

# Parte 2 del progetto

## analisi sintattica in generale

cominciamo da un esempio (che ci servirà)

grammatica libera da contesto per le espressioni:

$e ::= n \mid e + e \mid e - e$

$n ::= d \mid nd$

$d ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

derivazioni generano forme sentenziali :

$e \rightarrow e - e \rightarrow n - e \rightarrow nd - e \rightarrow dd - e \rightarrow 1d - e \rightarrow 10 - e \rightarrow \dots$

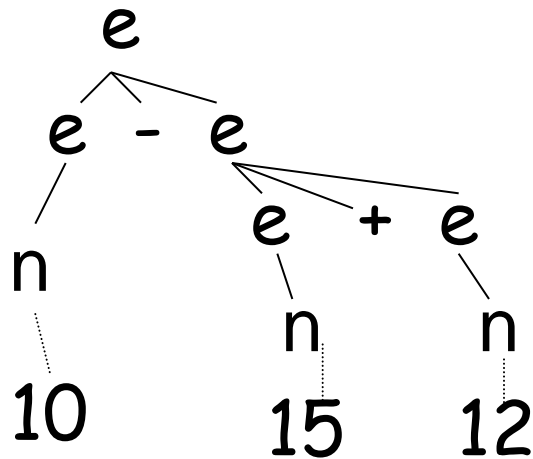
$e \rightarrow 10 - e \rightarrow 10 - 12 - 14 \quad // \text{ fino a stringhe terminali}$

tutte queste stringhe di terminali formano il linguaggio generato dalla grammatica

# derivazione

$e \rightarrow e - e \rightarrow n - e \rightarrow n - e + e \rightarrow \dots \rightarrow 10 - 15 + 12$

## albero di derivazione

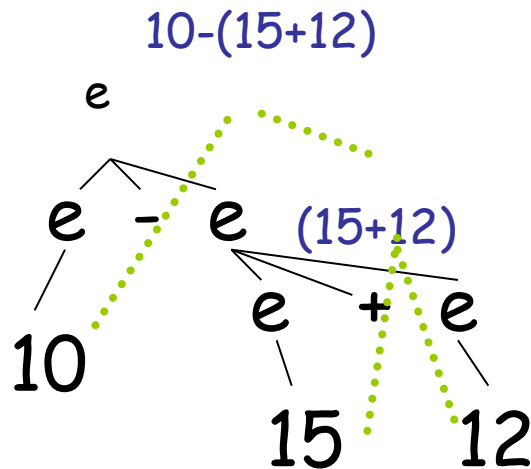


*non rappresenta solo la stringa 10-15+12,*

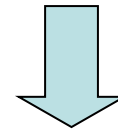
*ma anche il modo in cui va valutata:*

*10-(15+12)*

*e l'albero serve per produrre codice*

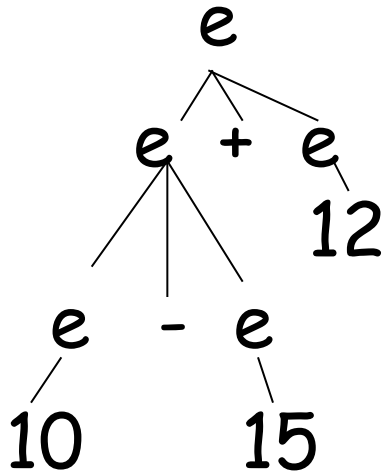


per esempio per creare  
codice che valuta  
l'espressione



```
LOAD 15 R0; LOAD 12 R1; ADD R1 R0;  
LOAD 10 R0; SUB R0 R1;
```

ma allora ci deve essere solo un  
albero di derivazione: e non è così



anche questo genera  
 $10-15+12$

ma lo rappresenta come  
 $(10-15)+12$

**ambiguità !** è importante visto  
che

$$(10-15)+12 \neq 10-(15+12)$$

vorremmo che la grammatica seguisse  
l'associatività e la precedenza degli operatori:

