Lezione 3 CFG

ambiguità

automi a pila

parsing:
il problema del bilanciamento delle parentesi
(()), (()(())) sono ben bilanciate, (((o ()) non lo sono

G_{bal}=({B},{(,)},P,B), con P uguale a :

B->BB | (B) | ε

è facile dimostrare che non è un linguaggio regolare

anche begin-end e anche altre parentesi

nei linguaggi ci sono anche costrutti che richiedono che ci possano essere più aperte che chiuse

Cond -> if Exp Com else Com | if Exp Com

S ->ε | SS | iS | iSeS

ei non va, anche iee non va, mentre ie e iiiie vanno

è chiaro che questa grammatica genera solo stringhe valide, ma le genera tutte ?

modo semplice per sapere se w in {i,e}* è nella grammatica:

partendo dall'e più a sinistra, trovare il primo i alla sua sinistra ed eliminarli entrambi,

continuare finché possibile e

se alla fine restano solo i o la stringa vuota, allora ok

Yacc è un parser generator: da una descrizione della grammatica genera automaticamente un parser per essa, cioè un programma che data una stringa cerca di costruire un albero sintattico della grammatica che genera la stringa.
--se riesce allora stringa è ok

--se no stringa ha errori sintattici

```
Exp : Id
      | Exp ‘+’ Exp {azioni da fare quando la si usa}
      | Exp ‘*’ Exp {...}
      | ‘(‘ Exp ‘)’      {.....}
      ;
Id     : ‘a’   {.....}
```

Linguaggi di Markup
HTML e XML

DTD = Document Type Definition

<!DOCTYPE nome-della-DTD[
 elenco di definizioni di elementi
]>

<!ELEMENT nome-elemento(descrizione dell'elemento)>
le descrizioni sono espressioni regolari

```
<!DOCTYPE PcSpecs [
        <!ELEMENT PCS (PC*)>
        <!ELEMENT PC (MODEL,PRICE,PROCESSOR,RAM,DISK+)>
        <!ELEMENT MODEL (#PCDATA)>
        <!ELEMENT PROCESSOR (MANF,MODEL,SPEED)>
.....
]>
```

Processor -> Manf Model Speed

Pc -> Model Price Processor Ram Disks
Disks -> Disk Disks

Ambiguità nelle grammatiche e nei linguaggi

grammatiche associano una struttura a programmi, DTD eccetera

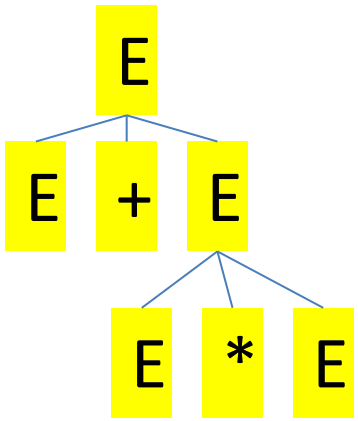
ma è una struttura univoca?

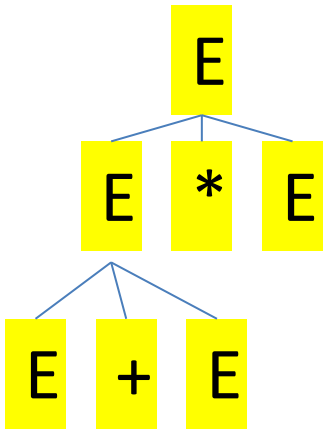
non sempre

esempio di ambiguità

E => E+E =>E+E*E

E=>E*E = E+E*E





1 + 2 * 3

1 + (2*3)=7

ok

(1+2)*3 =9

sbagliato!

Definizione:
Una CFG G=(V,T,P,S) è ambigua, se esiste una stringa w in T* che appartiene al linguaggio di G e per cui esistono 2 (almeno) alberi di derivazione diversi con w come prodotto.

attenzione: non derivazioni diverse ! Ma ALBERI diversi!!

Eliminare l’ambiguità di una grammatica ?

Non è sempre possibile !!
Dipende dal linguaggio che deve generare. A volte è necessario cambiare il linguaggio introducendo dei simboli che servono solo a disambiguarlo.

