

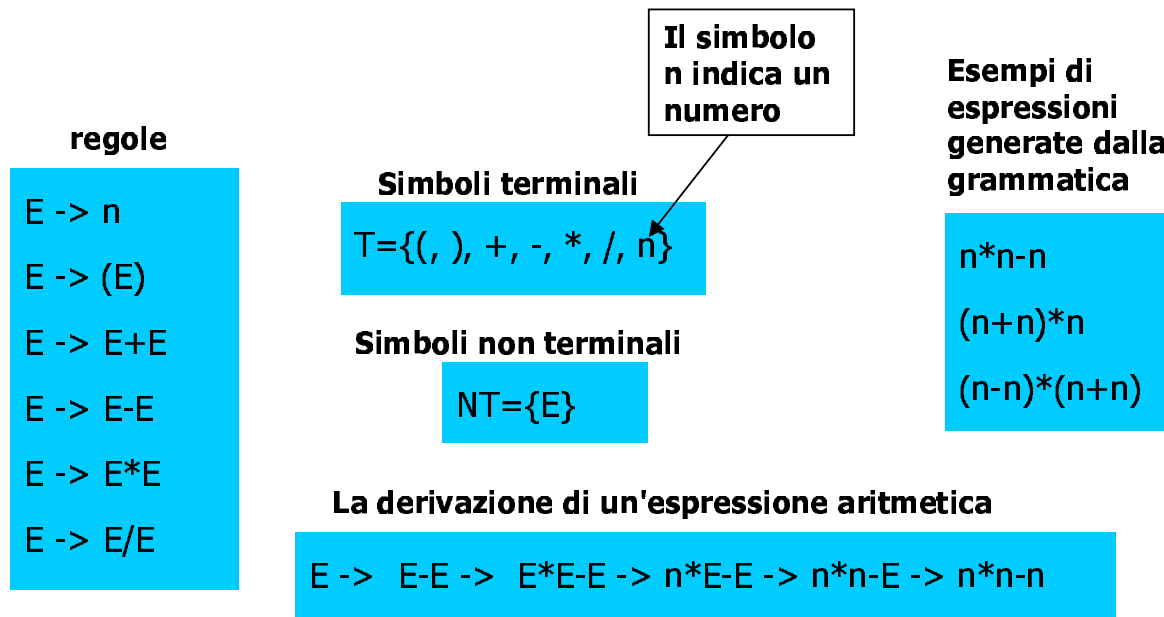
# Analisi Sintattica

## Le grammatiche libere da contesto

- Le **grammatiche libere** da contesto (context free) sono uno strumento con cui si possono definire i principali linguaggi di programmazione
- Una grammatica libera da contesto è definita da
  - Un insieme di **simboli terminali** che definiscono i simboli atomici del linguaggio
  - Un insieme di **simboli non terminali** (variabili) che definiscono le categorie sintattiche
  - Un insieme di **produzioni** che sono regole che definiscono come si possono generare le stringhe del linguaggio

# Le grammatiche libere da contesto: un esempio

- La seguente grammatica genera le espressioni aritmetiche



# Le grammatiche libere da contesto II

- Formalmente una grammatica libera da contesto è una quadrupla  $G = (V, T, P, S)$

- $V$  è un insieme di **simboli non terminali**
- $T$  un insieme di **simboli terminali**
- $P$  un insieme di **produzioni** del tipo  $A \rightarrow \alpha$ , dove  $A \in V$  e  $\alpha \in (V \cup T)^*$
- $S \in V$  è lo scopo o **simbolo di partenza**

# Le grammatiche libere da contesto III

## ■ Una **derivazione diretta**

$$\beta A \gamma \Rightarrow \beta \alpha \gamma$$

■ dove  $\beta, \alpha, \gamma \in (V \cup T)^*$  e  $A \rightarrow \alpha \in P$

## ■ Una **derivazione**

$$\alpha_1 \Rightarrow^* \alpha_n$$

■ dove  $\alpha_1, \dots, \alpha_n \in (V \cup T)^*$  e  $\alpha_1 \Rightarrow \alpha_2 \dots \alpha_{n-1} \Rightarrow \alpha_n$

## ■ Il **linguaggio $L(G)$ generato** da una grammatica

$$L(G) = \{\alpha \mid \alpha \in T^*, S \Rightarrow^* \alpha\}$$

## Un esempio

### ■ In generale determinare cosa genera una grammatica è **un problema difficile**

Una grammatica che genera tutte le stringhe con lo stesso numero di a e b

$$V = \{S, A, B\}$$

$$T = \{a, b\}$$

$$S \rightarrow aB \quad A \rightarrow bA$$

$$S \rightarrow bA \quad B \rightarrow b$$

$$A \rightarrow a \quad B \rightarrow bS$$

$$A \rightarrow aS \quad B \rightarrow aBB$$

Ipotesi

■  $S \rightarrow \alpha$  se e solo se  $\alpha$  ha lo stesso numero di a e b

■  $A \rightarrow \alpha$  se e solo se  $\alpha$  ha un numero di a maggiore di uno rispetto al numero dei b

