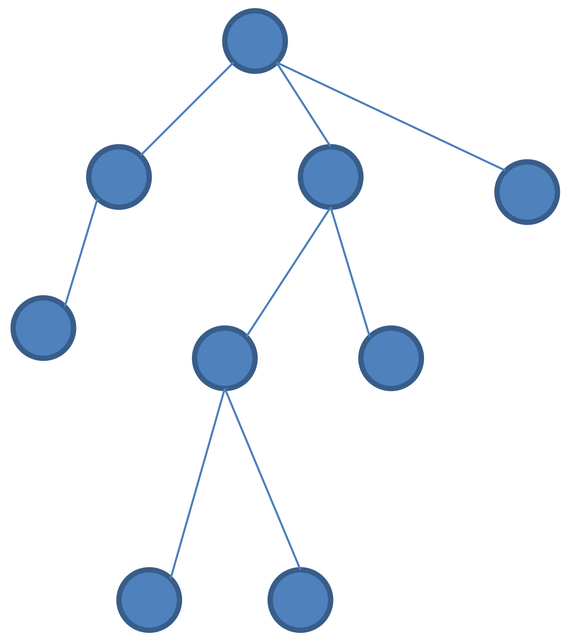
Lezione 2

alberi sintattici



radice
nodo interno
foglia

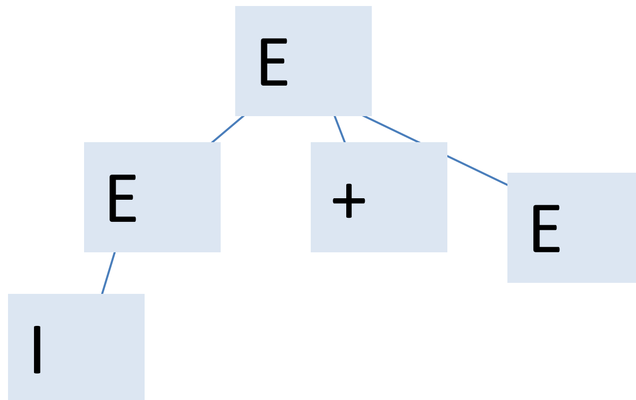
padre
figli ordinati
discendente
frontiera