Nell’esempio delle espressioni notiamo che si sono 2 cause di ambiguità:
--la precedenza degli operatori
--l’associatività degli operatori

--- Si può cambiare la grammatica in modo che implementi la precedenza e anche l'associatività:

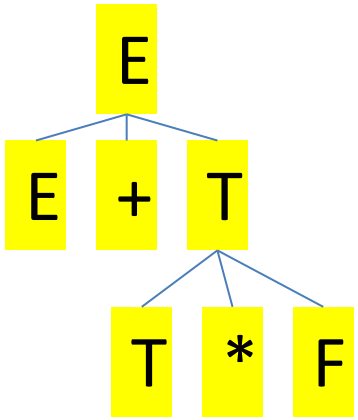
---per la precedenza basta introdurre una variabile per ogni livello di precedenza. Quelle che corrispondono a livelli di precedenza più bassi generano le altre.

I -> a | b | Ia | Ib | I0 | I1

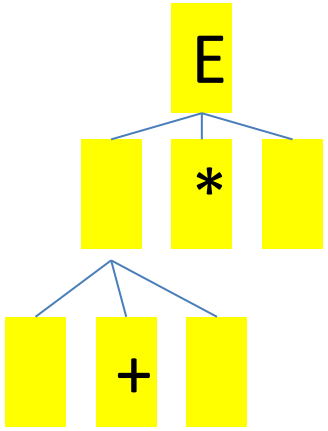
F -> I | (E)

