

Definizioni :

$$D = \left\{ x \in \{0,1\}^* \mid F_u(x \$ x) \downarrow \right\}$$

Linguaggio dell' Arresto Ristretto

$$D^c = \left\{ x \in \{0,1\}^* \mid F_u(x \$ x) \uparrow \right\}$$

Teo:

- 1) D è RICORSIVAMENTE ENUMERABILE
- 2) D non è RICORSIVO
- 3) D^c non è RICORSIVAMENTE ENUMERABILE

ohimostrazione:

1) Devo esibire per D una procedura p.t.c.

$$F_p(x) = \begin{cases} 1 & \text{se } x \in D \Rightarrow F_u(x|x) \downarrow \\ 0 & \text{se } x \notin D \Rightarrow F_u(x|x) \uparrow \end{cases}$$

Penso costruire la seguente:

Procedura RICNUM ($x \in \{0,1\}^*$)

$$\left\{ \begin{array}{l} y = F_u(x|x); \end{array} \right.$$

return (1);

}

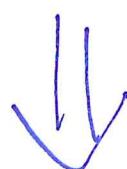
dim. di correttezza di RICNUM :

$x \in D$ $\Rightarrow \bar{F}_U(x \$ x) \downarrow \Rightarrow$ viene eseguito
l'assegnamento
 $y = \dots$

\Rightarrow viene eseguito return(1) $\Rightarrow \underline{\text{RICNUM}(x) = 1}$

$x \notin D$ $\Rightarrow \bar{F}_U(x \$ x) \uparrow \Rightarrow$ il programma va
in loop
all'istruzione $y = \dots$

$\Rightarrow \underline{\text{RICNUM}(x) \uparrow}$



RICNUM è la procedura P cercata



D è RICORS. ENUMERABILE

2) D non è ricorsivo

Si dimostra per assurbo.

Supponendo che D sia ricorsivo

posso costruire il seguente programma:

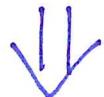
Procedura ASSURDOA ($x \in \{0,1\}^*$)

{
 if $x \in D$ then return ($1 - F_u(x|x)$)

 else return (0);

}

Tale procedura è implementabile
in un linguaggio di programmazione
accettabile a patto che D sia RICORSIVO.



esiste il codice "e"

e = coolifica del programma

ASSURDO A

Adesso passo in input ad

ASSURDO A la parola "e"

Cosa succede? ASSURDDA(e) = ?

→ Secondo alcune considerazioni ottengo:

$$\text{ASSURDOA}(e) = \bar{F}_e(e) = \bar{F}_u(e \& e)$$

↑

↑

per definizione
di e

per definizione
di u

→ in base a come è fatto il codice
di ASSURDOA ottengo:

$$\underline{\text{caso}} \quad e \in D \Rightarrow F_U(e \notin e) \downarrow \begin{cases} 1 \\ 0 \end{cases}$$

↑
per def. di D

$$F_U(e \notin e) = \text{ASSURDOA}(e) = 1 - F_U(e \in e)$$

↑ ↑
per (#) per def. di ASSURDOA

nel caso $F_U(e \notin e) = 1$

$$1 = 0$$

nel caso $F_U(e \notin e) = 0$ \Downarrow Contradizione

$$0 = 1$$

caso $e \notin D \Rightarrow F_u(e \notin e) \uparrow$

↑
per def. di D

$F_u(e \notin e) = \text{ASSURDOA}(e) = 0$

↑
per (*)

↑
per def. di ASSURDOA

si ottiene

$\uparrow = 0 \quad \Downarrow$

contradizione

\Downarrow

" e " non esiste

dunque D non può essere
RICORSIVO

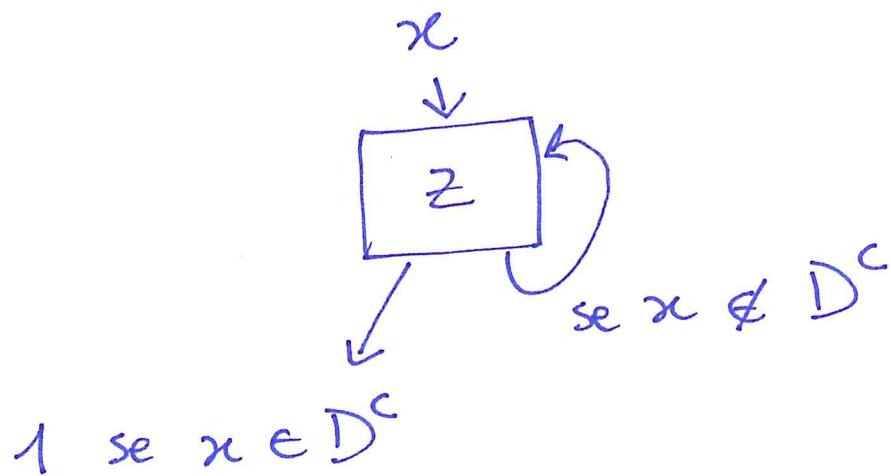
$$D^c = \{ x \in \{0,1\}^* \mid F_\mu(x|x) \uparrow \}$$

3) D^c non è RICORSIV. ENUMERABILE

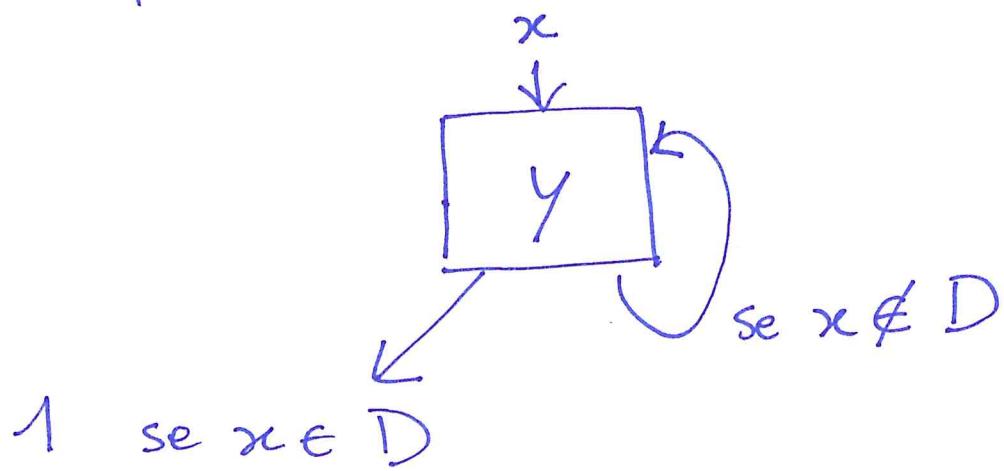
Tecnica per assurdo.