+ e - hanno uguale precedenza ed associano a  
sinistra

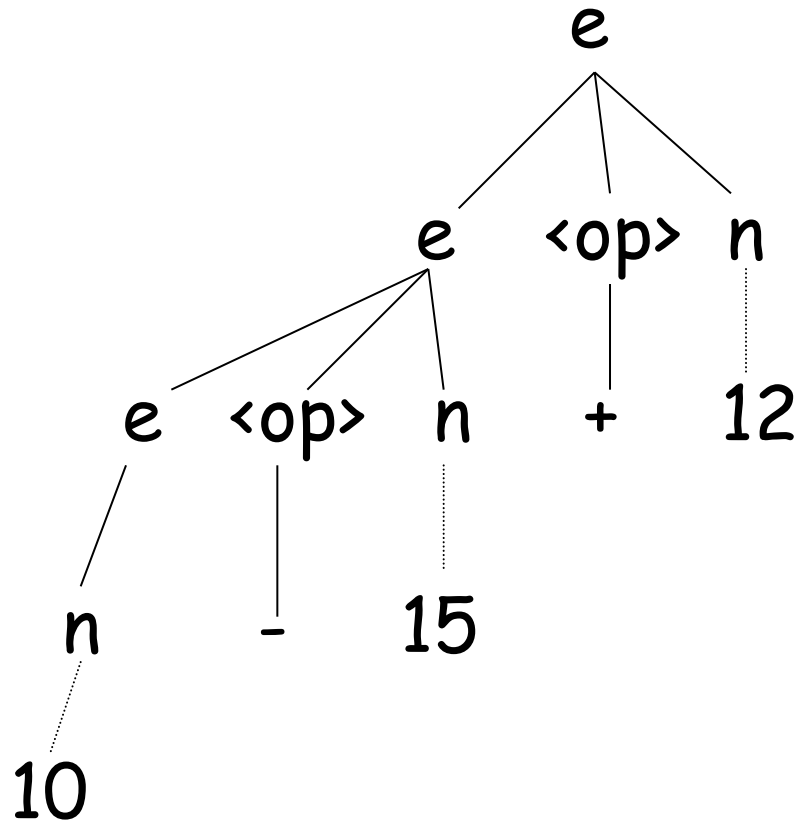
$$10-15+12 = (10-15)+12$$

la seguente grammatica rappresenta corret-  
tamente l'associatività a sinistra:

$e ::= \text{num} \mid e \langle \text{op} \rangle \text{num}$

$\langle \text{op} \rangle ::= + \mid -$

la grammatica produce 10-15+12 in un solo modo:



ed è quello giusto:  
(10-15) +12

funziona sempre?

però questa grammatica non permette di parentesizzare le espressioni in altro modo che tramite l'associatività a sinistra

e se volessimo forzare un altro ordine?


E se avessimo operatori che associano a destra?



introduciamo le  
parentesi esplicite  
nella grammatica

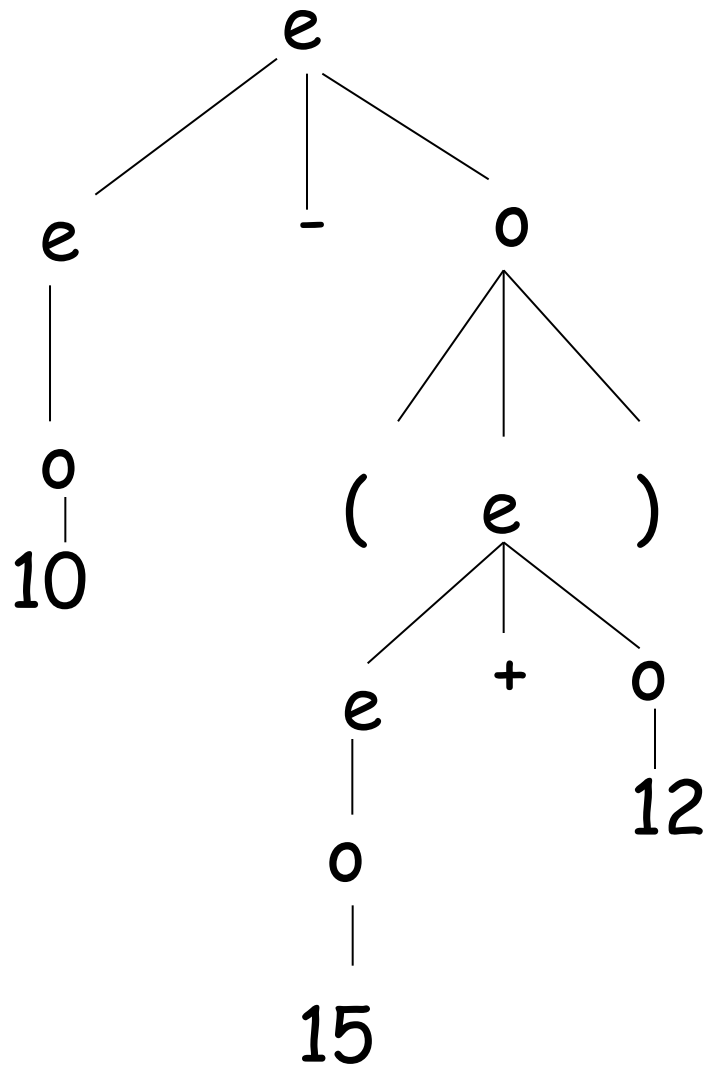
$$e ::= \text{num} \mid e \langle \text{op} \rangle \text{num}$$
$$\langle \text{op} \rangle ::= + \mid -$$

