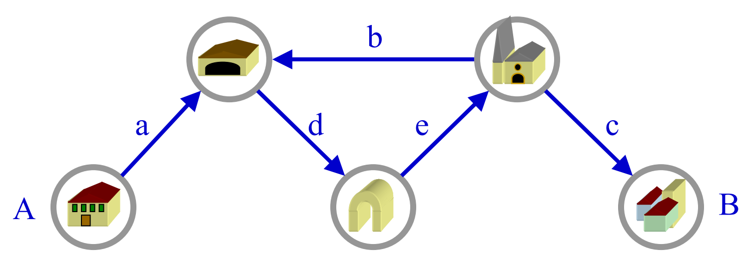
$f(ebd)^*g$

ebd lo faccio 0 o più volte

13

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B

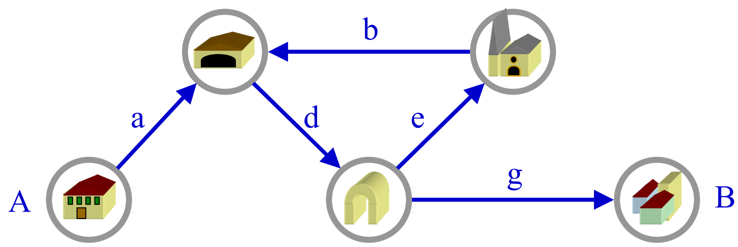


$a(deb)^*dec$ =ad(ebd)*ec

14

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B

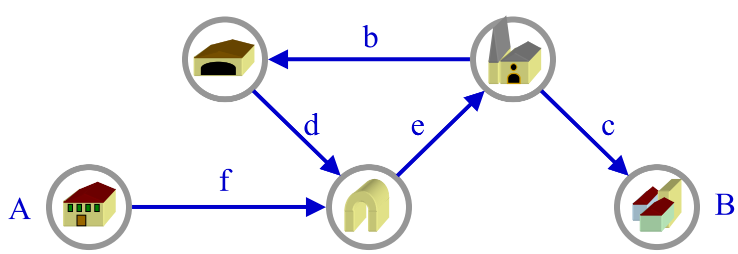


$a(deb)^*dg$ =ad(deb)*g

15

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B

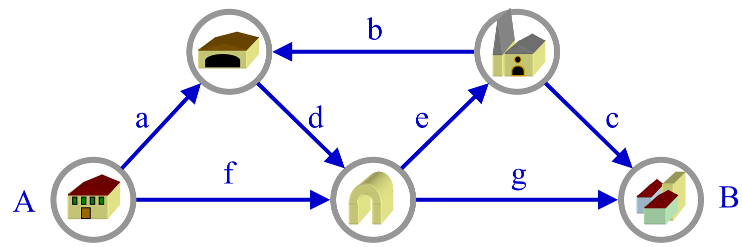


$f(edb)^*ec$

16

Esercizio

data una mappa stradale, scrivere un'espressione regolare che definisca tutti i percorsi possibili tra A e B



$$\mathtt{f(ebd)^*g} + \mathtt{f(ebd)^*ec} + \mathtt{a(deb)^*dec} + \mathtt{a(deb)^*dg} =$$
$$= \mathtt{f(ebd)^*(g+ec)} + \mathtt{a(deb)^*d(g+ec)} =$$
$$= (\mathtt{f(ebd)^*+a(deb)^*d})\ (\mathtt{g+ec}) =$$
$$= (\mathtt{f+ad})(\mathtt{ebd})^*(\mathtt{g+ec})$$

17

Cardinalità dei linguaggi

- dato un alfabeto Σ è possibile definire un ordinamento "lessicografico" per le stringhe di Σ^*
- ciò implica l'esistenza di un algoritmo di enumerazione per Σ^*
- quindi $|\Sigma^*| = \aleph_0$
- un linguaggio definito su Σ è un sottoinsieme di Σ^*
- l'insieme di tutti i linguaggi definiti su Σ è equinumeroso a $P(\Sigma^*)$ e quindi ha cardinalità $2^{\aleph_0}$
- ne segue che l'insieme di tutti i linguaggi definiti su Σ non è numerabile