Suppongo che D^c sia ricorsiv. enumerabile

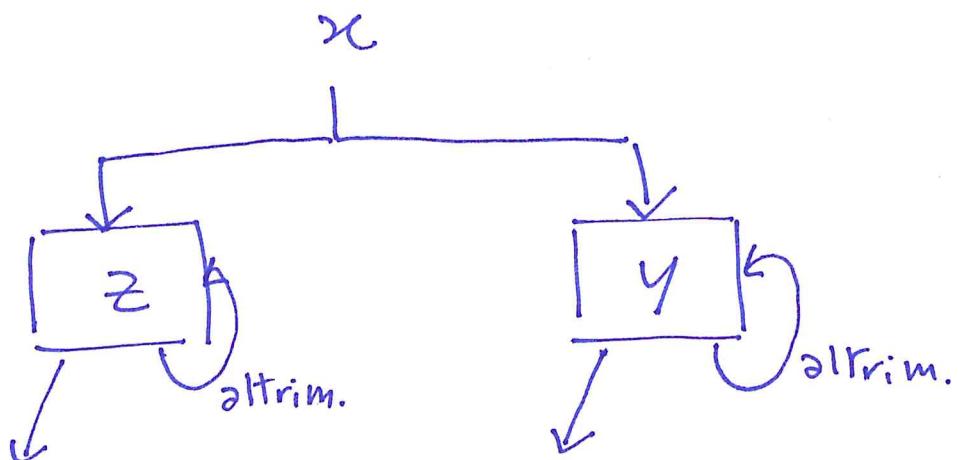
\Rightarrow esiste una procedura \geq t.c.



Inoltre D è RICORSIV. ENUM. (punto 1)
e pertanto esiste y t.c.



Ideas:



1 se $x \in D^c$

||

se $x \notin D$



0

Programma ASSURDOB ($x \in \{0,1\}^*$)

```
{ K=1; // numero di passi
```

```
while ( z(x) non Termina in K passi
```

Λ

```
      y(x) non Termina in K passi )
```

```
{ K=K+1; }
```

if (ha terminato y) then return (1)
else return (0);

}

cosa fa ASSURDOB(x)?

$\underline{x \in D} \Rightarrow$ esiste K t.c. $y(x)$ termina
 \Rightarrow $\underline{\underline{\underline{\underline{\underline{ASSURDOB}(x) = 1}}}}$

$\underline{x \notin D} \Rightarrow$ esiste K t.c. $z(x)$ termina
 \Rightarrow $\underline{\underline{\underline{\underline{\underline{ASSURDOB}(x) = 0}}}}$



ASSURDOB è un algoritmo per D



contradizione perché D non è ricorsivo
(punto 2) del Teo)



z non esiste e D^C non è RIC.ENUM.

Domanda:

Esiste un problema che non ammette soluzione algoritmica?

||

Esiste un problema indecidibile?

||

Esiste un linguaggio non ricorsivo?

Problema dell'arresto = Problema della fermata

Input: $w\$x \in \{0,1\}^*$

Output: $F_w(x) \downarrow ?$

$F_w(w\$x) \downarrow$

$A = \left\{ w\$x \in \{0,1\}^* \mid \overbrace{F_w(x) \downarrow} \right\}$

linguaggio dell'ARRESTO

$$A = \{ w \$ x \in \{0,1\}^* \mid F_u(w \$ x) \downarrow \}$$

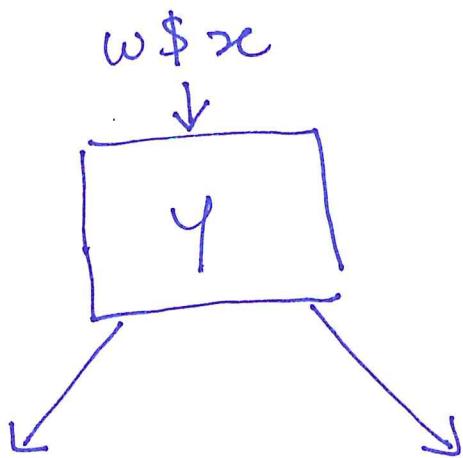
$$D = \{ x \in \{0,1\}^* \mid F_u(x \$ x) \downarrow \}$$

Teo: A non è RICORSIVO.

olim. uso il fatto che D non è RICORSIVO.

Tecnica per ASSURDO.

Suppongo che A sia RICORSIVO:



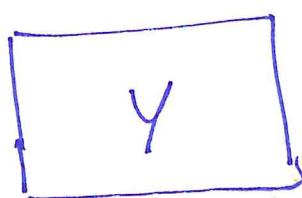
1 se $F_u(w \$ x) \downarrow$ 0 se $F_u(w \$ x) \uparrow$

Allora posso:

$$x \in \{0,1\}^*$$



$$x \$ x$$



1 se $x \in F_u(x \$ x) \downarrow$

"

0 se $F_u(x \$ x) \uparrow$

"

1 se $x \in D$

0 se $x \notin D$

Programma ASSURDOC ($x \in \{0,1\}^*$)

```
{ if  $x \$ x \in A$  then return (1)  
    else return (0);
```

}

Cosa fa ASSURDOC(x)?

$x \in D$ $\Rightarrow F_\mu(x \$ x) \downarrow \Rightarrow x \$ x \in A$

$\Rightarrow \underbrace{\text{ASSURDOC}(x)}_{=} = 1$

$x \notin D$ $\Rightarrow F_\mu(x \$ x) \uparrow \Rightarrow x \$ x \notin A$

$\Rightarrow \underbrace{\text{ASSURDOC}(x)}_{=} = 0$



ASSURDOC è un algoritmo per D



contradizione perché D non è RICORSIVO



A non è ricorsivo!



Il problema dell'arresto è indecidibile

Un particolare sistema generativo:

la grammatica

Consideriamo un sottolinguaggio
dell'italiano: gli annunci
ferroviari in stazione

$\langle \text{soppressione} \rangle \rightarrow \text{Il treno } S\langle \text{num} \rangle$
o nelle $\langle \text{orario} \rangle$ è soppresso.

$\langle \text{num} \rangle \rightarrow 1|2| \dots |13$

$\langle \text{orario} \rangle \rightarrow \langle \text{ore} \rangle \text{ e } \langle \text{minuti} \rangle$

$\langle \text{ore} \rangle \rightarrow 1|2| \dots |24$

$\langle \text{minuti} \rangle \rightarrow 00|01| \dots |59$

Osservazioni:

$\langle \dots \rangle$ è variabile ed è chiamato
METASIMBOLO

non $\langle \dots \rangle$ è fisso ed è chiamato
SIMBOLO TERMINALE

\langle soppressione \rangle è il simbolo iniziale detto
ASSIOMA

$\dots \rightarrow \dots$ è una regola detta
regola di PRODUZIONE

\Rightarrow indica l'applicazione di una regola
ed è chiamato PASSO DI DERIVAZIONE

\Rightarrow^* è l'applicazione di più regole dette
DERIVAZIONE IN ZERO O PIÙ PASSI

Esempio:

<soppressione> \Rightarrow Il treno S<num>

delle <orario> è soppresso \Rightarrow

Il treno S6 delle <orario> è soppresso

\Rightarrow Il treno S6 delle <ore> e <minuti> è
soppresso \Rightarrow^* Il treno S6 delle
10 e 20 è soppresso



