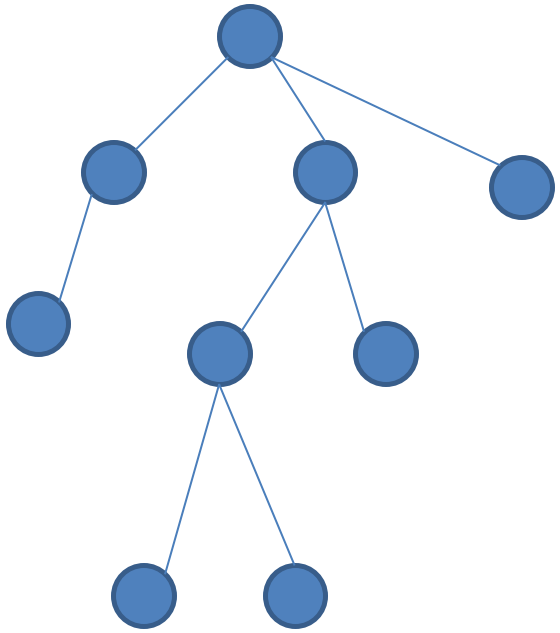
# Lezione 2

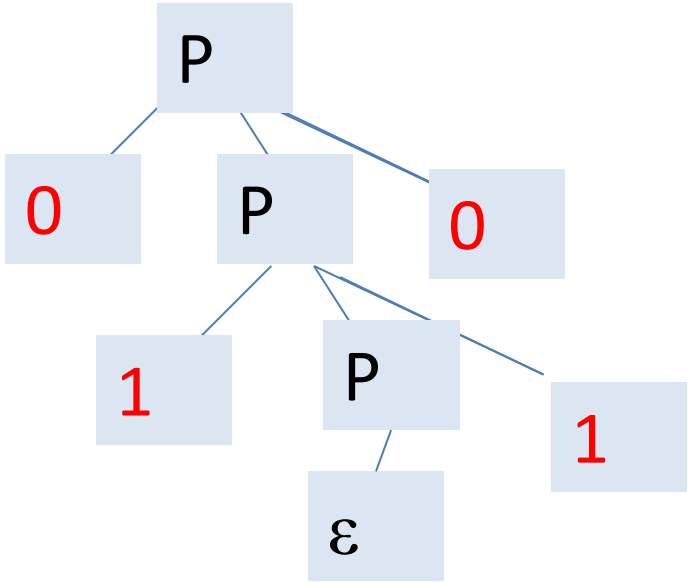
alberi sintattici



radice
nodo interno
foglia

padre
figli ordinati
discendente
frontiera

P -> ε | 0 | 1 | 0P0 | 1P1



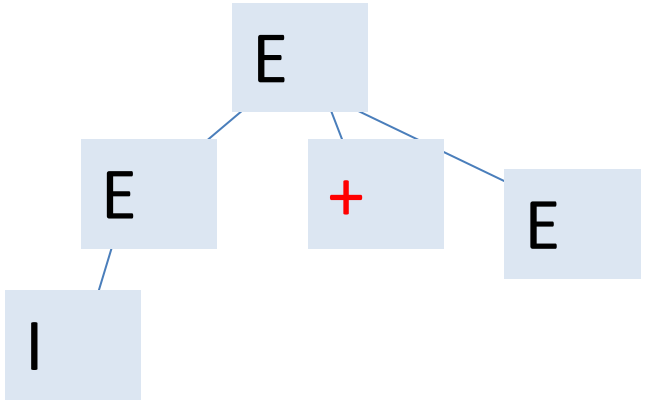

prodotto = 0110

## alberi sintattici

data G=(V,T,R,S)

un albero sintattico di G soddisfa:

1) ciascun nodo interno è etichettato da una variabile
2) ciascuna foglia è etichettata da variabile o terminale o ε, in quest'ultimo caso deve essere l'unico figlio
3) se un nodo interno è etichettato A e i suoi figli (da sinistra a destra) sono X1...Xn, allora A -> X1...Xn è in R.