18

Cardinalità dei programmi

l'insieme dei programmi scritti in Java (per esempio) ha cardinalità $\aleph_0$, infatti posso facilmente enumerarli

problema:

dato un qualunque $L \subseteq \Sigma^*$ esiste sempre un programma scritto in Java che, data una stringa x di Σ^*, decida se x appartiene ad L?

soluzione:

semplici considerazioni sulla cardinalità evidenziano una risposta negativa

ci sono più problemi che soluzioni

ci sono più problemi che programmi che li sappiano risolvere

19

Le grammatiche formali

- approccio generativo alla descrizione dei linguaggi
- metodo di costruzione delle stringhe basato sulla riscrittura

1940 Post e Markof, riscrittura e derivazione di stringhe

1950 Chomsky, classificazione delle grammatiche nell'ambito degli studi sul linguaggio naturale

1960 Backus Naur Form per descrivere Algol

20

Algol è un linguaggio di programmazione

Grammatiche formali

- grammatiche di Chomsky
- ε-produzioni
- riconoscimento di linguaggi

21

Grammatiche di Chomsky

Vt = caratteri delle stringhe

Vn = verbi, soggetti etc.

una *grammatica* è una quadrupla

$$G=\langle V_T, V_N, P, S\rangle$$

- $V_T \subseteq \Sigma$ è l'alfabeto (finito) di *simboli terminali*
- V_N è un insieme (finito) di *simboli non terminali*, o *categorie sintattiche*, tale che $V_N \cap \Sigma = \varnothing$
- P, detto insieme delle *produzioni*, è una relazione binaria finita su

$$(V_T \cup V_N)^* \circ V_N \circ (V_T \cup V_N)^* \times (V_T \cup V_N)^*$$

deve contenere almeno un non terminale in una posizione qualsiasi

forma sintetica

$\langle\alpha,\beta\rangle \in P$ si indica generalmente con $\alpha \rightarrow \beta$

le P sono regole di riscrittura. si passa da a->b

- $S \in V_N$ è *l'assioma*

22

Esempio

una grammatica definisce implicitamente tutte le stringhe di terminali generabili a partire dall'assioma tramite una sequenza di riscritture

esempio:

G=<{a, b, c}, {S, B, C}, P, S>, con P composto da:

❶ $S \to aS$ ❷ $S \to B$ ❸ $B \to bB$

❹ $B \to bC$ ❺ $C \to cC$ ❻ $C \to c$

genera $L(G) = \{a^n b^m c^h \mid n \geq 0,\ m, h \geq 1\}$

forma sintetica

$\alpha \to \beta_1 \quad \alpha \to \beta_2 \quad \ldots \quad \alpha \to \beta_n$

viene anche indicato con

$\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

23

Linguaggio generato

forma sintetica

$V_T \cup V_N$ viene talvolta indicato con V

- *derivazione diretta*: relazione su

$$(V^* \circ V_N \circ V^*) \times V^*$$

$<\varphi,\psi>$ appartiene alla relazione (si scrive $\varphi \Rightarrow \psi$) se $\exists \alpha \in V^* \circ V_N \circ V^*$ ed $\exists \beta,\gamma,\delta \in V^*$ t.c. $\varphi = \gamma\alpha\delta$ $\psi = \gamma\beta\delta$ e $\alpha \to \beta \in P$