è una derivazione
per la generazione di
una frase degli annunci
ferroviari

Formalizziamo:

Definizione:

Una grammatica è una quadrupla:

$$G = (\Sigma, M, S, P) \text{ dove}$$

- Σ insieme finito dei simboli terminali
- M " " " dei metasimboli

$$\Sigma \cap M = \emptyset$$

- S è un simbolo in M , $S \in M$ ed è l'ASSIOMA
- P è l'insieme finito delle regole di produzione

Definizione di regola di produzione

$\alpha \rightarrow \beta$ dove: $\alpha \in (\Sigma \cup M)^+$
 $\beta \in (\Sigma \cup M)^*$

Esempio: $S \rightarrow \epsilon$ o.k. $\epsilon \rightarrow S$ no

Definizione: passo di derivazione

Si dice che w è derivabile da z
in un passo e si scrive

$$z \Rightarrow w$$

quando:

$$z = x\alpha y \quad \text{e} \quad w = x\beta y \quad \text{e}$$

$$\alpha \rightarrow \beta \in P.$$

Nota: $x\alpha y \xrightarrow{\alpha \rightarrow \beta} x\beta y$

Definizione: derivazione in zero o più passi

Si dice che w è derivabile da z in zero o più passi e si scrive

$$z \Rightarrow^* w$$

quando:

esiste $K \in \mathbb{N} \setminus \{0\}$ e esistono le parole w_1, w_2, \dots, w_K tali che

$$z \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_K = w$$

oppure

$$w = z.$$

Definizione

Il linguaggio generato da G è:

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

Osservazioni:

- Un linguaggio può ammettere più grammatiche che lo generano
- Due grammatiche G_1 e G_2 si dicono equivalenti se

$$L(G_1) = L(G_2)$$

Esempi

$$G = (\{a\}, \{S\}, S, \{S \rightarrow aS, S \rightarrow a\})$$

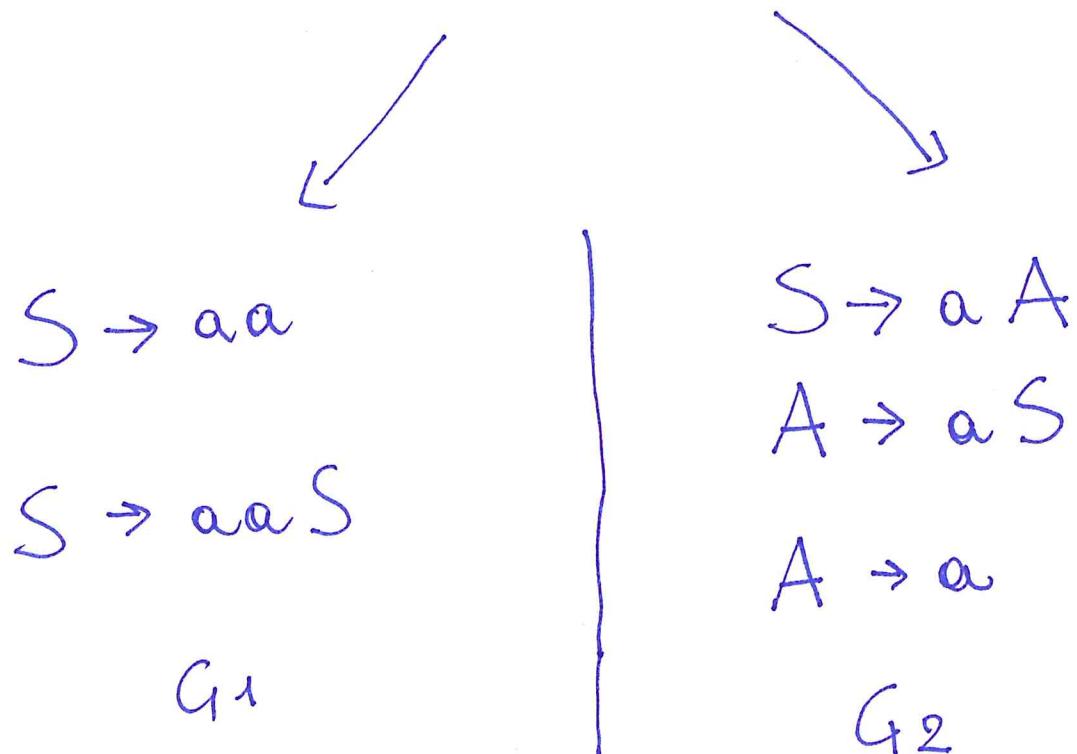
$$L(G) = ?$$

$$S \Rightarrow a$$

$$\begin{aligned} S &\Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow^* a^{n-k}S \\ &\Rightarrow a^n \end{aligned}$$

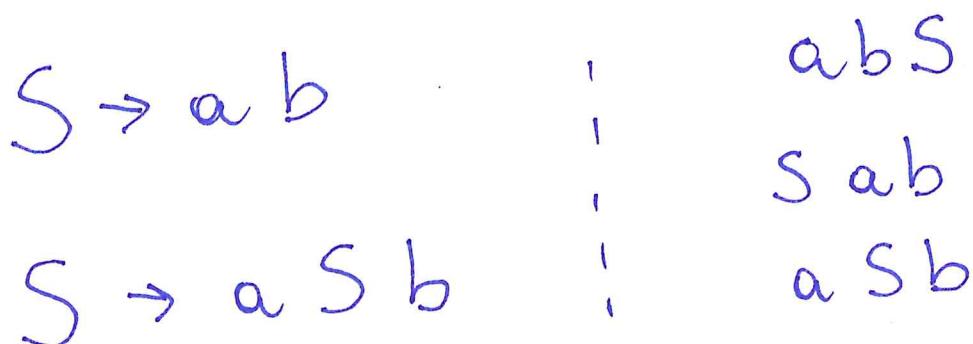
$$L(G) = a^+$$

- $L = \{ a^{2n} \mid n \geq 1 \}$



G_1 e G_2 sono equivalenti

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



$$G = (\Sigma, M, S, P)$$

$$L(G) = \{ \omega \in \Sigma^* \mid S \xrightarrow{*} \omega \}$$

$$\cdot \quad \{a^n b^n \mid n > 0\}$$

$$S \xrightarrow{} a S b$$

$$S \xrightarrow{} a b$$

$$S \xrightarrow{} a S b \xrightarrow{} a a S b b \Rightarrow$$

$$\dots \xrightarrow{} a^{n-1} S b^{n-1} \xrightarrow{} a^{n-1} a b b^{n-1} \Rightarrow$$

$$\Rightarrow a^n b^n$$

- $\Sigma = \{0, 1\}$

Palindrome: $\hat{P} = \{ w w^R \mid w \in \{0, 1\}^*\}$

$$w = w_1 w_2 \dots w_n$$

$$w^R = w_n w_{n-1} \dots w_1$$

Esempio:	0'0	01
	01'10	010
	101'101	si no

Regole:

$$S \rightarrow 00$$

$$S \rightarrow 11$$

$$S \rightarrow 0 S 0$$

$$S \rightarrow 1 S 1$$

- $\{ \omega \omega^R \mid \omega \in \{0,1\}^+ \}$

$S \Rightarrow 0S0$

$S \Rightarrow 00$

$S \Rightarrow 1S1$

$S \Rightarrow 11$

$S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 010S010$

$\Rightarrow 010\underset{\omega}{\overset{|}{\underset{|}{0}}}010$
 $\omega \qquad \omega^R$