■  $A \rightarrow \alpha$  se e solo se  $\alpha$  ha un numero di b maggiore di uno rispetto al numero degli a

Dimostrazione per induzione su  $|\alpha|$

■ se  $|\alpha| = 1$ , le uniche produzioni possibili sono  $A \rightarrow a$  e  $B \rightarrow b$

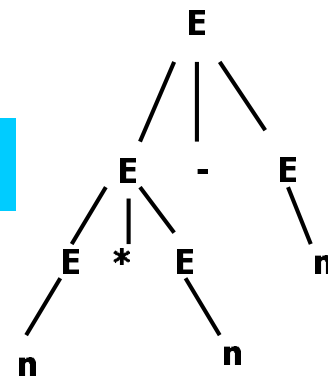
■ se  $|\alpha| > 1$ , si considera la prima produzione di  $A \Rightarrow^* \alpha$

■ se è  $S \rightarrow aB$ , allora  $aB$  contiene un a in più .....

# Alberi sintattici e derivazioni

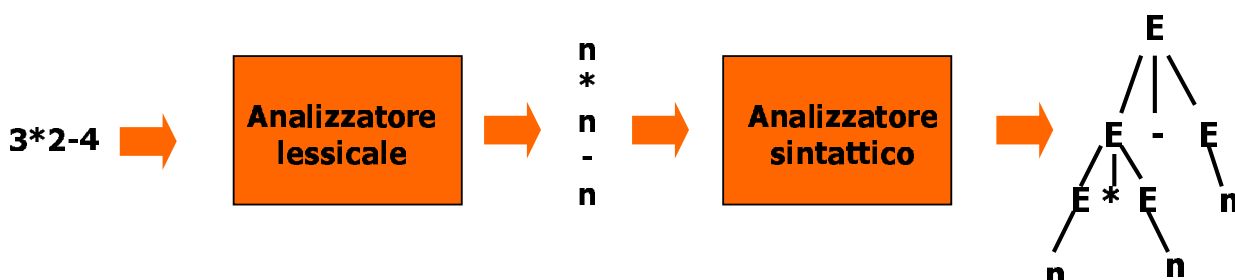
- Un **albero sintattico** (o albero di analisi) è una rappresentazione di una derivazione
- Ogni nodo corrisponde ad una produzione
- L'albero rappresenta quali produzioni sono state usate, ma **non l'ordine** in cui sono state usate

$E \rightarrow E-E \rightarrow E * E-E \rightarrow n * E-E \rightarrow n * n-E \rightarrow n * n-n$



## Come cooperano l'analizzatore lessicale e quello sintattico

- Nei comuni compilatori
  - l'analizzatore lessicale riconosce i **token** (simboli terminali) del linguaggio: numeri interi, reali, variabili, parole chiave, operatori, ...
  - l'analizzatore lessicale invia all'analizzatore sintattico la sequenza dei simboli terminali individuati
  - l'analizzatore sintattico analizza la sequenza di simboli terminali e produce l'albero sintattico



# Costruire un analizzatore sintattico dalla grammatica

- Come fa un analizzatore a decidere se una stringa  $w$  appartiene al linguaggio generato dalla grammatica  $G$  ?

- Analizzatori discendenti (top down)

- Partendo da simbolo  $S$  si tenta di ricostruire l'albero di derivazione con cui si ottiene  $w$

- Si usa le regole della grammatica in avanti

$S \rightarrow aABe \rightarrow aAbcBe \rightarrow abbcBe \rightarrow abbcde$

Prima il simbolo più a sinistra

- Analizzatori ascendenti (bottom up)

- Partendo da  $w$  si tenta di ricondursi a  $S$

- Si usa le regole della grammatica all'indietro

$abbcde \rightarrow aAbcde \rightarrow aAde \rightarrow aABe \rightarrow S$

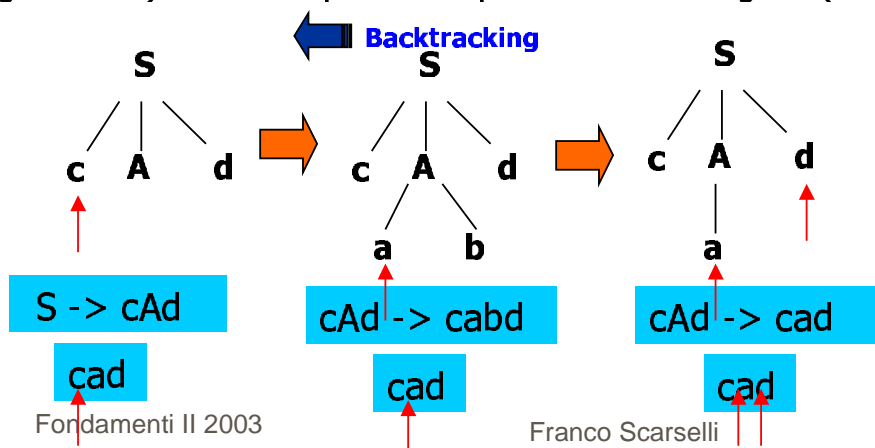
(1)  $S \rightarrow aABe$   
(2)  $A \rightarrow Abc|b$   
(3)  $B \rightarrow d$