φ e ψ sono dette *forme di frase*

- *derivazione*: chiusura riflessiva e transitiva della derivazione diretta, si rappresenta con $\Rightarrow^*$
- il *linguaggio generato* da G è $L(G) = \{x \mid x \in V_T^* \wedge S \Rightarrow^* x\}$
- due *grammatiche* G_1 e G_2 sono *equivalenti* se $L(G_1) = L(G_2)$

forma sintetica

talvolta $\Rightarrow$ al posto di $\Rightarrow^*$

24

Grammatiche formali

esempio: generazione di $\{a^n b^n c^n | n \geq 1\}$

grammatica G=<{a, b, c},{S,B,C,F,G},P,S>

con P composto da

❶ $S \rightarrow aSBC$ ❷ $CB \rightarrow BC$

❸ $SB \rightarrow bF$ ❹ $FB \rightarrow bF$

❺ $FC \rightarrow cG$ ❻ $GC \rightarrow cG$

❼ $G \rightarrow \varepsilon$

per generare aabbcc

$S \Rightarrow^{1} aSBC \Rightarrow^{1} aaSBCBC \Rightarrow^{2} aaSBBCC$

$\Rightarrow^{3} aabFBCC \Rightarrow^{4} aabbFCC \Rightarrow^{5} aabbcGC$

$\Rightarrow^{6} aabbccG \Rightarrow^{7} aabbcc$

25

Grammatiche formali

osservazione: non è detto che una sequenza di derivazioni porti ad una stringa del linguaggio

esempio:

la grammatica G=<{a, b, c}, {S, A}, P, S> con

$S \rightarrow aSc \mid A$

$A \rightarrow bAc \mid \varepsilon$

genera $\{a^n b^m c^{n+m} | n,m \geq 0\}$

esempio:

la grammatica G=<{a, b, c}, {S, A}, P, S> con

$S \rightarrow Ab$

$A \rightarrow Sa$

genera Λ

26

Grammatiche di Chomsky

- di tipo 0, non limitate
- di tipo 1, context sensitive, contestuali
- di tipo 2, context free (CF), non contestuali
- di tipo 3, lineari destre (RL), regolari

27

Grammatiche di Chomsky di tipo 0, non limitate

- sono le meno restrittive
- produzioni del tipo

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

ammettono anche derivazioni che accorciano stringhe

linguaggi di tipo 0

esempio:

il linguaggio $\{a^n b^n | n \geq 1\}$ è di tipo 0 in quanto generato da

$S \rightarrow aAB \qquad B \rightarrow b$

$aA \rightarrow aaAb \qquad aAb \rightarrow ab$

$aAA \rightarrow aA$

28

Grammatiche di Chomsky di tipo 1, context sensitive, contestuali

- produzioni che non riducano la lunghezza delle forme di frase

$$\alpha \longrightarrow \beta,\ |\alpha| \le |\beta|,\ \alpha \in V^* \circ V_N \circ V^*,\ \beta \in V^*$$

linguaggi di tipo 1

esempio:

il linguaggio $\{a^n b^n c^n | n \ge 1\}$ è di tipo 0 in quanto generato da

$S \rightarrow aSBC \qquad CB \rightarrow BC$

$SB \rightarrow bF \qquad FB \rightarrow bF$

$FC \rightarrow cG \qquad GC \rightarrow cG$

$cG \rightarrow c$

ma è anche di tipo 1, infatti è generato anche da

$S \rightarrow aSBc \mid aBc$

$cB \rightarrow Bc$

$bB \rightarrow bb$

$aB \rightarrow ab$

29

Generazione di stringhe di $a^n b^n c^n$

(1) $S \longrightarrow aSBc \mid aBc$ (2) $cB \longrightarrow Bc$

(3) $bB \longrightarrow bb$ (4) $aB \longrightarrow ab$

$S \Rightarrow aSBc$
$\Rightarrow aaSBcBc$
$\Rightarrow aaaSBcBcBc$
$\Rightarrow aaaaBcBcBcBc$
$\Rightarrow aaaaBcBcBBcc$
$\Rightarrow aaaaBcBBcBcc$
$\Rightarrow aaaaBcBBBccc$
$\Rightarrow aaaaBBcBBccc$
$\Rightarrow aaaaBBBcBccc$
$\Rightarrow aaaaBBBBcccc$
$\Rightarrow aaaabBBBcccc$
$\Rightarrow aaaabbBBcccc$
$\Rightarrow aaaabbbBcccc$
$\Rightarrow aaaabbbbcccc$