T -> F | T * F

E -> T | E + T

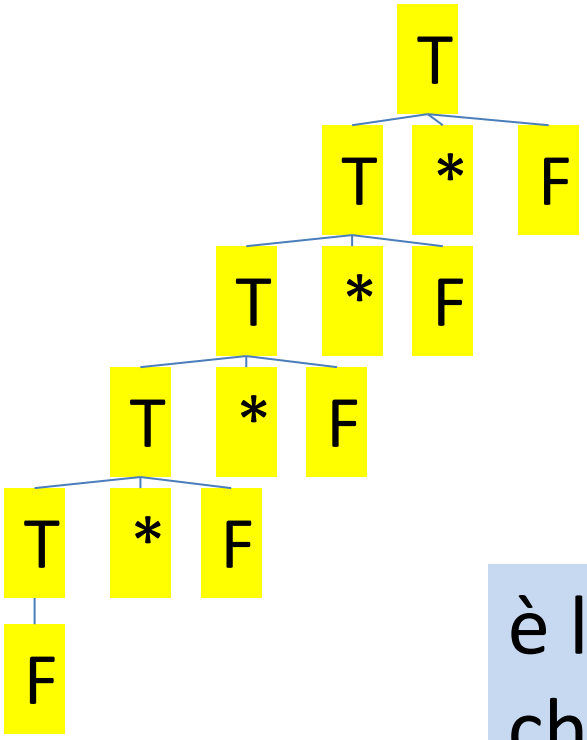


SI



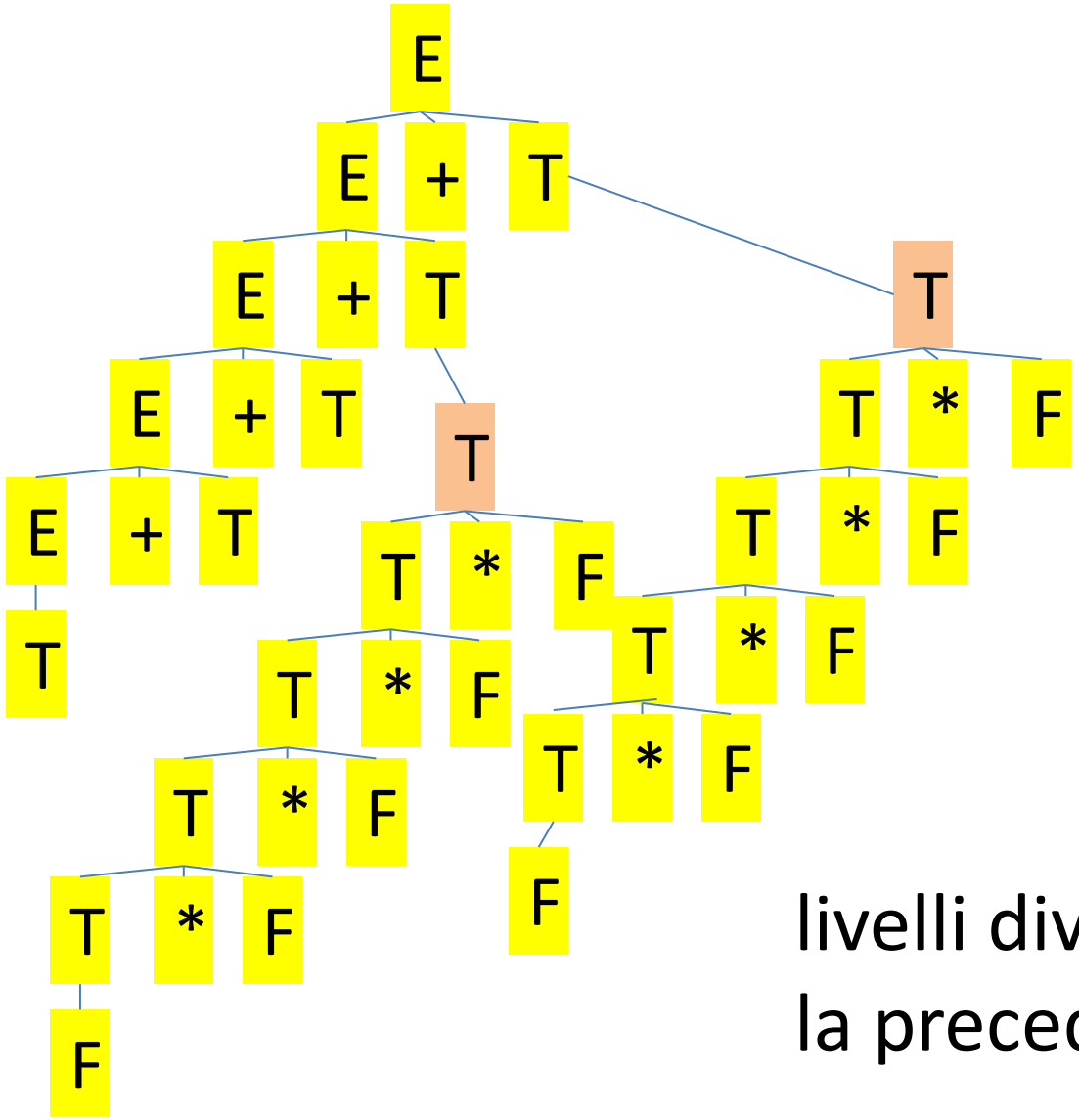
NO

T può produrre solo alberi così:



è l'unico albero di derivazione che genera questa sequenza di * ed è coerente con associatività a sinistra

ed E come funziona?



livelli diversi realizzano
la precedenza

L’ambiguità è denunciata anche dalle derivazioni lm/rm

Teorema: per ogni grammatica G=(V,T,P,S) e per ogni w in T*, w ha 2 alberi sintattici distinti sse ha 2 derivazioni leftmost distinte.

Dimostrazione: ogni albero rappresenta un’unica derivazione leftmost (e anche rightmost)

Ambiguità inerente al linguaggio:

ci sono linguaggi liberi da contesto tali che ogni CFG che li genera è ambigua

quindi non basta cambiare grammatica, ma è necessario cambiare il linguaggio!!

un linguaggio *inerentemente* ambiguo

$L=\{a^n b^n c^m d^m \mid n>01, m>=1\} \cup \{a^n b^m c^m d^n \mid n>01, m>=1\}$

S->AB | C
A->aAb | ab
B -> cBd | cd
C-> aCd | aDd
D-> bDc | bc

S =>AB=>aAbB=>aabbB=>aabbcBd=>aabbccdd
S=>C=>aCd=>aaDdd=>aabDcdd=>aabbccdd

vedere esercizio 5.4.1
interessanti anche 5.4.2 e 5.4.3