$w=abbcde$

Prima il simbolo più a destra

## Analisi discendente

- Si applicano le regole in maniera ricorsiva verificando se la stringa generata è compatibile con quella da riconoscere
- Può essere necessario fare backtracking
- La computazione finisce quando la stringa è stata generata (stringa generata) o non è possibile provare altre regole (stringa rifiutata)

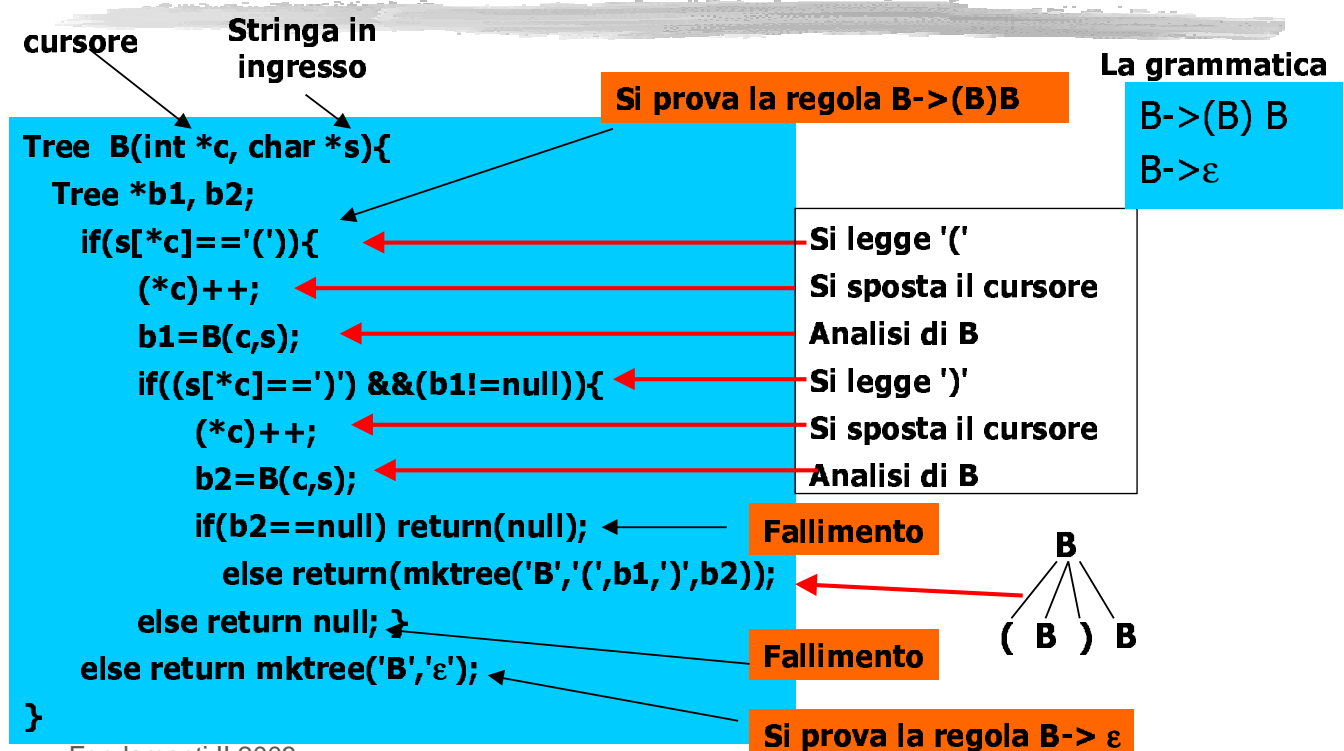


(1)  $S \rightarrow cAd$   
(2)  $A \rightarrow ab | a$   
 $w=cad$

# Analisi ricorsiva discendente

- Si tiene **traccia del prossimo simbolo** da sinistra della stringa in ingresso che deve essere generato dall'albero di analisi
  - serve a limitare l'insieme di produzioni che possono essere usate
- Si espandono i nodi **da sinistra a destra**
  - Un simbolo terminale soddisfa l'obiettivo se coincide con il prossimo simbolo in ingresso
  - Un simbolo non terminale soddisfa l'obiettivo se lo soddisfa un suo albero di analisi
- Quando espandendo una produzione e si genera un simbolo terminale **si verifica se corrisponde con quello in ingresso**
  - Sì - Si sposta il cursore in avanti e si prosegue
  - No - Si fallisce e si prova a soddisfare l'obiettivo con un altro albero

## Analisi ricorsiva discendente: un esempio



# Analizzatori predittivi

- Un **analizzatore predittivo** è un analizzatore che guardando il prossimo simbolo della stringa in ingresso è in grado di
  - prevedere quale regola deve essere usata ad ogni passo
  - evitare il backtracking

Il simbolo terminale in ingresso: a

Il simbolo non terminale: A

Un insieme di regole per A:  $A \rightarrow w_1 | w_2 | \dots | w_n$



Esiste una sola regola i t.c.