30

Grammatiche di Chomsky di tipo 2, context free (CF), non contestuali

- produzioni del tipo

$$A \to \gamma,\ A \in V_N,\ \gamma \in V^+$$

linguaggi di tipo 2

esempio:

il linguaggio $\{\mathtt{a}^n\mathtt{b}^n | n \geq 1\}$ è di tipo 0 in quanto generato da

$S \to \mathtt{a}A\mathtt{b}$

$\mathtt{a}A \to \mathtt{aa}A\mathtt{b}$

$A \to \varepsilon$

ma è anche di tipo 2, infatti è generato anche da

$S \to \mathtt{a}S\mathtt{b} \mid \mathtt{ab}$

31

Esempi di linguaggi di tipo 2

linguaggio delle espressioni aritmetiche con la variabile i (come per esempio "`i*i+(i*i+(i))*i*i`", oppure "`((i+i)*i)`").
L'assioma è E.

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to \mathtt{i} \mid (E)$

grammatica delle parentesi ben bilanciate (esempio "`((()))()`")

$S \to () \mid SS \mid (S)$

da quale sequenza di produzioni è generata "`()((()()))`" ?

grammatica delle stringhe palindrome (esempio "elle", "ereggere")

32

Grammatiche di Chomsky di tipo 3, lineari destre (RL), regolari

- produzioni del tipo

$$A \rightarrow \delta,\ A \in V_N,\ \delta \in (V_T \circ V_N) \cup V_T$$

- linguaggi di tipo 3

esempio:

il linguaggio $\{a^n b | n \geq 0\}$ è di tipo 3 in quanto generato da

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

si possono anche definire grammatiche lineari sinistre (LL) con

$$A \rightarrow \delta,\ A \in V_N,\ \delta \in (V_N \circ V_T) \cup V_T$$

esempio: il linguaggio $\{a^n b | n \geq 0\}$ è anche generato da

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

teorema: i linguaggi generati da grammatiche LL e RL coincidono

33

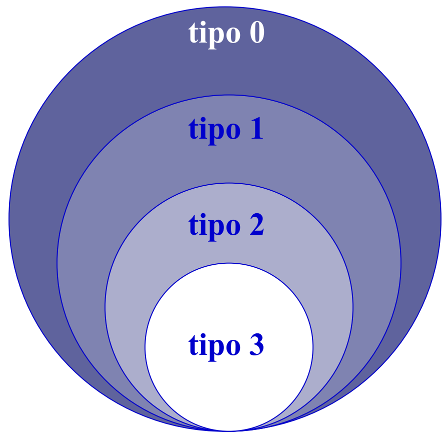
Grammatiche di Chomsky

un linguaggio è *strettamente di tipo n* se esiste una grammatica di tipo n che lo genera e non esiste una grammatica di tipo m>n che lo genera

esempio: il linguaggio $\{a^n b^n | n \geq 1\}$ è generato da una grammatica di tipo 2 e non è generato da nessuna grammatica di tipo 3

34

Grammatiche di Chomsky



contenimento tra i linguaggi

35

Grammatiche di Chomsky

tipo	produzioni	vincoli
tipo 0 non limitate	$\alpha \rightarrow \beta$	$\alpha \in V^* \circ V_N \circ V^*$, $\beta \in V^*$
tipo 1 contestuali	$\alpha \rightarrow \beta$	$\|\alpha\| \leq \|\beta\|$ $\alpha \in V^* \circ V_N \circ V^*$, $\beta \in V^*$
tipo 2 non contestuali	$A \rightarrow \gamma$	$A \in V_N$, $\gamma \in V^+$
tipo 3 regolari	$A \rightarrow \delta$	$A \in V_N$, $\delta \in (V_T \circ V_N) \cup V_T$

quadro riassuntivo della classificazione delle grammatiche secondo Chomsky

36

ε-produzioni

- con grammatiche di tipo 1, 2, 3 non è possibile generare ε
  - per generare ε occorre una produzione $\alpha\rightarrow\varepsilon$ che viene detta ε-produzione
  - per Chomsky tutti i linguaggi che contengono ε-produzioni sono linguaggi di tipo 0
- qual'è l'impatto sui corrispondenti linguaggi delle ε-produzioni nelle grammatiche?