- $\{ \omega \in \{0,1\}^+ \mid \#_1(\omega) = \#_0(\omega) \}$

$\#_\sigma(\omega)$ = numero di simboli σ in ω

0101 : $\#_0 = 2 = \#_1$

1100 " "

1101 $\#_1 = 3$ $\#_0 = 1$

Diamo una possibile G

(Tale G non è la migliore ma è
molto semplice da capire)

- $S \rightarrow ZSU$ • $Z \rightarrow 0$
- $S \rightarrow ZU$ • $U \rightarrow 1$

$$S \Rightarrow ZSU \Rightarrow ZZSUV$$

$$\Rightarrow ZZZUVUV$$

$ZZ \dots ZZ \underbrace{UU \dots UU}$

- $ZU \rightarrow UZ$

$ZZ \dots \underbrace{ZUZU} \dots UU$

$ZZ \dots UZZU \dots UV$

e così via

• $\{ a^m b^n c^m \mid m > 0 \}$

$S \rightarrow a SBC$

$S \rightarrow a BC$

$S \Rightarrow a SBC \Rightarrow a a SBCBC$

$\Rightarrow a a a BC \overbrace{BCBC}^{BBBCCC}$

$a BC \overbrace{BCBC}^{a BBB CCC}$
 $a BB \overbrace{CCBC}^{a BBB CCC}$
 $a BB \overbrace{CBCC}^{a BBB CCC}$
 $a BBB CCC$

$CB \rightarrow BC$

~~$B \rightarrow b$~~
 ~~$C \rightarrow c$~~

\rightsquigarrow

$aB \rightarrow ab \mid$

$bB \rightarrow bb \mid$

$bC \rightarrow bc \mid$

$cC \rightarrow cc \mid$

DIPENDENTI
CONTESTO

DA

CONTESTO

DOMANDA :

Tutti i linguaggi ammettono
una grammatica?

Teorema

L è ricorsivamente enumerabile



L è generato da G

dim.

(\uparrow) Data la grammatica G per L
posso costruire una procedura w t.c.

$$F_w(x) = \begin{cases} 1 & x \in L \\ \uparrow & x \notin L \end{cases}$$

Definizioni:

$F_i = \{ \text{parole di simboli terminali e variabili ottenute da } S \text{ in "i" passi di derivazione} \}$

$T_i = \{ \text{parole di soli simboli Terminali ottenute da } S \text{ in "i" passi di derivazione} \}$

nota: $T_i \subseteq F_i$

Fatto: $L(G) = \bigcup_i T_i$

In fatti:

$$(1) \quad x \in L(G) \Rightarrow \text{ho } S \xrightarrow{*} x \Rightarrow$$

$\exists i$ t.c. i è il numero di passi
di derivazione in $S \xrightarrow{*} x \Rightarrow$

$$x \in T_i \Rightarrow x \in \bigcup_i T_i$$

$$(2) \quad x \in \bigcup_i T_i \Rightarrow \exists i \text{ t.c. } x \in T_i \Rightarrow$$

\Rightarrow ho $S \xrightarrow{*} x$ in " i " di derivazione

$$\Rightarrow \text{ho } S \xrightarrow{*} x \Rightarrow x \in L(G)$$

$$(1) \quad L(G) \subseteq \bigcup_i T_i$$

$$(2) \quad \bigcup_i T_i \subseteq L(G)$$

Formalizzo F_i e T_i :

$$F_i = \{ \gamma \in (\Sigma \cup M)^* \mid \eta \Rightarrow \gamma \text{ e } \eta \in F_{i-1} \}$$

$$T_i = \{ x \in \Sigma^* \mid x \in F_i \}$$

Inizialmente costruisco la procedura ELENCA che genera tutte le parole

oh $L(a) = \bigcup_i T_i$

"Elenca" stampa:

T_1

T_2

T_3

:

:

:

Procedura ELENCA ($x \in \Sigma^*$)

{ $G = (\Sigma, M, P, S)$; // Fisso G

$F_0 = \{S\}$;

$i = 1$;

while ($i > 0$) do

{ $F_i = \text{COSTRUISCI} F(F_{i-1}, G)$;

$T_i = \text{COSTRUISCI} T(F_i, G)$;

if $\rightarrow \text{OUTPUT}(T_i)$;

$i = i + 1$;

}

}

funzioni di ELENCA:

costruisci $F(F_{i-1}, g)$

$\{ F_i = \emptyset;$

for each $\gamma \in F_{i-1}$ do

foreach $\alpha \rightarrow \beta \in P$ do

for each $x, y \in (\Sigma \cup M)^*$

such t.c. $M = x \alpha y$ do

$F_i = F_i \cup \{x\beta y\}$

return $(F_i);$

}

costruisce (\bar{F}_i , q)

{ $T_i = \emptyset$;

for each $w \in \bar{F}_i$ do

if ($w \in \Sigma^*$) then

$T_i = T_i \cup \{w\}$;

return (T_i);

}

Adesso trasformo ELENCA nella
procedura W :

I modifica :

Passo $x \in \Sigma^*$ in input ad ELENCA.

II modifica :

Sostituisco OUTPUT(T_i) con

if ($x \in T_i$) Then return(1);

correttezza di ELENCA modificata:

$x \in L(a) \Rightarrow \exists i \text{ t.c. } x \in T_i$

\Rightarrow verrà eseguito return(1)

$\Rightarrow ELENCA(x) = 1$

$x \notin L(a) \Rightarrow x \notin \bigcup_i T_i$

$\Rightarrow \forall i \quad x \notin T_i$

\Rightarrow il ciclo while andrà in loop

$\Rightarrow ELENCA(x) \uparrow$

Linguaggio delle espressioni booleane = L
su \wedge e \vee

Definizione

- $0, 1 \in L$
- $x, y \in L \Rightarrow \{ (x \wedge y), (x \vee y) \} \in L$
- nient'altro appartiene ad L