può diventare:


$$e ::= o \mid e \langle \text{op} \rangle o$$
$$o ::= \text{num} \mid (e)$$

è ancora non ambigua?

$10-(15+12)$



ambiguità= 2 alberi di derivazione per una stessa frase del linguaggio

è sempre possibile trovare una grammatica non ambigua per un linguaggio dato ?

NO, ci sono linguaggi inerentemente ambigui

ma per i linguaggi di programmazione il problema è (in generale) facilmente risolvibile

basta introdurre opportuni simboli, tipicamente ; , ( ) { }  
ecc

altra ambiguità tipica nei linguaggi di programmazione : if-then-else

$c ::= \text{if } b \text{ then } c \mid \text{if } b \text{ then } c \text{ else } c$

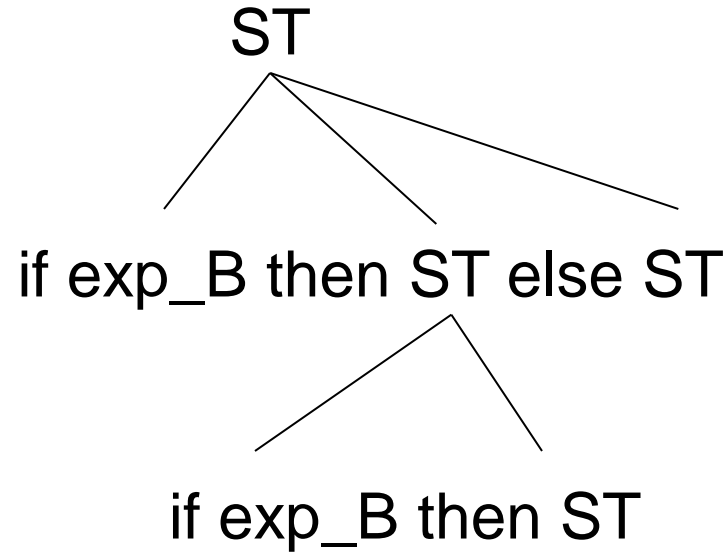
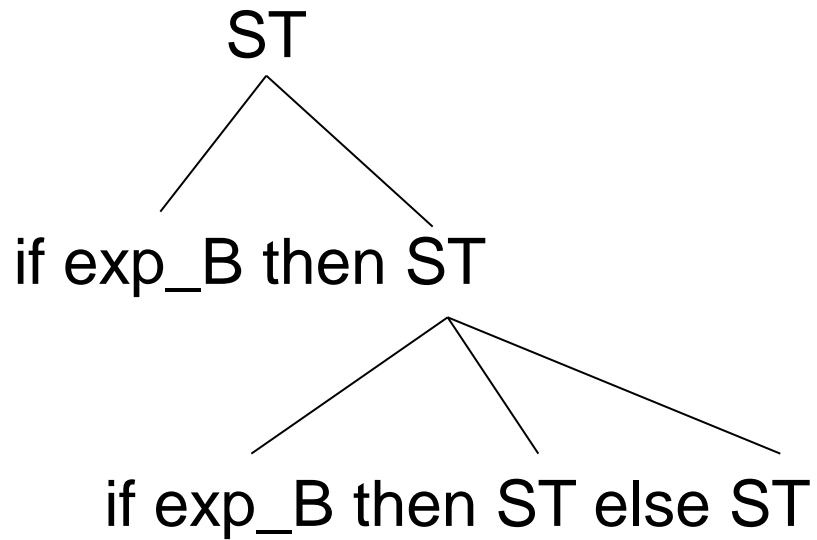
if b1 then if b2 then c1 **else** c2 // quale if?