37

ε-produzioni in posizione arbitraria

esempio:

G è una grammatica di tipo 0 con una sola produzione $\alpha\rightarrow\beta$ con $|\alpha|\geq|\beta|$; supponiamo sia $AB \rightarrow C$, si può costruire una G′ di tipo 1 e con ε-produzioni equivalente a G

$$\begin{cases} G'=<V_T,V_N\cup\{H\},P',S> \\ P'= P-\{AB \rightarrow C\}\cup\{AB \rightarrow CH, H \rightarrow \varepsilon\} \end{cases}$$

teorema:

data una grammatica di tipo 0 esiste una grammatica di tipo 1 con ε-produzioni equivalente

dimostrazione:

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 $V_N'=V_N\cup\{X\}$, $X\notin V_N$

a P aggiungiamo $X\rightarrow\varepsilon$ e modifichiamo tutte le $\alpha\rightarrow\beta$ con $|\alpha|\geq|\beta|$ in $\alpha\rightarrow\beta X\ldots X$ con X ripetuta $|\alpha|-|\beta|$ volte

38

ε-produzioni

- teorema
  ad ogni grammatica $G=<V_T,V_N,P,S>$ di tipo 1, 2 o 3 corrisponde una grammatica $G'=<V_T,V_N \cup \{S'\},P',S'>$ tale che
  – $L(G') = L(G) \cup \{\varepsilon\}$
  – G' è dello stesso tipo (1, 2, o 3) di G a meno di ε-produzioni sull'assioma
- dimostrazione
  – sia $G=<V_T,V_N,P,S>$ la grammatica che genera L
  – definiamo $G'=<V_T,V_N \cup \{S'\},P',S'>$ che genera $L'=L \cup \{\varepsilon\}$
  – se G è di tipo 1 o 2, $P' = P \cup \{S' \rightarrow \varepsilon, S' \rightarrow S\}$
  – se G è di tipo 3, $P' = P \cup \{S' \rightarrow \varepsilon\} \cup \{S' \rightarrow \alpha |$ per ogni $S \rightarrow \alpha\}$

- è facile dimostrare che, se l'assioma non compare mai a destra di qualche produzione, vale anche la corrispondenza opposta

39

ε-produzioni e grammatiche regolari

le grammatiche di tipo 3 (right linear, RL, lineari destre, regolari) non modificano la loro natura se aggiungiamo ε-produzioni

esempio: la grammatica RL

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

si può modificare in

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

40

ε-produzioni e grammatiche regolari

teorema:

se L è il linguaggio definito da una grammatica regolare a cui sono state aggiunte ε-produzioni, il linguaggio L'= L-{ε} è regolare

dimostrazione:

aggiungiamo una ε-produzione $E \to \varepsilon$ ad una grammatica regolare

- supponiamo, per semplicità, che l'assioma S non appaia mai a destra di una produzione, se E=S il teorema è dimostrato
- supponiamo che E compaia a destra delle $k \geq 0$ produzioni $A_i \to \alpha_i$
- eliminiamo la produzione $E \to \varepsilon$ e aggiungiamo k produzioni $A_i \to \alpha_i'$ dove α_i' è ricavato da α_i sostituendo ε alle E

41

ε-produzioni e grammatiche CF

le grammatiche di tipo 2 (context free, CF, non contestuali) non modificano la loro natura se aggiungiamo ε-produzioni

esempio: nella grammatica CF

$S \to AB \mid aB \mid B$
$A \to ab \mid aB$
$B \to cX \mid X$
$X \to \varepsilon$

la ε-produzione può "migrare" verso l'assioma

$S \to AB \mid aB \mid B$
$A \to ab \mid aB$
$B \to c \mid \varepsilon$

nuovamente

$S \to AB \mid A \mid aB \mid a \mid B \mid \varepsilon$
$A \to ab \mid aB \mid a$
$B \to c$