Grammatica G:

$$\langle \Sigma = \{ 0, 1, (,), \wedge, \vee \} \\ M = \{ E \} \rangle$$

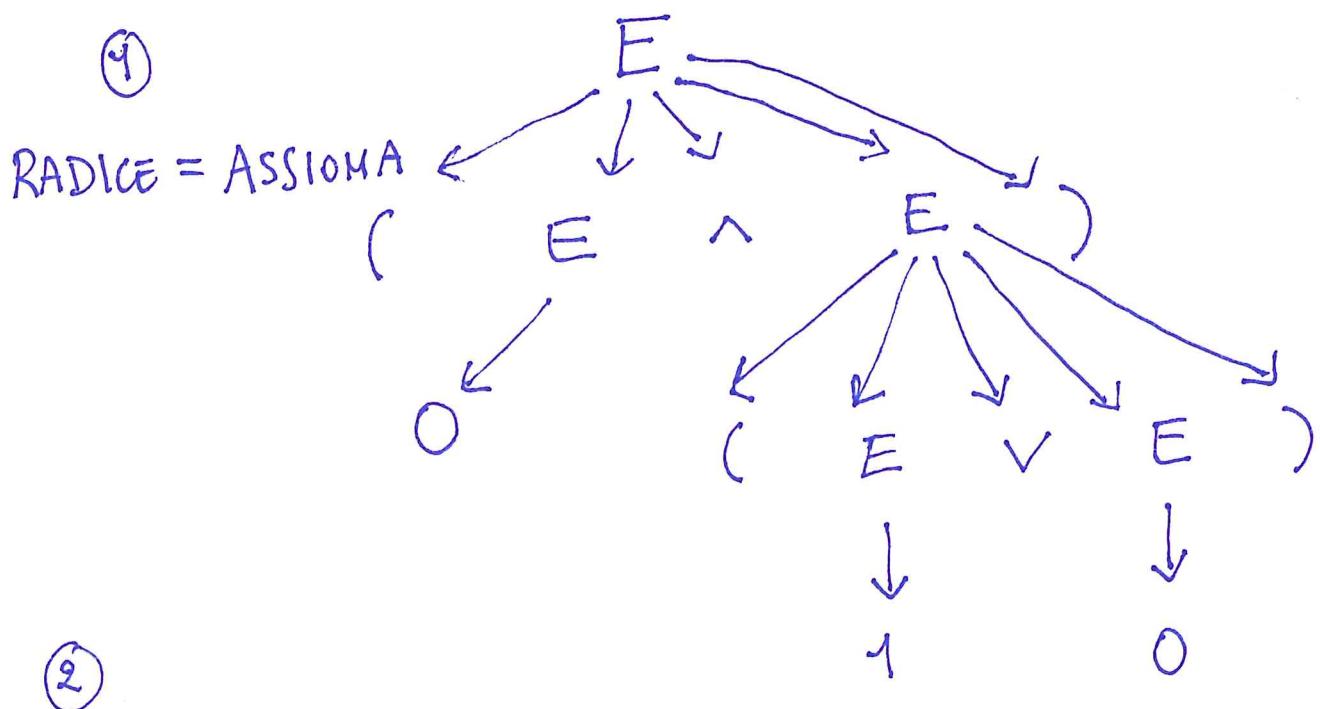
E

$$P = \{ E \rightarrow 0, E \rightarrow 1, \\ E \rightarrow (E \wedge E) \\ E \rightarrow (E \vee E) \}$$

$$\omega = (0 \wedge (1 \vee 0)) \in L$$

$$E \Rightarrow (E \wedge E) \Rightarrow (0 \wedge E) \Rightarrow \\ \Rightarrow (0 \wedge (E \vee E)) \Rightarrow^* (0 \wedge (1 \vee 0))$$

Albero di derivazione per ω



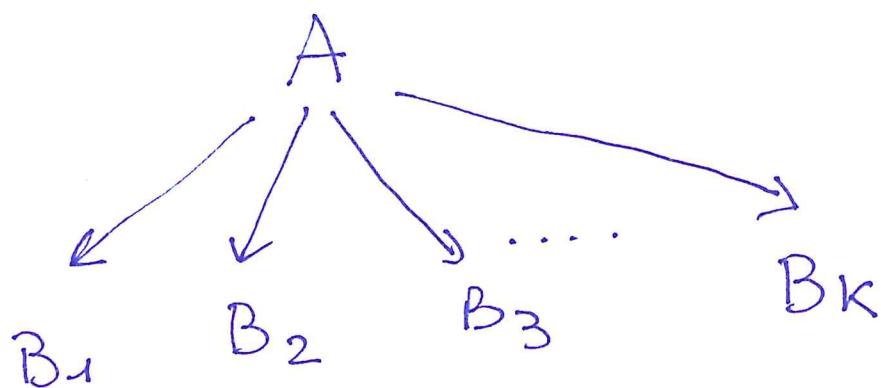
NODI INTERNI = VARIABILI

③
FOGLIE = SIMBOLI TERMINALI

Leggendo le foglie da sin a deo otengo ω

④

E ammesso il sottoalbero :



solo se :

$$A \rightarrow B_1 B_2 B_3 \dots B_k$$

è una regola di G

Osservazione:

Per avere alberi di derivazione
 G deve essere di tipo 2

Classificazione di Chomsky

(versione provvisoria)

Abbiamo 4 tipi in funzione delle regole:

complesse

TIPO 0: nessun vincolo sulle regole

TIPO 1: Sono ammesse le regole

$$\alpha \rightarrow \beta \quad \text{con } |\beta| \geq |\alpha|$$

TIPO 2: Sono ammesse regole della forma

$$A \rightarrow \beta \quad \text{dove } A \in M$$

$$\beta \in (\Sigma \cup M)^+$$

TIPO 3: Sono ammesse regole della forma

$$A \rightarrow yB \quad \text{dove } A, B \in M$$

$$A \rightarrow x \quad y \in \Sigma^*$$

$$x \in \Sigma^+$$

semplice

FATTO : Una grammatica di tipo K
è anche di Tipo K-1.
(per $K=3, 2, 1$)

Definizione

Un linguaggio si dice di tipo K se
ammette G di tipo K che lo genera.

Definizione

$$R_K = \{ L \subseteq \Sigma^* \mid L \text{ è di tipo } K \}$$

R_3 = linguaggi regolari

R_2 = linguaggi liberi dal contesto

R_1 = linguaggi dipendenti dal contesto

R_0 = linguaggi RICORSIVAMENTE ENUMERABILI

Esempi di tipi di G:

TIPO 3: $A \rightarrow yB$, $A \rightarrow xe$

- $S \rightarrow aaS$ per $\{a^{2^n} \mid n > 0\}$
 $S \rightarrow aa$
- $a^+ b^+$ $A \rightarrow aA$
 $A \rightarrow aB$
 $B \rightarrow bB$
 $B \rightarrow b$

TIPO 2: $A \rightarrow B$

- $S \rightarrow aSb$ pu $\{a^m b^m \mid m > 0\}$
- $S \rightarrow ab$

- $B = \text{parole su parentesi ben formate}$

$$S \rightarrow (S)S$$

$$S \rightarrow (S)$$

$$S \rightarrow ()S$$

$$S \rightarrow ()$$

Esempio di parole in B non in B

$()()$, $(())$

$(())$

$((())(())$

$(()(())$
 $)(())$

Tipo 1: $\alpha \rightarrow \beta$ $|\beta| \geq |\alpha|$

* $S \rightarrow a S B C$

$S \rightarrow a B C$

$C B \rightarrow B C$

$a B \rightarrow a b$

$b B \rightarrow b b$

$b C \rightarrow b c$

$c C \rightarrow c c$

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

DOCUMENTI XML

Sono composti da testo + Tag

Tag = marcatore di testo che consente
di dare informazioni semantiche
al testo

Esempio:

<rubrica>

<nome> Bianchi </nome>

<tel> 02 874810 </tel>

<tel> 349 657831 </tel>

<nome> Rossi </nome>

<tel> 05 314861 </tel>

:

:

</rubrica>

Notazione:

Tag aperto $\langle t \rangle$ $\leftarrow t$

Tag chiuso $\langle /t \rangle$ $\leftarrow \bar{t}$

Condizioni:

Un documento XML deve soddisfare:

- ① deve esistere un unico Tag che contiene il documento:

$S \dots \bar{S}$

- ② ogni Tag aperto deve essere seguito dal suo Tag chiuso
- ③ i Tag devono essere imestati correttamente:

$\overline{x} \dots x \dots \bar{y} \dots \bar{y} \dots \bar{x}$

Definizione: un documento XML è CORRETTO se valgono ① e ② e ③

Oltre alla correttezza si richiede

che il documento soddisfi un DTD
(= Grammatica di Tipo 2)

DTD = definisce come i Tag possono
essere innestati fra loro

Esempio: nel documento rubrica
`<tel>` non può stare dentro a `<nome>`

Definizione: Un documento XML è

VALIDO secondo un certo DTD

se è generato da quel DTD

!!

grammatica di Tipo 2

Forme delle regole per i DTD

$$A \rightarrow_a R_a \bar{a}$$

U
↓

R_a = espressione di variabili con $+, ^*, *$

R_a è trasformabile in regole di tipo 2

Esempio di DTD:

$$S \rightarrow_s C^* \bar{s}$$

$$C \rightarrow_c M N (E V)^* \bar{c}$$

$$M \rightarrow_m \bar{m}$$

$$N \rightarrow_n \bar{n}$$

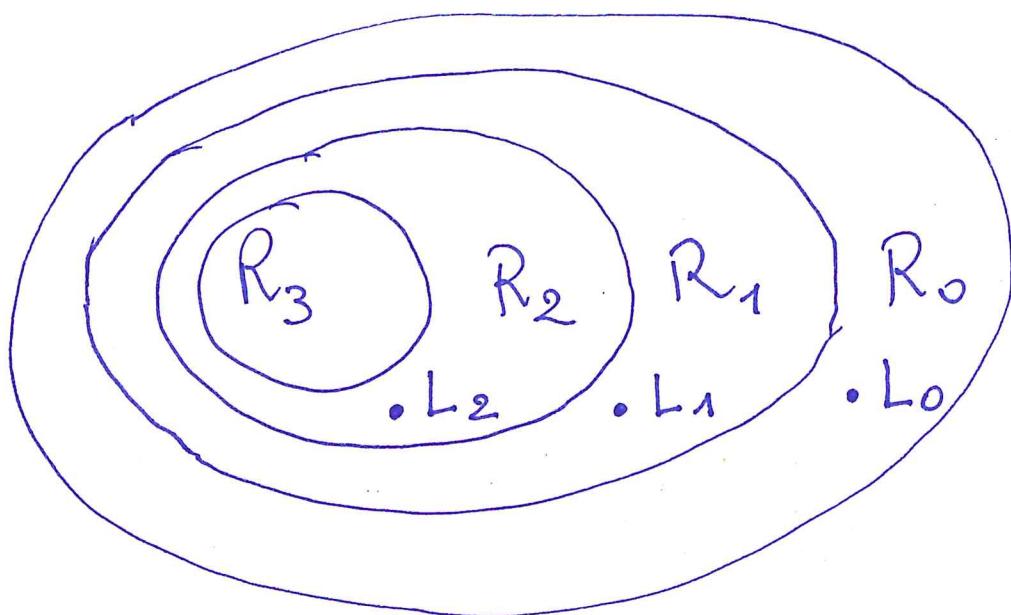
$$E \rightarrow_e \bar{e}$$

$$V \rightarrow_v \bar{v}$$

S = studenti
C = curriculum
U = matricola
M = niente
Z = essere
E = voto

Teorema sugli R_K :

$$R_3 \subset R_2 \subset R_1 \subset R_0$$



" \subset " = sottoinsieme di \neq " \subseteq "
o
inclusione propria

olim. :

- l'inclusione Tra gli R_K segue
dal FATTO

che abbiamo dato sui tipi oli G :

tipo $K \Rightarrow$ tipo $K-1$

- dimostriamo che l'inclusione è propria
- esiste $L_2 \in R_2$ ma $L_2 \notin R_3$

$$L_2 = \{ a^m b^m \mid m > 0 \}$$

Infatti $a^m b^m$ ammette G di tipo 2

$$S \xrightarrow{} a S b, \quad S \xrightarrow{} a b$$

e inoltre vorremo che $a^n b^n \notin R_3$

perché non ammette un automa
a stati finiti che lo genera ricchiosce.

- esiste $L_1 \in R_1$ ma $L_1 \notin R_2$

$$L_1 = \{ a^m b^m c^m \mid m > 0 \}$$

Infatti $a^m b^m c^m$ ammette G di tipo 1

e inoltre vorremo che $a^n b^n c^n \notin R_2$

perché non soddisfa il pumping lemma
per i linguaggi liberi da contesto.