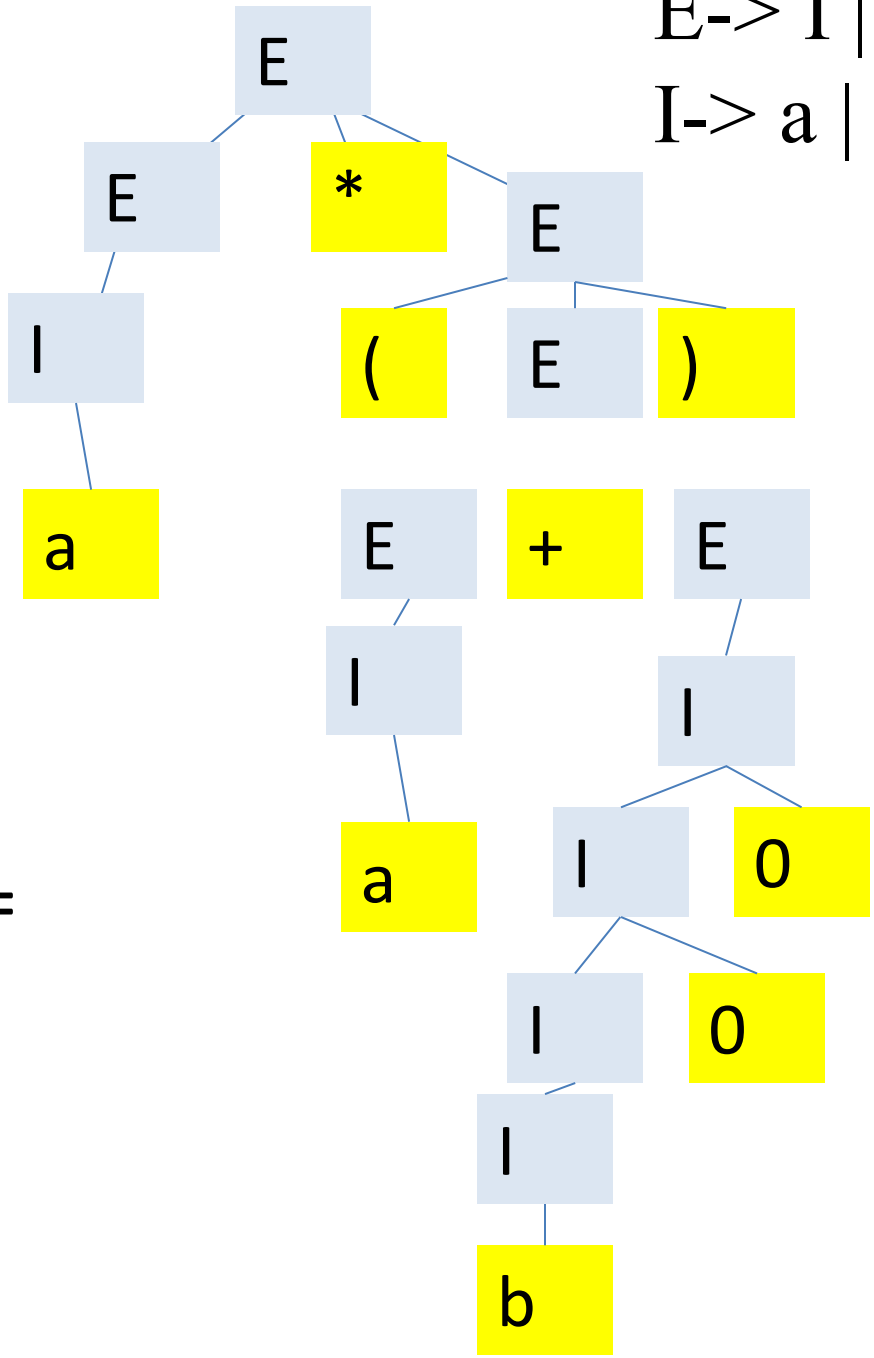



E-> I | E+E | E*E | (E)

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

E-> I | E+E | E*E | (E)

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




a*(a+b00) =

prodotto

dell’albero

abbiamo visto molti modi di caratterizzare il funzionamento delle grammatiche.
Sono:
--inferenza ricorsiva che stabilisce che w è nel linguaggio della variabile  A
-- $A \Rightarrow^* w$
--$A \Rightarrow^{*lm} w$
--$A \Rightarrow^{*rm} w$
-- albero sintattico con radice A e prodotto w

sono tutte equivalenti


derivazione a destra
albero sintattico
derivazione a sinistra
inferenza ricorsiva
derivazione

dagli alberi alle derivazioni

serve una osservazione:
usiamo la  grammatica delle espressioni,
E=>I=>Ib=>ab

per ogni coppia di forme sentenziali α e β vale
α E β=>α I β=>α Ib β => α ab β

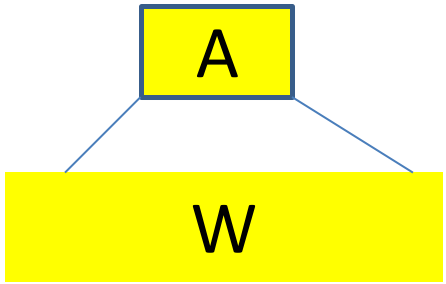
E+(E)=> E+(I)=>E+(Ib)=>E+(ab)