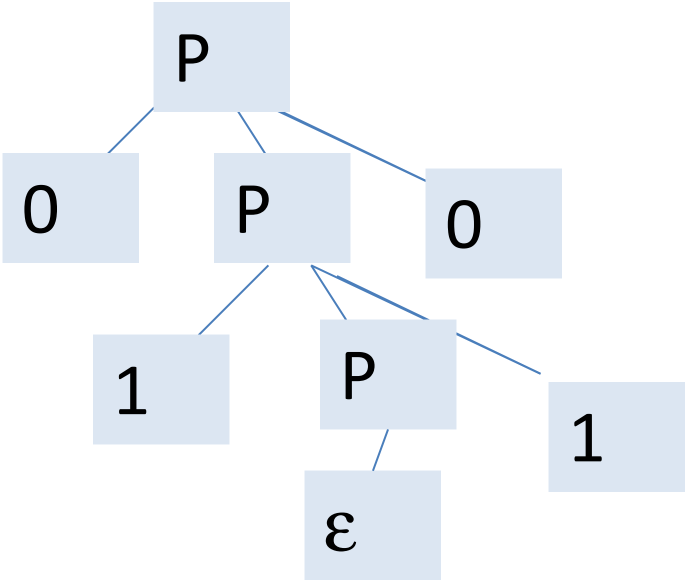
alberi sintattici

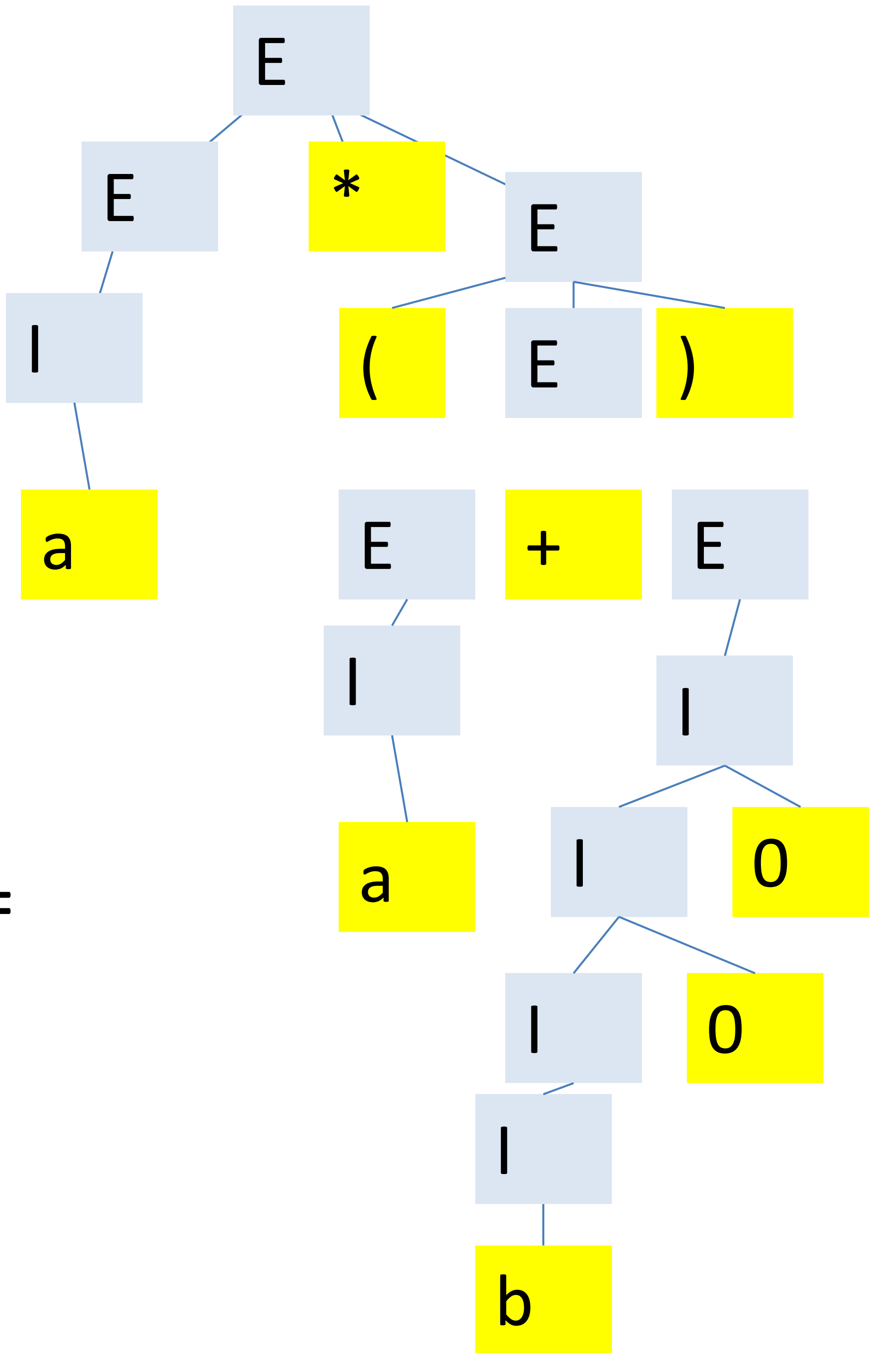
data G=(V,T,P,S)

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 X1...X2, allora A -> X1...Xn è in P.


E
E
+
E
I


P
0
P
0
1
P
1
ε



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 =>* w
--A =>*lm w
--A =>*rm w
--esiste albero sintattico con radice A e prodotto w

sono tutte equivalenti


derivazione a destra
albero sintattico
inferenza ricorsiva
derivazione a sinistra
derivazione

Teorema 5.12 Sia G=(V,T,P,S) una CFG. Se la procedura di inferenza ricorsiva indica che la stringa terminale w è nel linguaggio della variabile A, allora esiste un albero sintattico con radice A e prodotto w.