- esiste $L_0 \in R_0$ ma $L_0 \notin R_1$

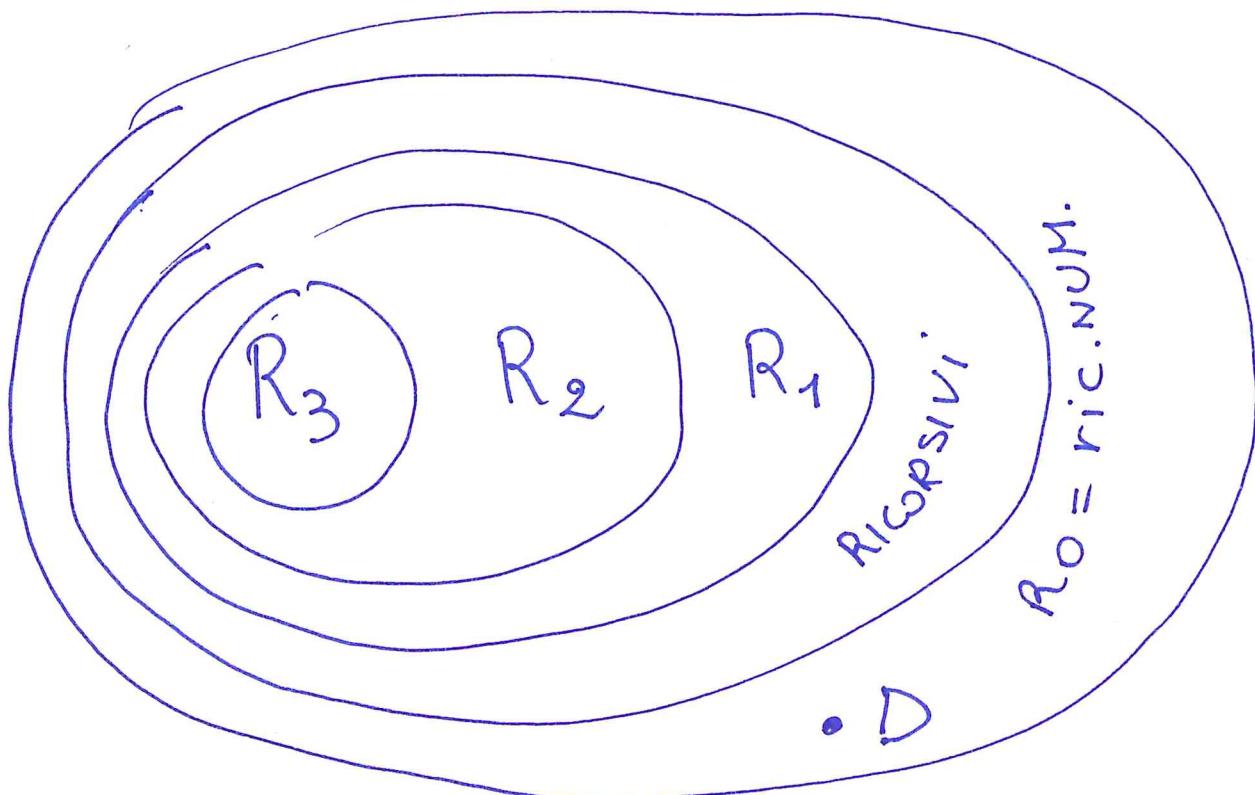
$$L_0 = D = \{x \in \{0,1\}^* \mid F_\mu(x\$x) \downarrow\}$$

olim:

I passo: $R_1 \subseteq$ Ricorsivi

II passo: D non è ricorsivo $\Rightarrow D \notin R_0$

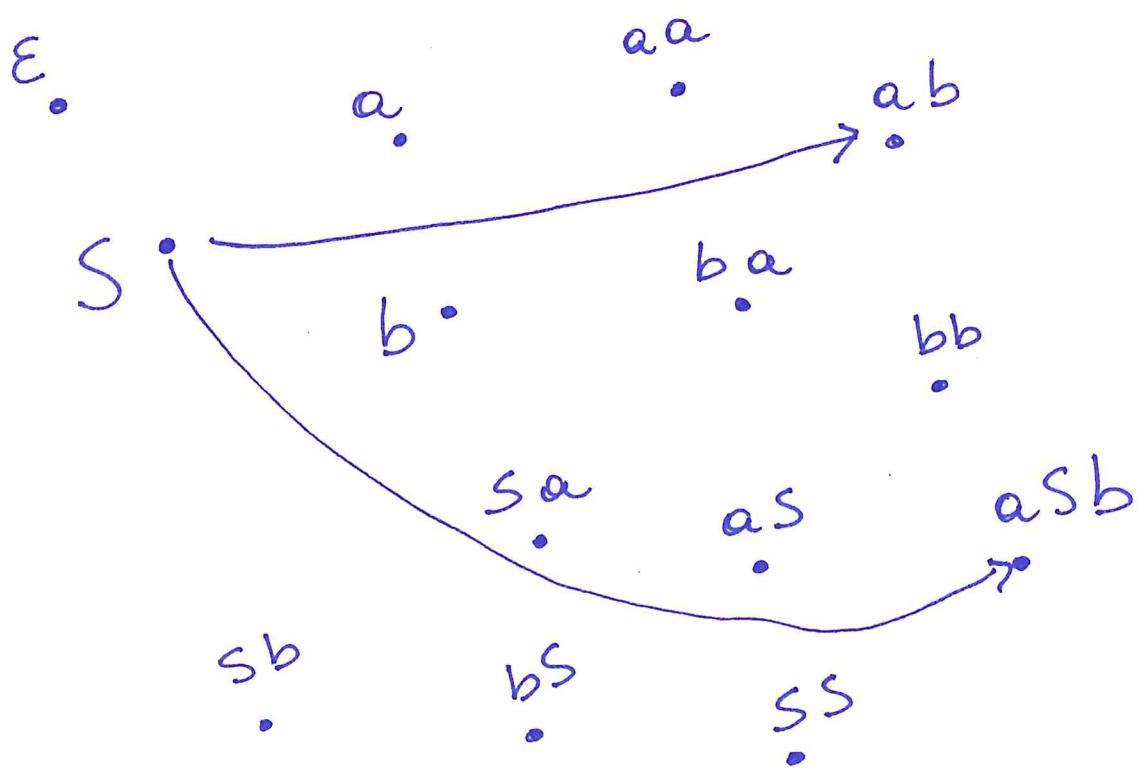
D è ric. enum. $\Rightarrow D \in R_0$



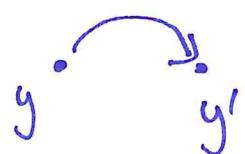
Definizione di $GR(x)$ con $x \in \Sigma^*$

$GR(x) = \langle V_x, E_x \rangle$ dove :

$$V_x = \{ y \in (\Sigma \cup M)^* \mid |y| \leq |x| \}$$



$$E_x = \{ (y, y') \mid y \Rightarrow y' \}$$



Algoritmo ω ($x \in \Sigma^*$)

{ COSTRUISCI GR(x); \leftarrow serve G

if (esiste un cammino
 $S \xrightarrow{*} x$)

 then return(1);

 else return(0);

}

OSServazione:

Il problema della generazione di x
si trasforma nella ricerca di
un cammino in un grafo.

Il tempo dell'algoritmo ω
è esponenziale, richiesto
da costruisci $CIR(x)$.

correttezza di ω :

$x \in L(G) \Rightarrow$ esiste una derivazione in G

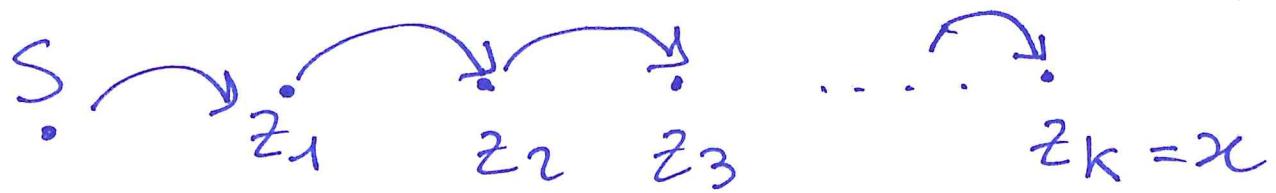
$S \Rightarrow z_1 \Rightarrow z_2 \Rightarrow \dots \Rightarrow z_k = x$ t.c.

per ogni i $|z_i| \leq |z_{i+1}| \Rightarrow$

per ogni i $|z_i| \leq |x|$ e quindi

per ogni i $z_i \in V_x$ inoltre $\overset{\curvearrowright}{z_i} \overset{\curvearrowleft}{z_{i+1}}$