$A \rightarrow w_i \rightarrow a\alpha$

Un esempio: ad ogni passo conosco quale regola scegliere

$S \rightarrow aSc | bSc | d$   
 $w = abdcc$

$S \rightarrow aSc$

abdcc

$\rightarrow abSc$

abdcc

$\rightarrow abdcc$

abdcc

Fondamenti II 2003

Franco Scarselli

74

## Analizzatori predittivi II

- **Solo alcune grammatiche** permettono di realizzare analizzatori predittivi
- Occorre
  - evitare grammatiche **ambigue**
  - rimuovere la **ricorsione a sinistra**
  - **fattorizzare la grammatica a sinistra**

Come faccio a scegliere la regola giusta guardando solo il primo simbolo in ingresso?

$S \rightarrow aSc | aSd$   
 $w = aacd$

?

$S \rightarrow aSc$

$S \rightarrow aSd$

$S \rightarrow Sa | Sb$   
 $w = abb$

?

$S \rightarrow Sa$

$S \rightarrow Sb$

Fondamenti II 2003

Franco Scarselli

75

# Ambiguità

- Una grammatica si dice **ambigua** se ci sono due alberi di analisi che producono la stessa stringa

Una grammatica **ambigua** che produce le parentesi bilanciate

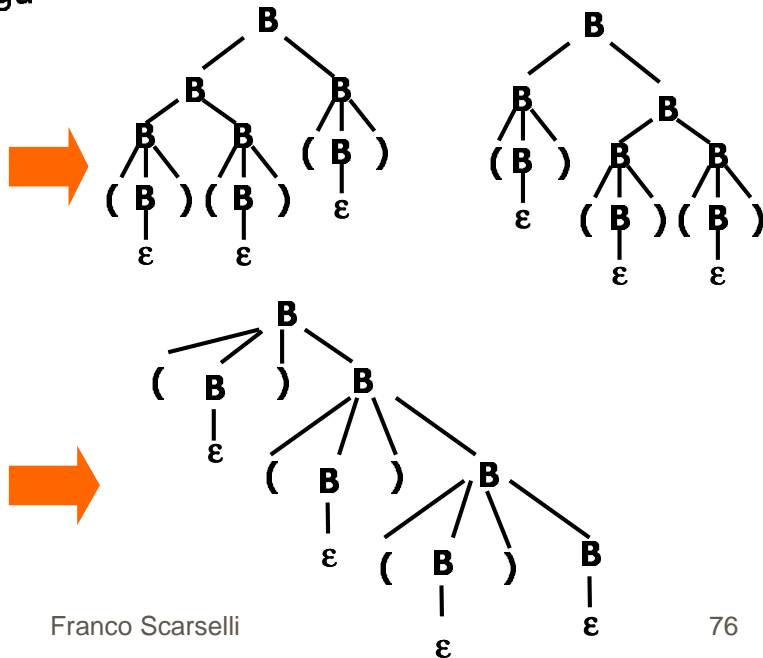
$B \rightarrow (B)$     $B \rightarrow BB$     $B \rightarrow \epsilon$

Esempi di parentesi bilanciate

$()()()$     $((()))$     $((()))()$

Una grammatica **non ambigua**

$B \rightarrow (B) B$     $B \rightarrow \epsilon$



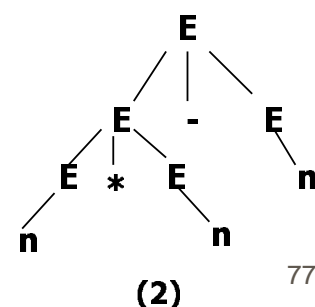
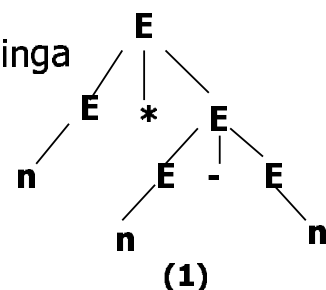
# Problemi dovuti all'ambiguità

## L'ambiguità

- non è possibile scegliere un'unica regola di derivazione
- può causare problemi nell'interpretazione di una stringa
- Nell'esempio sottostante usando (1) invece di (2) **non si rispetta la precedenza** fra gli operatori

(1)  $E \rightarrow E * E \rightarrow E * E - E \rightarrow n * E - E \rightarrow n * n - E \rightarrow n * n - n$

(2)  $E \rightarrow E - E \rightarrow E * E - E \rightarrow n * E - E \rightarrow n * n - E \rightarrow n * n - n$





