

Automi e Linguaggi Formali

Grammatiche libere da contesto – Alberi sintattici –

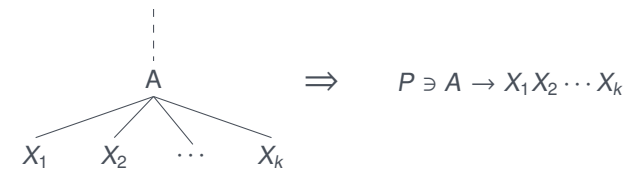
A.A. 2014-2015
Enrico Mezzetti
emezzett@math.unipd.it



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Costruzione di un albero sintattico

- Sia CFG $G = (V, T, P, S)$
- Gli **alberi sintattici** per G soddisfano le seguenti condizioni:
 - 1 Ogni nodo interno e' etichettato con una variabile in V
 - 2 Ogni foglia e' etichettata con un simbolo in $V \cup T \cup \{\epsilon\}$.
(Una foglia etichettata con ϵ e' l'unico figlio del suo genitore)
 - 3 Se un nodo interno e' etichettato A , e i suoi figli (da sinistra a destra) sono etichettati X_1, X_2, \dots, X_k , allora
$$A \rightarrow X_1 X_2 \dots X_k \in P$$

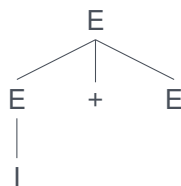


Automi e Linguaggi Formali – A.A 2014-2015
Docente: Enrico Mezzetti

2 of 17

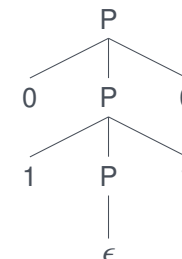
Esempio

- Grammatica delle espressioni $G_{PL} = (\{E, I\}, T, P, E)$
- L'insieme delle produzioni P e' definito
 - $E \rightarrow I \mid E + E \mid E \times E \mid (E)$
 - $I \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$
- Albero sintattico per $E \xRightarrow{*} I + E$ in G_{PL}



Esempio - 2

- Grammatica delle palindrome $G_{pal} = (\{P\}, \{0, 1\}, A, P)$
- L'insieme delle produzioni P e' definito
 - $P \rightarrow \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$
- Albero sintattico per $P \xRightarrow{*} 0110$ in G_{pal}



Automi e Linguaggi Formali – A.A 2014-2015
Docente: Enrico Mezzetti

3 of 17



Automi e Linguaggi Formali – A.A 2014-2015
Docente: Enrico Mezzetti

4 of 17

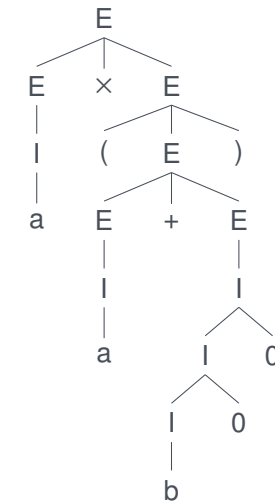
Il prodotto di un albero sintattico

- Il **prodotto** di un albero sintattico e' la stringa ottenuta dal *concatenamento* delle foglie da sinistra a destra
 - E' una stringa *derivabile* dalla variabile radice
- Sono rilevanti gli alberi sintattici in cui:
 - Il prodotto e' una *stringa terminale*
 - La radice e' etichettata dal *simbolo iniziale*
- L'insieme dei prodotti di questi alberi sintattici rappresenta esattamente il linguaggio della grammatica



Esempio

- $G_{PL} = (\{E, I\}, T, P, E)$
 - Root e' E (iniziale)
 - Foglie sono *terminali*
- Prodotto $a \times (a + b00)$



Equivalenza tra inferenza, derivazioni e alberi

- Data una CFG $G = (V, T, P, S)$ e una variabile $A \in V$, i seguenti enunciati sono equivalenti:
 - Per inferenza ricorsiva si puo' stabilire che $w \in L(A)$
 - $A \xRightarrow{*} w$
 - $A \xRightarrow{lm} w \iff A \xRightarrow{rm} w$
 - Esiste un albero sintattico con radice A e' prodotto w .
- Per provare l'equivalenza, usiamo il seguente piano



👁️ 2-4 valgono per qualsiasi forma sentenziale



Da inferenze ad alberi

Th. 5.12 Sia $G = (V, T, P, S)$ una CFG. Se la procedura di inferenza ricorsiva indica che la stringa terminale w e' nel linguaggio della variabile A , allora esiste un albero sintattico con radice A e prodotto w .