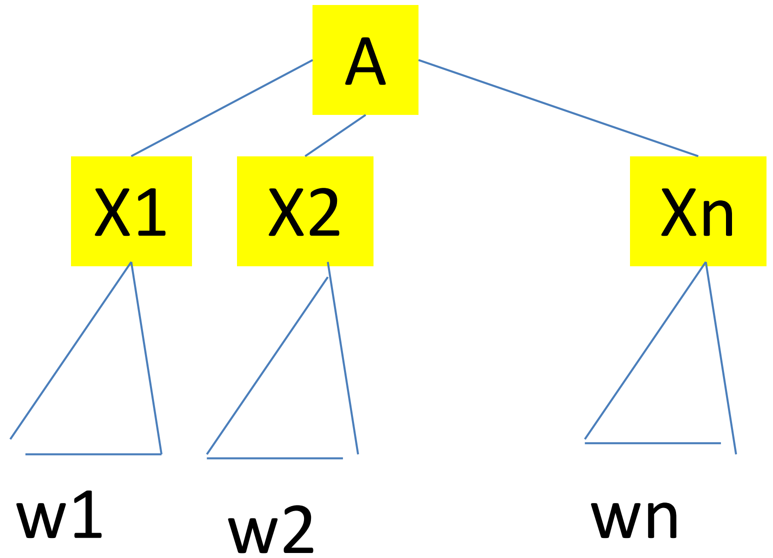
Dimostrazione per induzione sul numero di passi dell'inferenza ricorsiva.
Base: 1 passo
deve esistere in P, A -> w e quindi esiste anche l'albero sintattico desiderato.

passo induttivo: supponiamo di aver usato n1 passi di inferenza, e ce l’enunciato sia valido per tutte le stringhe x e variabili B tali che l’appartenenza di x al linguaggio di B sia deducibile in n passi di inferenza.

Consideriamo l’ultimo passo dell’inferenza di w. Questo passo usa una produzione, sia A -> X1X2..Xn, dove ogni Xi è una variabile o un terminale. possiamo scomporre w=w1w2..wn in modo che:

i) se Xi è terminale allora Xi=wi
ii) se Xi è variabile allora wi appartiene al ling. di Xi in al più n passi

applichiamo l'ipotesi induttiva a wi e Xi =>
esiste un albero sintattico con radice Xi e
prodotto wi.
=>
costruiamo un albero sintattico:



con radice A e prodotto w1w2...wn=w

dagli alberi alle derivazioni

serve osservazione:
nella grammatica delle espressioni,
E=>I=>Ib=>ab

e quindi per ogni coppia di stringhe α e β,
α E β=>α I β=>α Ib β => α ab β

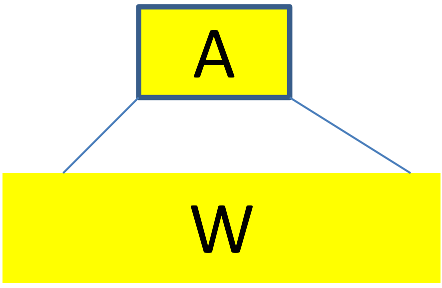
E+(E)=> E+(I)=>E+(Ib)=>E+(ab)

insomma la derivazione è libera dal contesto α, β

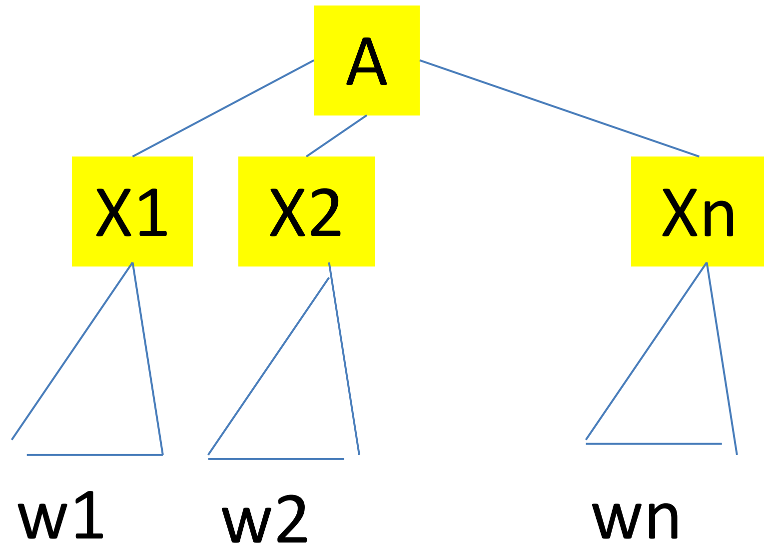
Teorema 5.14. Sia G=(V,T,P,S) una CFG e supponiamo che esista un albero sintattico con radice A e prodotto w in T*. Allora esiste una derivazione a sinistra A=>*lm w

Dimostrazione: induzione sull’altezza dell’albero.