# Eliminazione dell'ambiguità per le espressioni aritmetiche

- Per risolvere il problema della precedenza degli operatori occorre trovare un'altra grammatica non ambigua per lo stesso linguaggio
- Si introducono **nuovi simboli non terminali**
  - E (Espressione)
  - T (Termine)
  - F (Fattore)

## Grammatica non ambigua

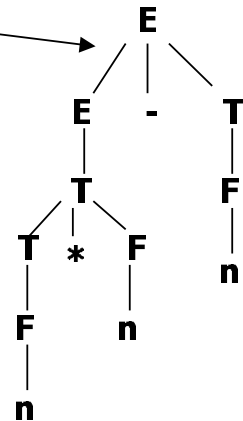
$E \rightarrow E+T \mid E-T \mid T$   
 $T \rightarrow T * F \mid T / F \mid F$   
 $F \rightarrow (E) \mid n$

Fondamenti II 2003

## Un esempio di derivazione

$E \rightarrow E-T \rightarrow T-T \rightarrow T * F-T$   
 $\rightarrow F * F-T \rightarrow n * F-T$   
 $\rightarrow n * n-T \rightarrow n * n-F \rightarrow n * n-n$

Franco Scarselli



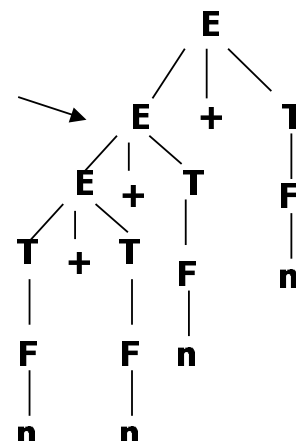
78

# La grammatica determina l'ordine di raggruppamento

- La nuova grammatica usa produzioni dove la variabile da sostituire appare a sinistra nella stringa prodotta (cioè, produzioni del tipo  $A \rightarrow Aw$ )
  - le espressioni saranno valutate **raggruppando a sinistra**

$E \rightarrow E+T \mid E-T \mid T$   
 $T \rightarrow T * F \mid T / F \mid F$   
 $F \rightarrow (E) \mid n$

## L'albero dell'espressione $n+n+n$



Fondamenti II 2003

Franco Scarselli

79

# Eliminazione dell'ambiguità nel caso dell' if-then-else

- La seguente grammatica che genera un linguaggio contenente il comando if-then-else è ambigua

**S -> if e then S**

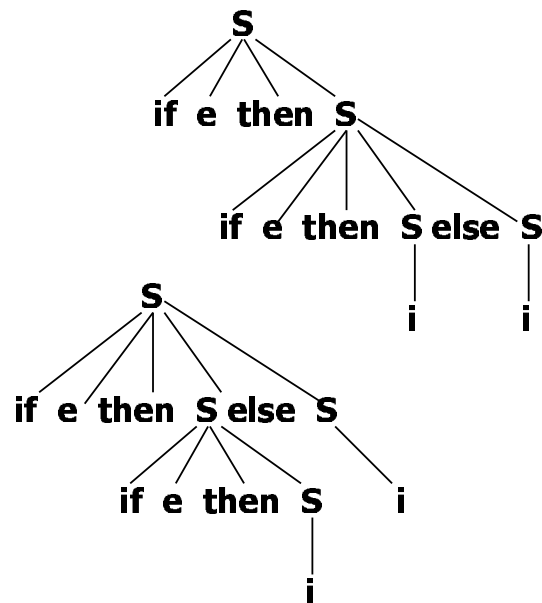
**S -> if e then S else S**

**S -> i**

Altre istruzioni

L'istruzione dopo else è legata al primo o il secondo if ?

**if e then if e then i else i**



# Eliminazione dell'ambiguità nel caso dell' if-then-else II

- Per eliminare l'ambiguità si introducono nuove variabili
  - S (uno statment)
  - I (un statment **if** incompleto, cioè senza **else**)
  - C (un statment **if** incompleto)
- La grammatica associa la parte else all'if più vicino (l'ultimo)

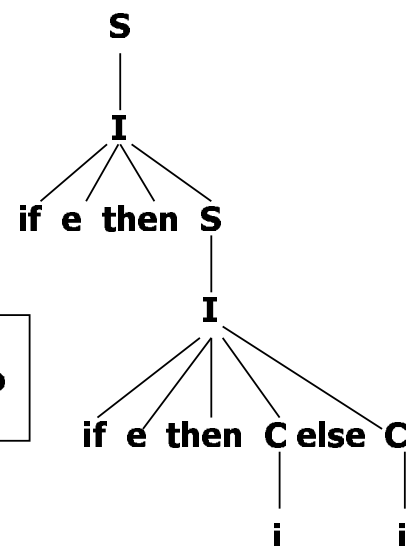
**S -> C | I**

**C -> if e then C else C | i**

**I -> if e then S |**

**if e then C else I**

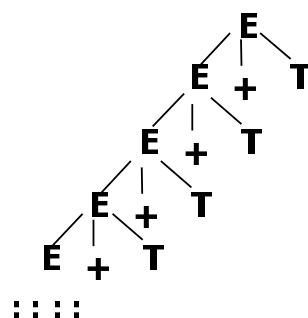
Fra un then e un else ci può essere solo uno statment completo