insomma la derivazione è libera dal contesto

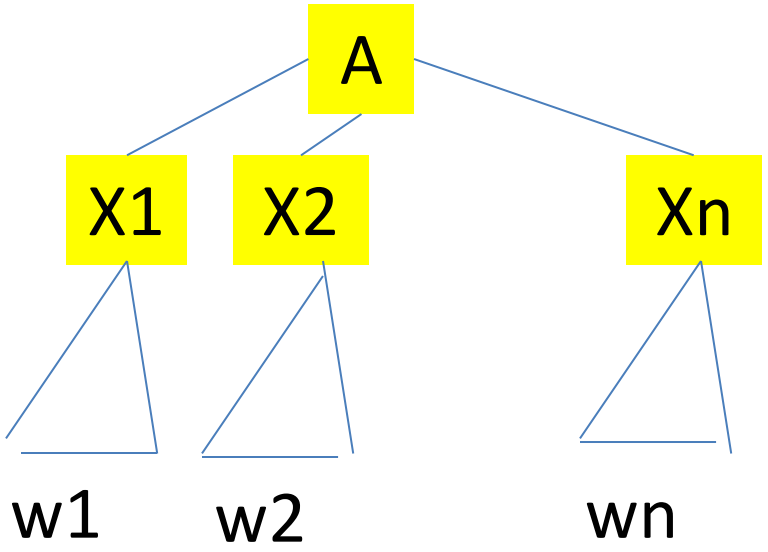
Teorema 5.14. Sia G=(V,T,R,S) una CFG e supponiamo che esista un albero sintattico con radice A e prodotto w in T*. Allora esiste una derivazione leftmost $A =>^{*lm} w$

**Dimostrazione**: induzione sull'altezza dell'albero.

**Base**: altezza 1, consiste di 1 sola produzione di G, ovvio che $A =>^{lm} w$



passo induttivo: un albero di altezza n con n>1, ha la forma, con w=w1w2..wn




dove la produzione A->X1X2..Xn è in R e,

--se Xi è terminale allora wi=Xi

--se Xi è variabile, allora è radice di un sottoalbero di altezza <n e prodotto wi,

per ipotesi induttiva $X_i \Rightarrow^{*lm} w_i$
e quindi per la libertà di contesto:

$A \Rightarrow^{*lm} X_1X_2...X_n \Rightarrow^{*lm} w_1X_2..X_n \Rightarrow^{*lm} w_1w_2...X_n$