Prova: per induzione sul numero di passi dell'inferenza

Base (un solo passo) \rightarrow solo base della procedura di inferenza
Esiste una produzione $A \rightarrow w$ (incluso caso $w = \epsilon$)



Da inferenze ad alberi - 2

Induz. (n+1 passi) → per ip.induttiva vale per inferenze di n passi
Consideriamo l'ultimo passo dell'inferenza

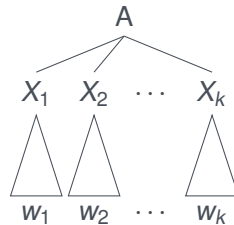
$$A \rightarrow X_1 X_2 \dots X_k \quad \text{dove } X_i \in \{V \cup T\}$$

Scomponiamo w in $w_1 w_2 \dots w_k$:

- Se X_i e' terminale allora $w_i = X_i$
- Se X_i e' variabile allora $w_i \in L(X_i)$ (in n passi)

Possiamo costruire un albero con radice A e prodotto w

- Prodotto determinato dalla concatenazione dei prodotti dei sub-tree



Da alberi a derivazioni

- Costruire una derivazione (*lm* o *rm*)
- **Incorporazione** di derivazioni
- Se esiste una derivazione

$$E \Rightarrow I \Rightarrow Ib \Rightarrow ab$$

allora vale anche

$$\alpha E \beta \Rightarrow \alpha I \beta \Rightarrow \alpha I b \beta \Rightarrow \alpha a b \beta \quad \forall \alpha, \beta$$

- In altre parole, possiamo applicare le stesse produzioni **indipendentemente dal contesto** (**context-free**)
- Permette di passare da albero a derivazione



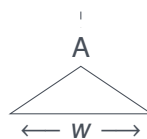
Da alberi a derivazioni - 2

Th. 5.14: Sia $G = (V, T, P, S)$ una CFG, e supponiamo che esista un albero sintattico con radice etichettata con la variabile A e prodotto w con $w \in T^*$. Allora esiste una derivazione a sinistra $A \xRightarrow{*}_{lm} w$ (e anche a destra) in G.

Prova: Per induzione sull'altezza dell'albero.

Base (Altezza 1) Albero di altezza minima

- Una radice e figli che formano w (ds sx. a dx.)
- Deve esistere una produzione $A \rightarrow w$
- Quindi esiste anche una derivazione a sx $A \xRightarrow{*}_{lm} w$ di un solo passo di w da A

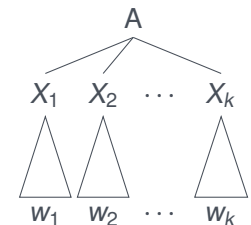


Da alberi a derivazioni - 3

Induz. (Altezza n>1)

Una radice A con figli etichettati $X_1 X_2 \dots X_k$ dove $X_i \in \{V \cup T\}$

- ▶ Se X_i e' terminale allora $w_i = X_i$
 - ▶ Se X_i e' variabile allora X_i e' radice di sottolabero con prodotto w_i (e di altezza n)
- Costruiamo una derivazione a sx. partendo da $A \xRightarrow{*}_{lm} X_1 X_2 \dots X_k$ e proviamo



$$\forall i \in 1, 2, \dots, k \quad A \xRightarrow{*}_{lm} w_1 w_2 \dots w_i X_{i+1} X_{i+2} \dots X_k$$