# Ricorsione a sinistra

- Una grammatica si dice **ricorsiva a sinistra** se
  - esiste una derivazione t.c.  $A \Rightarrow^* A\alpha$ , dove  $\alpha \in (V \cup T)^*$
- Nel caso più semplice la grammatica contiene una produzione  $A \rightarrow A\alpha$  questa viene detta **ricorsione immediata**
- Il problema è che non si capisce **quando fermare l'espansione** del non terminale

$E \rightarrow E+T \mid T$   
 $E \rightarrow T * F \mid F$   
 $F \rightarrow (E) \mid n$   
 $w = n+n+n$



Quando  
mi fermo ?

$E \rightarrow E+T \rightarrow E+E+T \rightarrow E+E+E+T \rightarrow \dots$

# Eliminare la ricorsione a sinistra

- Per eliminare la ricorsione sinistra immediata
  - Si trasforma la ricorsione da sinistra a destra
  - Occorre introdurre un nuovo simbolo non terminale

Non contengono  
A (a sinistra)

$A \rightarrow \beta_1 \mid \dots \mid \beta_m$   
 $A \rightarrow A \alpha_1 \mid \dots \mid A \alpha_n$



$A \rightarrow \beta_1 A' \mid \dots \mid \beta_m A'$   
 $A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$

# Eliminazione della ricorsione a sinistra: un esempio

- L'eliminazione della ricorsione nel caso delle espressioni aritmetiche

$E \rightarrow E+T \mid T$   
 $T \rightarrow T * F \mid F$   
 $F \rightarrow (E) \mid n$



$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$

- Si osservi che
  - nel caso in cui gli  $\alpha, \beta$  della produzione ( $A \rightarrow A \alpha \mid \beta$ ) contengano copie di  $A$ , può essere necessario iterare la trasformazione
  - abbiamo descritto come rimuovere la ricorsione immediata, ma esiste anche un **algoritmo per la ricorsione non immediata**

## Fattorizzazione a sinistra

- Una grammatica è **fattorizzabile a sinistra** se una variabile ha due produzioni che la espandono in stringhe con lo stesso prefisso

$$A \rightarrow \alpha \beta \mid \alpha \gamma$$

- Il problema è: come si espande  $A$  quando vedo  $\alpha$  ?
- Per fattorizzare a sinistra la grammatica
  - Si raggruppano le regole che generano lo stesso prefisso
  - Si introduce un nuovo simbolo per generare la parte destra della regola

$A \rightarrow \alpha \beta_1 \dots \mid \alpha \beta_m \mid \gamma$



$A \rightarrow \alpha A' \mid \gamma$   
 $A' \rightarrow \beta_1 \dots \mid \beta_m$

# Fattorizzazione a sinistra: un esempio

- Nel caso dell'if-then-else si introduce un nuovo non terminale E che rappresenta la parte **else**

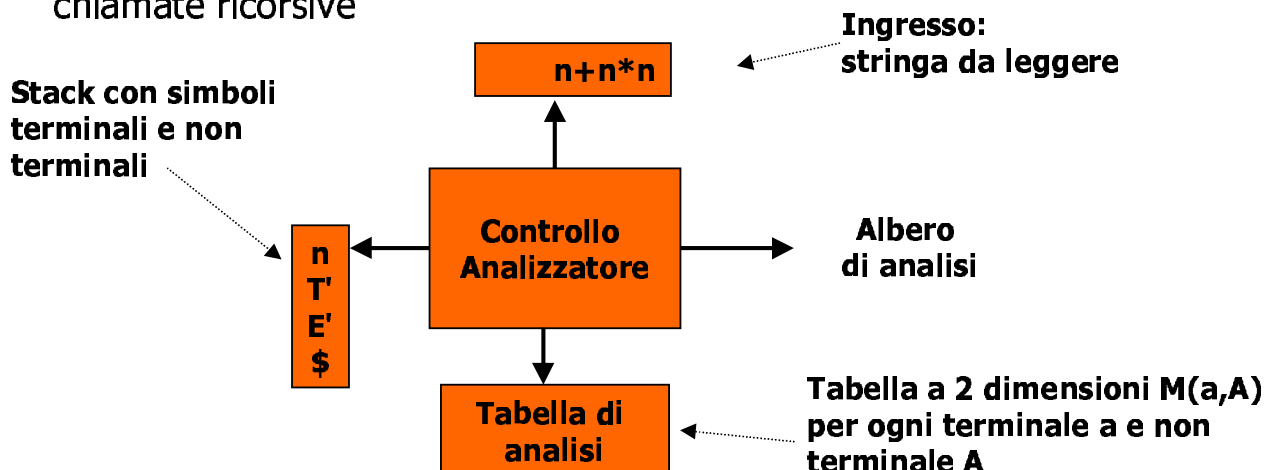
$S \rightarrow \text{if } e \text{ then } S$   
 $S \rightarrow \text{if } e \text{ then } S \text{ else } S$   
 $S \rightarrow i$