la regola è che :

if b1 then (if b2 then c1 else c2)

cioè, l'else appartiene al più vicino if

ST::=if exp\_B then ST | if exp\_B then ST else ST | ....  
*altri comandi*



di nuovo la grammatica è ambigua e i LP interpretano

if exp\_B then if exp\_B then ST else ST

sempre come

if exp\_B then (if exp\_B then ST else ST)

vogliamo una grammatica che produca solo  
l'albero di derivazione corretto con il significato  
adottato dai LP

cioè il primo albero della slide precedente

ecco una grammatica che soddisfa la richiesta:

**MST**=Matched ST    **UMST**=UnMatched ST

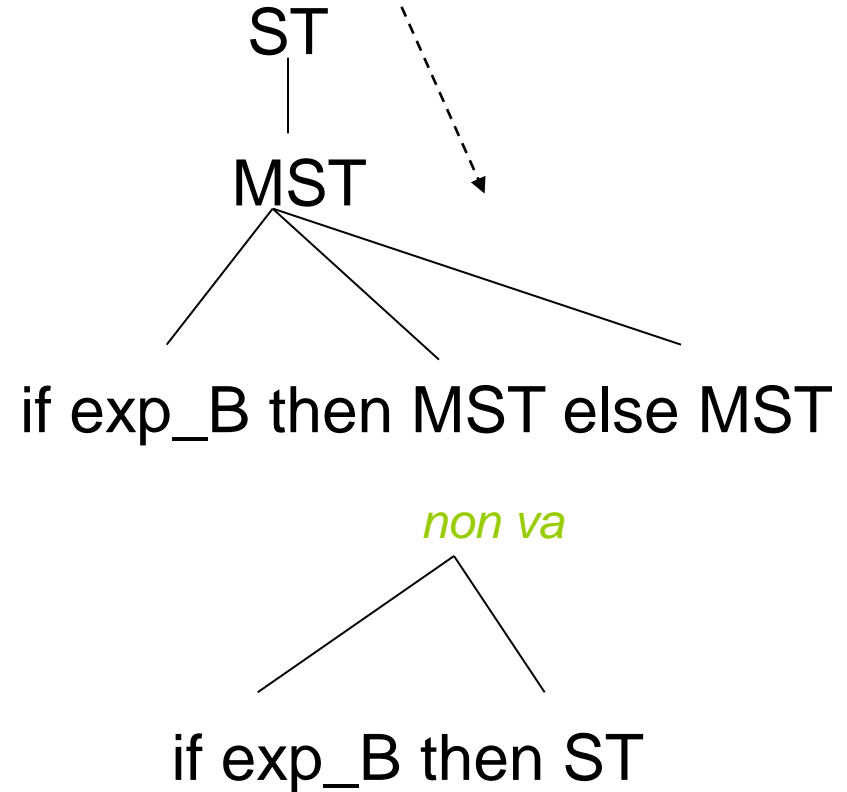
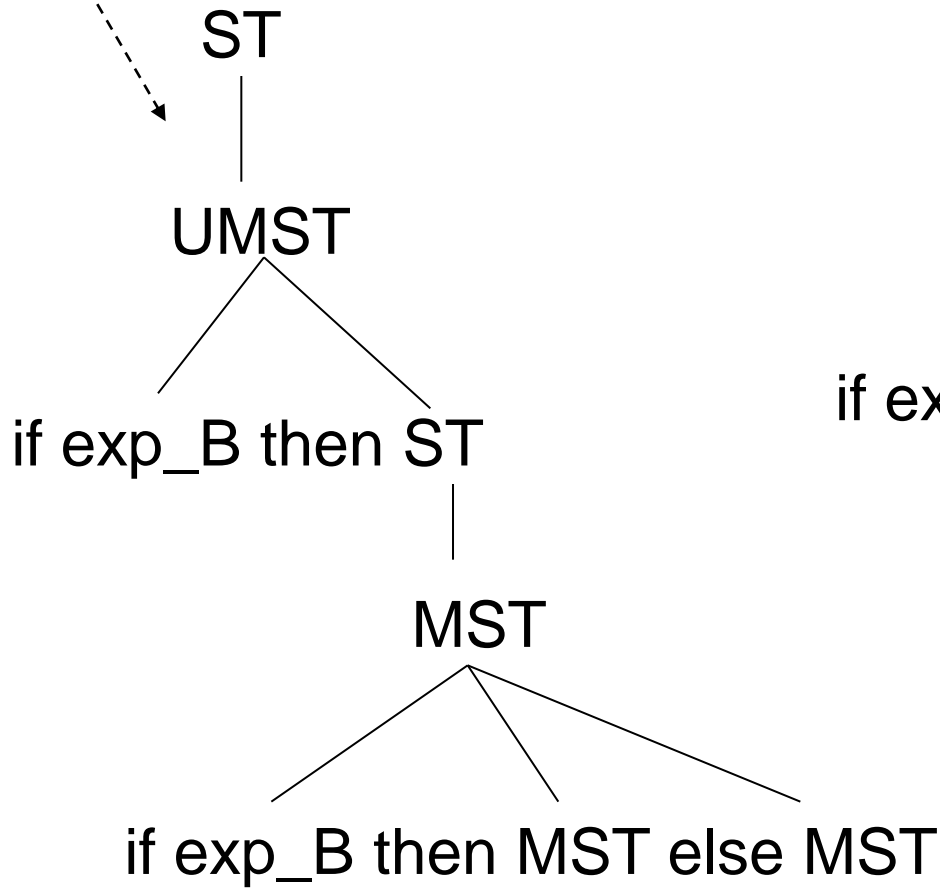
PROG ::= .... | **ST** | ....