\Rightarrow nel grafo $GR(x)$ si ha



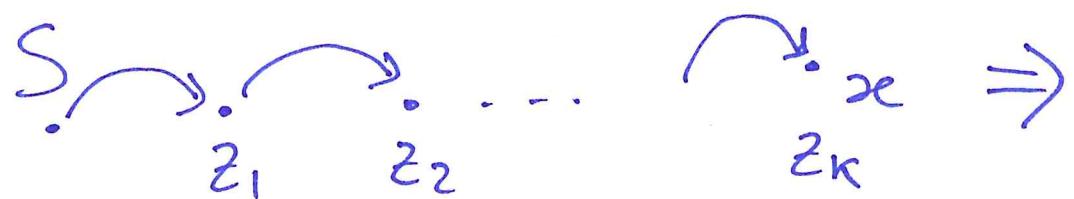
$\Rightarrow F_\omega(x) = 1$

$$x \notin L(G) \Rightarrow F_w(x) = 0$$



$$F_w(x) = 1 \Rightarrow x \in L(G)$$

$F_w(x) = 1$ \Rightarrow esiste un cammino



\Rightarrow allora in G esistono i seguenti passi

$$S \Rightarrow z_1, z_1 \Rightarrow z_2, \dots, z_{k-1} \Rightarrow z_k = x$$

$$\Rightarrow$$
 allora $S \Rightarrow z_1 \Rightarrow z_2 \Rightarrow \dots \Rightarrow z_k = x$

$$\Rightarrow x \in L(G)$$

$x \in L(G)$

Data la classificazione di Chomsky provvisoria

Se L è di tipo K , di che tipo è
 $L \cup \{\epsilon\}$?

Avrei bisogno di $S \rightarrow \epsilon$ consentita
solo dal tipo 0.

E' poco RAGIONEVOLE pertanto
apportiamo 2 modifiche.

I modifica: per le grammatiche di tipo
 $K = 3, 2, 1$

(S) E ammessa la regola $S \rightarrow \epsilon$ a patto che:

- S sia lassiva
- S non compaia sulla destra di
altri regole

FATTO I:

Se L è di tipo K , allora anche $L \cup \{\epsilon\}$
è di Tipo K .

Oss.: Ho G per L di tipo K e
devo costruire G' per $L \cup \{\epsilon\}$ di tipo K

Introduco S' assioma per G' e:

$$S' \rightarrow S, \quad S' \rightarrow \epsilon$$



rispettano (S)

II modifica: per le grammatiche $K = 3, 2$

Ammetto regole nella forma

$$A \rightarrow \epsilon$$

dove A è una variabile arbitraria

FATTO II:

Se ho G di tipo k con $k=3, 2$
con regole $A \rightarrow E$ posso ottenere
 G' equivalente di tipo k che
rispetta (S)

Esempio:

$S \rightarrow 0 S 0$ } di tipo $k=2$
 $S \rightarrow 1 S 1$ } non rispetta (S)
 $S \rightarrow \epsilon$

Applico $S \rightarrow \epsilon$ alle altre regole:

$S \rightarrow 00$ } aggiungo queste regole e:
 $S \rightarrow 11$ } (ed elimino $S \rightarrow \epsilon$)

$S' \rightarrow S$ $S' \rightarrow \epsilon$

di tipo $k=2$ e rispetta (S)

Classificazione di Chomsky (revisione definitiva)

tipo 0: regole arbitrarie

tipo 1: regole nella forma

$$\alpha \rightarrow \beta \quad \text{con } |\beta| > |\alpha|$$

$\left\{ \begin{array}{l} \text{E' ammessa la regola } S \rightarrow E \text{ se:} \\ - S \text{ e' l'assimile} \\ - S \text{ non compare sulla dx di altre} \\ \text{regole} \end{array} \right.$

tipo 2: regole nella forma

$$A \rightarrow \beta \quad \text{dove } A \in M$$

$$\text{e } \beta \in (\Sigma \cup M)^*$$

tipo 3: regole nella forma

$$A \rightarrow xB \quad \text{dove } A, B \in M$$

$$A \rightarrow y \quad x, y \in \Sigma^*$$

- Teorema sugli R_k :

$$R_3 \subset R_2 \subset R_1 \subset R_0$$

Continua a valere grazie al FATTO II.

- Idem per il Teorema: $R_1 \subseteq$ Ricorsivi
La frase (S) preserva la lung. non decres. **stelle** derivazioni
Forme equivalenti per il tipo 3:

$$(i) \quad A \rightarrow^\sigma B, \quad A \rightarrow \sigma, \quad A \rightarrow \epsilon$$

dove $A, B \in M$ e $\sigma \in \Sigma$

$$(ii) \quad A \rightarrow \sigma B, \quad A \rightarrow \epsilon$$

dove $A, B \in M$ e $\sigma \in \Sigma$

$$(iii) \quad A \rightarrow \sigma B, \quad A \rightarrow \sigma \quad]$$

dove $A, B \in M$ e $\sigma \in \Sigma \quad]$

solose
 $\epsilon \notin L$

Trasformazioni:

- posso passare da (i) a (ii):
devo eliminare le regole $A \rightarrow \sigma$:
 - Introduco una nuova variabile X
 - Per ogni regola $A \rightarrow \sigma$ introduco
$$A \rightarrow \sigma X, \quad X \rightarrow \epsilon$$
 - cancello le regole $A \rightarrow \sigma$
- posso passare da (i) a (iii):
devo eliminare le regole $A \rightarrow \epsilon$:
 - per ogni regola $A \rightarrow \sigma B$ aggiungo
$$A \rightarrow \sigma \quad \text{se } B \rightarrow \epsilon \in P$$
 - Cancello le regole $A \rightarrow \epsilon$

Vale l'equivalenza Tra:

meave e desaparecer!

$$\begin{array}{l}
 A \rightarrow x B \\
 A \rightarrow y \\
 A, B \in M \\
 x, y \in \Sigma^*
 \end{array}
 \quad \longleftrightarrow$$

$$\begin{array}{c}
 A \rightarrow a \\
 A \rightarrow b \\
 A \rightarrow c \\
 A, B \in \Sigma \\
 a \in \Sigma
 \end{array}$$

caso particolare della
 linea re a destra

(\Rightarrow) Trasformazioni:

Esempio:

- $A \Rightarrow x B$ con $|x| > 0$
- i) $A \Rightarrow abcB$ $a, b, c \in \Sigma$
 - - - - - X
- - - - - $A \Rightarrow ax, X \Rightarrow bcB$
- - - - - $\{A \Rightarrow ax, X \Rightarrow b^y, Y \Rightarrow cB\}$ $y \leftarrow$
 - - - - - $decreasing$ $size$ of $string$ x

- stessa tecnica per $A \rightarrow y$, $|y| > 0$

problema :

regole del tipo $A \rightarrow xB$ con $|x| = 0$.

$A \rightarrow B$ regole unitarie!

possono essere trasformate in

$A \rightarrow \emptyset B$ e $A \rightarrow \emptyset$

prova: NO.