42

ε-produzioni e grammatiche CF

teorema:

se L è il linguaggio definito da una grammatica CF a cui sono state aggiunte ε-produzioni, il linguaggio L'= L-{ε} è CF

dimostrazione:

aggiungiamo una ε-produzione $E \rightarrow \varepsilon$ ad una grammatica CF

- supponiamo, per semplicità, 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 $k \geq 0$ produzioni $A_i \rightarrow \alpha_i$
- eliminiamo la produzione $E \rightarrow \varepsilon$ e aggiungiamo k produzioni $A_i \rightarrow \alpha_i'$ dove α_i' è ricavato da α_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

43

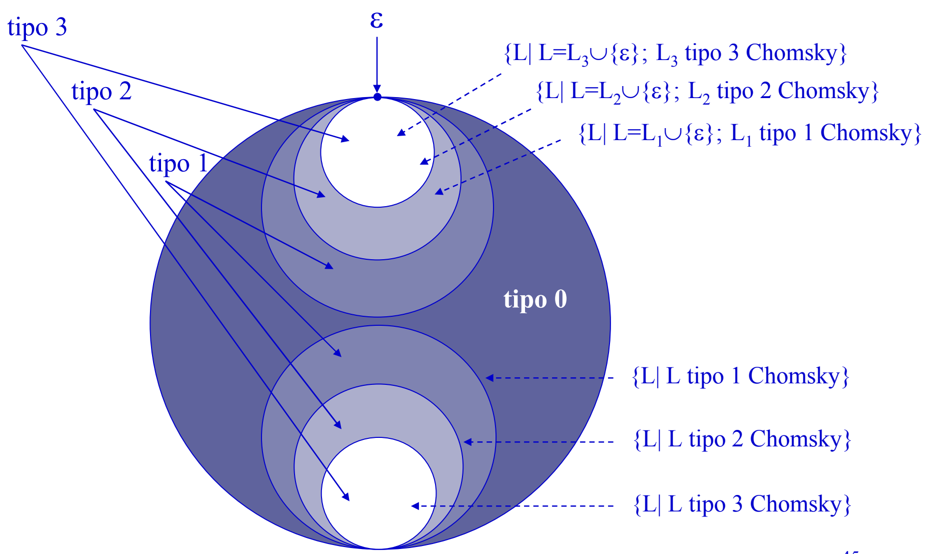
ε-produzioni: ricapitolazione

coerentemente a quanto stabilito dai teoremi precedenti nel seguito ammetteremo, nelle diverse grammatiche, le seguenti ε-produzioni:

tipo	ε-produzioni ammesse
0	tutte (per definizione)
1	solo sull'assioma quando quest'ultimo non compare mai a destra di una produzione
2	tutte
3	tutte

44

Grammatiche con ε-produzioni



45

Esempi di grammatiche

- il linguaggio $\{w \circ w \mid w \in (a+b)^*\}$
- è generato dalla grammatica contestuale

1	2	3	4
$S \to T \mid \varepsilon$	$Aa \to aA$	$AA_0 \to A_0a$	$A_0 \to a$
$T \to aAT \mid bBT \mid A_0a \mid B_0b$	$Ab \to bA$	$BA_0 \to A_0b$	$B_0 \to b$
	$Ba \to aB$	$AB_0 \to B_0a$	
	$Bb \to bB$	$BB_0 \to B_0b$	

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

le 2 e le 3 separano la prima stringa dalla seconda; le 4 chiudono la generazione, se sono applicate troppo presto il processo diverge

46

Esempi di grammatiche

- il linguaggio $\{(x\#)^* \mid x = \text{permutazione di } (a,b,c)\}$ (che contiene per esempio le stringhe ε, abc#, acb#, bac#cab# , ecc)