**ST** ::= **MST** | **UMST** | ....

**MST** ::= if cond then **MST** else **MST**

**UMST** ::= if cond then **ST**

questa grammatica può produrre solo questo albero di derivazione e non questo





gli alberi di derivazione di questa grammatica  
sono conformi alla regola che associa ogni else al  
primo if-then alla sua sinistra

PROG

if cond then MST else MST

matched

qui non ci sono if-then senza else che rendano  
ambiguo a chi attaccare questo else

ma se volessi proprio

if then if then.... else.....

$MST ::= \text{if cond then } MST \text{ else } MST \mid \{UMST\}$

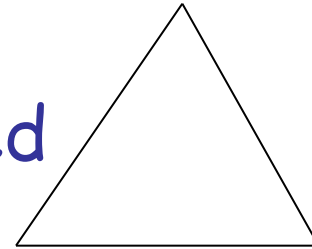
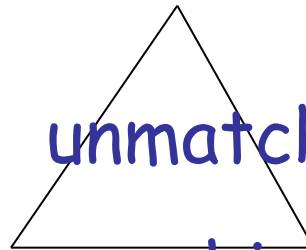
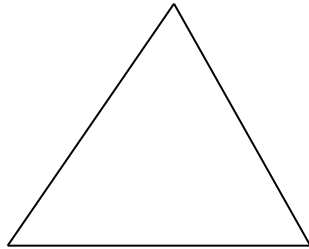
$UMST ::= \text{if cond then } ST$

PROG

NO



if cond then {UMST} else MST



unmatched

ma chiuso

da {...}

esiste una grammatica **non ambigua** che genera espressioni con 2 operatori (+ e \*) a precedenza diversa e che associano entrambi a sinistra e tale che **i suoi alberi di derivazione riflettono esattamente queste proprietà:**

$$e ::= e + t \mid t$$
$$t ::= t * f \mid f$$
$$f ::= n \mid (e)$$

## esercizio

costruire una grammatica che generi espressioni che possono contenere 3 operatori binari (+, \* e ^) con precedenza diversa (^ ha precedenza su \* che precede +) tali che + e \* associano a sinistra e ^ a destra

la grammatica deve essere non ambigua ed i suoi alberi di derivazione devono rispettare le proprietà degli operatori