$S \rightarrow \text{if } e \text{ then } S E \mid i$   
 $E \rightarrow \text{else } S \mid \epsilon$

# Costruire un analizzatore predittivo

- Riassumendo occorre: Eliminare eventuali ambiguità, eliminare la ricorsione a sinistra, fattorizzare la grammatica
- Poi, per realizzare un analizzatore predittivo si usa uno stack invece di chiamate ricorsive



# Un esempio

## Analisi di $n+n*n$

Stack	Input	Produzioni
\$E	$n+n*n\$$	
\$E'T	$n+n*n\$$	$E \rightarrow TE'$
\$E'T'F	$n+n*n\$$	$T \rightarrow FT'$
\$E'T'n	$n+n*n\$$	$F \rightarrow n$
\$E'T'	$+n*n\$$	
\$E'	$+n*n\$$	$T' \rightarrow \epsilon$
\$E'T+	$+n*n\$$	$E' \rightarrow TE'$
\$E'T	$n*n\$$	

## Grammatica

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$

## Tabella di analisi

	n	+	*	(	)	\$
E	TE'			TE'		
E'		+TE'			$\epsilon$	$\epsilon$
T	FT'			FT'		
T'		$\epsilon$	*FT'		$\epsilon$	$\epsilon$
F	n			(n)		

# Un esempio II

## Analisi di $n+n*n$

Stack	Input	Produzioni
\$E'T'F	$n*n\$$	$T \rightarrow FT'$
\$E'T'n	$n*n\$$	$F \rightarrow n$
\$E'T'	$*n\$$	
\$E'T'F*	$*n\$$	$T \rightarrow *FT'$
\$E'T'F	$n\$$	
\$E'T'n	$n\$$	$F \rightarrow n$
\$E'T'	$\$$	
\$E'	$\$$	$T' \rightarrow \epsilon$
$\$$	$\$$	$E' \rightarrow \epsilon$

## Grammatica

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$

## Tabella di analisi

	n	+	*	(	)	\$
E	TE'			TE'		
E'		+TE'			$\epsilon$	$\epsilon$
T	FT'			FT'		
T'		$\epsilon$	*FT'		$\epsilon$	$\epsilon$
F	n			(n)		

# Costruzione della tabella di analisi

- Si costruiscono due funzioni FIRST e FOLLOW

■  $w \in (V \cup T)^*$ ,  $\text{FIRST}(w)$  = l'insieme dei simboli terminali che iniziano le stringhe derivate da  $w$   
(se  $w \Rightarrow^* abAXs \dots$  allora  $a \in \text{FIRST}(w)$  )

■  $A \in V$ ,  $\text{FOLLOW}(A)$  = l'insieme dei simboli terminali che possono seguire  $A$  in stringhe derivate da  $S$   
(se  $S \Rightarrow^* bcAas \dots$  allora  $a \in \text{FOLLOW}(A)$  )

## Calcolare FIRST

- **$\text{FIRST}(X)$ ,  $X \in (V \cup T)$**

1) se  $X$  è un terminale, allora  $\text{FIRST}(X) = \{X\}$

2) se  $X \rightarrow \varepsilon$  è una produzione, allora  $\varepsilon \in \text{FIRST}(X)$

3) se  $X \rightarrow Y_1 Y_2 \dots Y_n$  è una produzione,

■ se  $a \in \text{FIRST}(Y_i)$  e per ogni  $1 \leq j < i$  vale  $\varepsilon \in \text{FIRST}(Y_j)$  allora  $a \in \text{FIRST}(X)$   
( $Y_1 Y_2 \dots Y_{i-1} \Rightarrow^* \varepsilon$ )

■ se per ogni  $1 \leq j \leq n$  vale  $\varepsilon \in \text{FIRST}(Y_j)$  allora  $\varepsilon \in \text{FIRST}(X)$

- **$\text{FIRST}(w)$ ,  $w \in (V \cup T)^*$ , si calcola come in (3)**

■ Per costruire la tabella di analisi **occorre conoscere  $\text{FIRST}(w)$ , per ogni regola  $A \rightarrow w$**

- $\text{FIRST}(X)$ , viene calcolato ricorsivamente con le regole descritte sopra

# Calcolare FIRST: un esempio

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$



$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(E') = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(T') = \{ *, \epsilon \}$   
 $\text{FIRST}(F) = \{ (, n \}$



$\text{FIRST}(TE') = \text{FIRST}(T) = \{ (, n \}$   
 $\text{FIRST}(+TE') = \{ + \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}(FT') = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(*FT') = \{ * \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}((E)) = \{ ( \}$     $\text{FIRST}(n) = \{ n \}$