- Per induzione su i $A \xRightarrow{*}_{lm} w_1 w_2 \dots w_{i-1} X_i X_{i+1} \dots X_k$

$$i=0 \quad A \xRightarrow{*}_{lm} X_1 X_2 \dots X_k \quad (\text{dato})$$

$$i \geq 1 \quad \text{Se } X_i \text{ e' terminale} \rightarrow A \xRightarrow{*}_{lm} w_1 w_2 \dots w_i X_{i+1} \dots X_k$$

$$\text{Altrimenti } A \xRightarrow{*}_{lm} w_1 w_2 \dots w_{i-1} \alpha X_{i+1} \dots X_k \text{ con } X_i \xRightarrow{*}_{lm} \alpha_1 \dots \xRightarrow{*}_{lm} w_i$$

i=k Abbiamo terminato



Esempio

- Costruiamo la derivazione a sinistra (\Rightarrow) per l'albero lm

- Supponiamo di aver induttivamente costruito la derivazione a sinistra

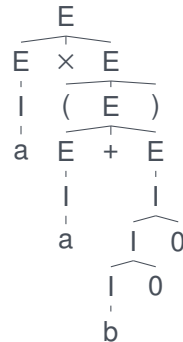
$$E \Rightarrow I \Rightarrow a$$

$lm \quad lm$

e la derivazione a sinistra

$$E \Rightarrow (E) \Rightarrow (E + E) \Rightarrow (I + E) \Rightarrow (a + E) \Rightarrow (a + I) \Rightarrow (a + I0) \Rightarrow (a + I00) \Rightarrow (a + b00)$$

$lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm$



Esempio - 2

- Per la derivazione corrispondente all'intero albero, iniziamo con $E \Rightarrow E * E$ ed espandiamo lm
 - la prima E con la prima derivazione
 - la seconda E con la seconda derivazione

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow a * (E) \Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

$lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm \quad lm$



Applicazioni delle CFG

- Chomsky \rightarrow descrivere il *linguaggio naturale* ✘

- **Terminali:** horse, dog, cat, saw, heard, the
- **Variabili:** <sentence>, <subject>, <verb>, <object>
- **Variabile iniziale:** <sentence>
- **Regole di produzione:**

```
<sentence>  $\Rightarrow$  <subject> <verb> <object>
<subject>  $\Rightarrow$  the horse
<subject>  $\Rightarrow$  the dog
<subject>  $\Rightarrow$  the cat
<object>  $\Rightarrow$  the horse
<object>  $\Rightarrow$  the dog
<object>  $\Rightarrow$  the cat
<verb>  $\Rightarrow$  saw
<verb>  $\Rightarrow$  heard
```

- **Senziali** (solo terminali): "the horse saw the dog", "the dog heard the cat", "the cat saw the horse"



Applicazioni delle CFG - 2

- Descrivere *concetti ricorsivi* in informatica ✔

- Descrizione di linguaggi di programmazione
- Trasformazione automatica da CFG a Parser
- XML (DTD) per la descrizione di tag legali e relazioni

- CFG (abbreviata) per il linguaggio C

- **Terminali:** if do while for switch break continue typedef struct return main int long char float double void static ;() a b c A B C 0 1 2 + * - / _ # include += ++ ...
- **Variabili:** <stmt> <expression> <C source file> <identifier> <digit> <nondigit> <identifier> <selection-stmt> <loop-stmt>
- **Variabile iniziale:** <C source file>



- Regole di produzione:

```
<identifier> ⇒ <nondigit>
              ⇒ <identifier> <nondigit>
              ⇒ <identifier> <digit>
  <nondigit> ⇒ a|b|...|Y|Z|_
  <digit>    ⇒ 0|1|2|3|4|5|6|7|8|9
  <expression> ⇒ <identifier>|<constant>
              ⇒ <cond-expression>|<assign-expression>
<cond-expression> ⇒ <expression> > <expression>
                  ⇒ <expression> != <expression>
<assign-expression> ⇒ <expression> = <expression>
                    ⇒ <expression> += <expression>
  <stmt>           ⇒ <select-stmt>|<loop-stmt>
                  ⇒ <compound-stmt>|<express-stmt>
<select-stmt>    ⇒ if (<expression>)
                  ⇒ if (<expression>)<stmt> else <stmt>
  <loop-stmt>     ⇒ while (<expression>) <stmt>
                  ⇒ do <stmt> while (<expression>)
<express-stmt>  ⇒ <expression> ;
```