Base: altezza 1, consiste di 1 sola produzione di G, ovvio che A=>lm w



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



quindi P ha la produzione A->X1X2..Xn e,
--se Xi è terminale allora wi=Xi
--se Xi è variabile è radic di un sottoalbero di altezza <n e prodotto wi,

per ipotesi induttiva Xi =>*lm wi
e quindi per la libertà di contesto:

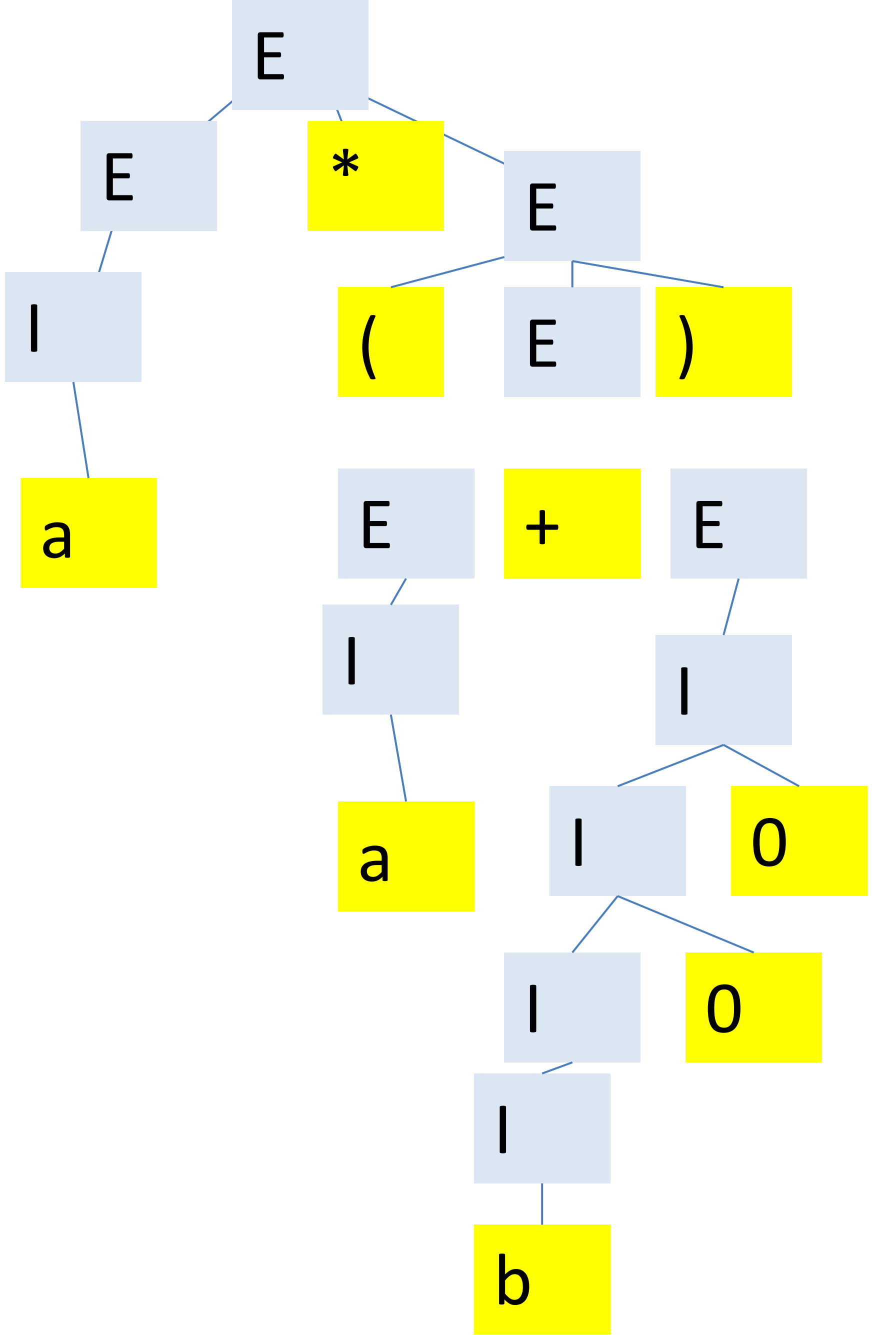
A=>*lm X1X2...Xn =>*lm w1X2..Xn =>*lm w1w2...Xn