  è generato dalla grammatica contestuale:

  $S \rightarrow S' \mid \varepsilon$ $\quad AB \rightarrow BA$ $\quad A \rightarrow a$
  $S' \rightarrow ABC\#$ $\quad AC \rightarrow CA$ $\quad B \rightarrow b$
  $S' \rightarrow ABC\#S'$ $\quad BC \rightarrow CB$ $\quad C \rightarrow c$

  ma è generato anche dalla grammatica CF:

  $S \rightarrow E\#S \mid \varepsilon \qquad E \rightarrow abc \mid acb \mid cba \mid cab \mid bac \mid bca$

  ed anche dalla grammatica regolare:

  $S \rightarrow aX \mid bY \mid cZ \mid \varepsilon \qquad R \rightarrow \#S$

  $X \rightarrow bX' \mid cX''$ $\quad Y \rightarrow aY' \mid cY''$ $\quad Z \rightarrow aZ' \mid bZ''$
  $X' \rightarrow cR$ $\quad Y' \rightarrow cR$ $\quad Z' \rightarrow bR$
  $X'' \rightarrow bR$ $\quad Y'' \rightarrow aR$ $\quad Z'' \rightarrow aR$

47

Linguaggi lineari

- esistono classificazioni dei linguaggi alternative a quella di Chomsky
- una grammatica è *lineare* quando la parte sinistra di ogni produzione è costituita da *esattamente un* non terminale e la parte destra contiene *al più un* non terminale
- le grammatiche di tipo 3 sono lineari
- esistono grammatiche CF lineari

esempio:

$S \rightarrow aSb$

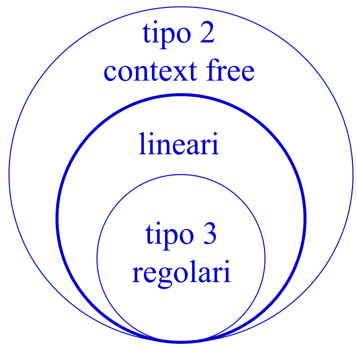
$S \rightarrow c$

che genera $\{a^n c b^n \mid n \geq 0\}$

- i linguaggi strettamente lineari sono un sottoinsieme di quelli strettamente CF e un soprainsieme di quelli regolari

48

Grammatiche lineari e grammatiche di Chomsky



contenimento tra i linguaggi

49

Forma normale di Backus

- la BNF è una notazione CF con alcuni accorgimenti sintattici che ne aumentano la leggibilità

esempio

```
<sequenza istruzioni>::= <istruzione>;
     {<istruzione>;}
```

può essere riscritto:

$$Q \rightarrow I\,\texttt{;} \mid I\,\texttt{;}\,Q$$

```
<istruzione if> ::= if ( <condizione> )
     <istruzione> [else <istruzione>]
```

può essere riscritto:

$$F \rightarrow \texttt{if (} C \texttt{) } I \texttt{ else } I \mid \texttt{if (} C \texttt{) } I$$

50

Riconoscimento dei linguaggi

problema:

stabilire se una stringa appartiene ad un dato linguaggio

- esistono linguaggi a cui non corrisponde alcun algoritmo di decisione
- i linguaggi di tipo 3 sono riconosciuti da dispositivi con memoria costante in tempo lineare (automi a stati finiti)
- i linguaggi strettamente di tipo 2 sono riconosciuti da dispositivi non deterministici con pila in tempo lineare (automi a pila non deterministici)

51

Riconoscimento dei linguaggi

- i linguaggi strettamente di tipo 1 sono riconosciuti da dispositivi non deterministici con memoria che cresce linearmente con la lunghezza della stringa da esaminare (automi non deterministici “linear bounded”)
- i linguaggi strettamente di tipo 0 sono riconosciuti da macchine di Turing con memoria e tempo illimitati
- è possibile che non esista un algoritmo di decisione ma un processo semidecisionale, in cui, se la stringa non fa parte del linguaggio non è detto che la computazione termini

52