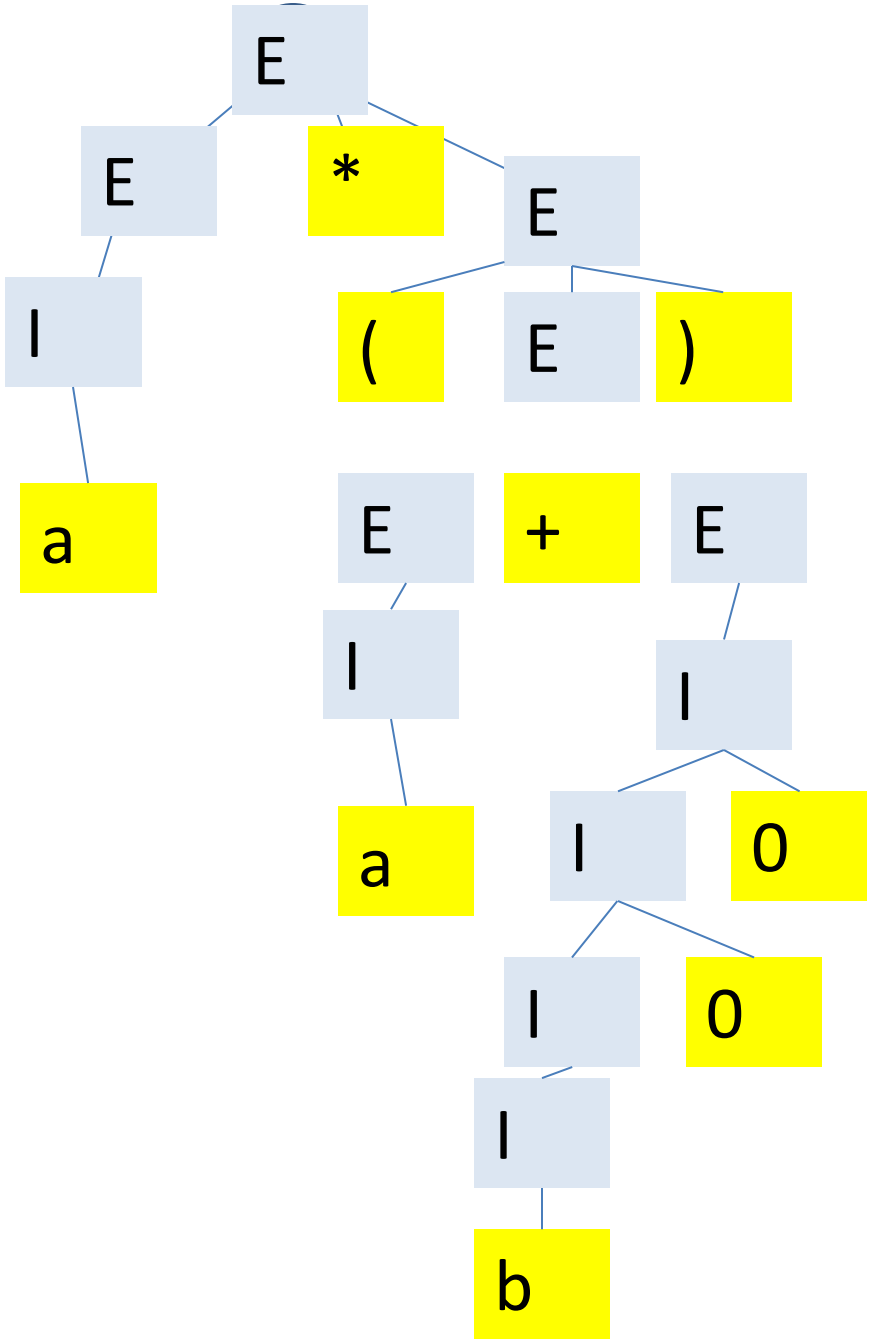
rispettando l'ordine leftmost

formalmente serve un'induzione su i in [1..n]

d1: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$

d2: $E \Rightarrow_{lm} (E)$
$\Rightarrow_{lm} (E+E) \Rightarrow_{lm} (I+E) \Rightarrow_{lm}$
$(a+E) \Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0)$
$\Rightarrow_{lm} (a+I00)$
$\Rightarrow_{lm} (a+b00)$

da cui
$A \Rightarrow_{lm} E*E \Rightarrow_{lm}$ d1 * E $\Rightarrow_{lm}$
d1 * d2



è ovvio che esiste un analogo teorema per le derivazioni rightmost

**esercizi**o 5.2.1

Applicazioni delle CFG

--parsing

--document type definition DTD  che descrive i tag ammessi

parsing:

il problema del bilanciamento delle parentesi

(()), (()(())) sono ben bilanciate, ((( o ()) non sono bilanciate

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

B->BB | (B) | $\varepsilon$

è facile dimostrare che non è un linguaggio regolare

tratta anche begin-end e altre parentesi

nei linguaggi ci sono anche costrutti che richiedono che ci possano essere più aperte (if) che chiuse (else)

Cond -> if (Exp) Cond |if (Exp)  Cond else Cond

S -> iS | iSeS | $\varepsilon$

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

**Vedi esercizio 5.4.2**

Yacc è un parser generator: da una grammatica CF 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’   {.....}
```

vedrete qualcosa di più nel progettino col Prof. Bresolin

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)>
    <!ELEMENT MANF (#PCDATA)>
    <!ELEMENT DISK (HARDDISK | CD | DVD)>
    <!ELEMENT HARDDISK(MANF, MODEL,SIZE)>
    <!ELEMENT CD (SPEED)>
.....
]>
```

documento conforme alla DTD precedente

```
<PCS>
    <PC>
        <MODEL>4560</MODEL>
        <PRICE>$2295</PRICE>
        <PROCESSOR>
            <MANF>Intel</MANF>
            <MODEL>Pentium</MODEL>
            <SPEED>800MHz</SPEED>
        </PROCESSOR>
        <RAM>256</RAM>
        <DISK><HARDDISK>
            <MANF>Maxtor</MANF>
            <MODEL>Diamond</MODEL>
            <SIZE>30.5Gb</SIZE>
        </HARDDISK></DISK>
        <DISK><CD>
            <SPEED>32x</SPEED>
        </CD></DISK>
    </PC>
    <PC>
        ...
    </PC>
</PCS>
```

il DTD è una CFG (o quasi)

<!ELEMENT PROCESSOR (MANF,MODEL,SPEED)>
Processor -> Manf Model Speed

<!ELEMENT DISK (HARDDISK | CD | DVD)>
Disk -> Harddisk | Cd | Dvd

espressioni regolari

<!ELEMENT PC (MODEL,PRICE,PROCESSOR,RAM,DISK+)>

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

Per trasformare espressioni regolari in CFG esercizio 5.1.3