rispettando l'ordine leftmost

formalmente serve un'induzione su i in [1..n] d1: E=>lm I=>lm a

d2: E=>lm (E)
=>lm (E+E) =>lm (I+E)
=>lm (a+E) =>lm (a+I)
=>lm (a+I0) =>lm (a+I00)
=> lm (a+b00)

da cui
A =>lm E*E=>lm d1 * E
=> lm d1 * d2



è ovvio che esiste un analogo teorema per le derivazioni destre

Manca un passo per chiudere il cerchio:
dalle derivazioni alle inferenze ricorsive

osserviamo che : A =>lm X1X2..Xn =>*lm w1w2..wn=w
è facile estrarre derivazioni
Xi =>*lm wi

osserva che se wi=a allora la derivazione è a=>*lm a

Teorema 5.18
Sia G=(V,T,P,S) e supponiamo che sistauna derivazione lm A =>*lm w con w in T*.Allora esiste una inferenza ricorsiva che determina che w è nel linguaggio di A

Dimostrazione: per induzione sulla lunghezza della derivazione A =>*lm w

Base: lunghezza 1, esiste A-> w in P con w solo i terminali, allora w è nel linguaggio di A

passo induttivo: supponiamo che sista una derivazione lm di n+1 passi e che l'enunciato valga per ogni derivazione di n o meno passi.

La derivazione comincia con una produzione di ,

A = >X1X2...Xn =>*lm w

allora possiamo scomporre w = w1w2..wn in moodo tale che

--se Xi è terminale allora , allora Xi=>*lm wiwi=Xi

--se Xi è variabile allora Xi =>*lm wi ed avrà al più n passi

per potesi induttiva, esiste un'inferenza induttiva che prova che wi è nel linguaggio di Xi,

da A -> X1X2...Xn si dimostra che w è nel linguaggio di A

esercizi 5.1.2 e 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 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 -> | 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 ad programmi, DTD eccetera

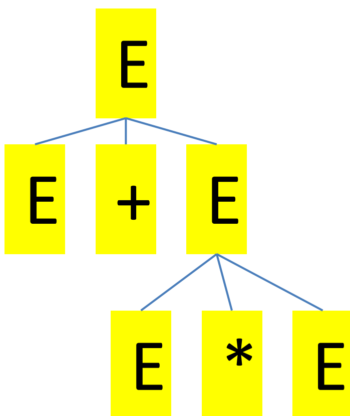
ma è una struttura univoca?

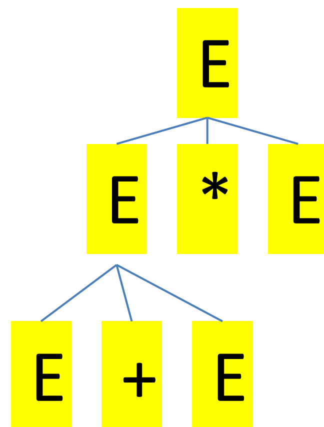
non sempre

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:

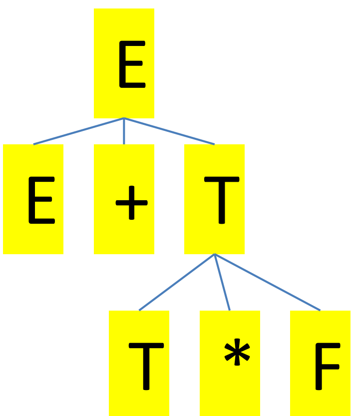
---per la precedenza basta introdurre una variabile per ogni livello di precedenza. Quelle che corrispondono a livelli di precedenza più bassi generano le altre.

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

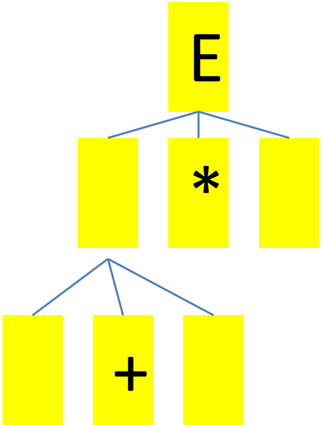
F -> I | (E)

T -> F | T * F

E -> T | E + T



SI



NO

ecco una grammatica che soddisfa la richiesta:

MST=Matched ST   UMST=UnMatched ST

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

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

MST::= if cond then MST else (M?)ST

UMST::= if cond then ST