# Calcolare FIRST: un esempio II

$S \rightarrow ABCde \mid fAg$   
 $A \rightarrow ad \mid \epsilon$   
 $B \rightarrow b \mid \epsilon \mid C$   
 $C \rightarrow c \mid \epsilon \mid B$



$\text{FIRST}(S) = \{ f \} \cup \text{FIRST}(A) \cup \text{FIRST}(B)$   
 $\quad \cup \text{FIRST}(C) \cup \{ d \} = \{ f, a, b, c, d \}$   
 $\text{FIRST}(A) = \{ a, \epsilon \}$   
 $\text{FIRST}(B) = \{ b, \epsilon \} \cup \text{FIRST}(C) = \{ b, c, \epsilon \}$   
 $\text{FIRST}(C) = \{ c, \epsilon \} \cup \text{FIRST}(B) = \{ b, c, \epsilon \}$

$\text{FIRST}(ABCde) = \text{FIRST}(A) \cup \text{FIRST}(B)$   
 $\quad \cup \text{FIRST}(C) \cup \{ d \} = \{ a, b, c, d \}$     $\text{FIRST}(fAg) = \{ f \}$   
 $\text{FIRST}(ad) = \{ a \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}(b) = \{ b \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$     $\text{FIRST}(C) = \{ b, c, \epsilon \}$   
 $\text{FIRST}(c) = \{ c \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$     $\text{FIRST}(B) = \{ b, c, \epsilon \}$



# Calcolare FOLLOW

## ■ FOLLOWS(X)

- $\$ \in \text{FOLLOW}(S)$ , dove  $\$$  rappresenta la fine della stringa
- se  $A \rightarrow \alpha B \beta$  è una produzione, allora  $\text{FIRST}(\beta) \setminus \{\epsilon\} \subseteq \text{FOLLOW}(B)$   
(B è seguito dai primi caratteri di  $\beta$ )
- se  $A \rightarrow \alpha B$  è una produzione oppure se  $A \rightarrow \alpha B \beta$  è una produzione e se  $\epsilon \in \text{FIRST}(\beta)$ , allora  $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$   
(se nella produzione non compare niente a destra di B allora B è seguito da ciò che segue A)

- Prima occorre calcolare FIRST, poi FOLLOW viene calcolato applicando ricorsivamente le regole descritte sopra

## Calcolare FOLLOW: un esempio

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$



$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(E') = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(T') = \{ *, \epsilon \}$   
 $\text{FIRST}(F) = \{ (, n \}$



$\text{FOLLOW}(E) = \{ \$ \} \cup \{ ) \} = \{ \$, ) \}$   
 $\text{FOLLOW}(E') = \text{FOLLOW}(E) = \{ \$, ) \}$   
 $\text{FOLLOW}(T) = \text{FIRST}(E') \cup \text{FOLLOW}(E') = \{ +, \$, ) \}$   
 $\text{FOLLOW}(T') = \text{FOLLOW}(T) = \{ +, \$, ) \}$   
 $\text{FOLLOW}(F) = \text{FIRST}(T') \cup \text{FOLLOW}(T) \cup \text{FOLLOW}(T')$   
 $\text{FOLLOW}(T') = \{ *, +, \$, ) \}$

# Calcolare FOLLOW: un esempio II

$S \rightarrow ABCde \mid fAg$

$A \rightarrow ad \mid \varepsilon$

$B \rightarrow b \mid \varepsilon \mid C$

$C \rightarrow c \mid \varepsilon \mid B$



$FIRST(S) = \{f\} \cup FIRST(A) \cup FIRST(B) \cup FIRST(C) \cup \{d\} = \{f, a, b, c, d\}$

$FIRST(A) = \{a, \varepsilon\}$

$FIRST(B) = \{b, \varepsilon\} \cup FIRST(C) = \{b, c, \varepsilon\}$

$FIRST(C) = \{c, \varepsilon\} \cup FIRST(B) = \{b, c, \varepsilon\}$



$FOLLOW(S) = \{\$ \}$

$FOLLOW(A) = FIRST(B) \cup FIRST(C) \cup \{d, g\} = \{b, c, d, g\}$

$FOLLOW(B) = FIRST(C) \cup \{d\} \cup FOLLOW(C) = \{b, c, d\}$

$FOLLOW(C) = \{d\} \cup FOLLOW(B) = \{b, c, d\}$

## Costruire la tabella di analisi

■ Per ogni produzione  $A \rightarrow w$  si **calcola FOLLOW(A) e FIRST(w)**

■ La tabella  $M[A, a]$  **è definita** da

■ se  $A \rightarrow w$  e  $a \in FIRST(w)$ , allora  $M[A, a] = \{A \rightarrow w\}$

(se vedo  $a$  in ingresso e  $A$  è il prossimo simbolo sullo stack, allora espando  $A \rightarrow w$ )

■ se  $A \rightarrow w$ ,  $\varepsilon \in FIRST(w)$  e  $a \in FOLLOW(A)$ , allora  $M[A, a] = \{A \rightarrow \varepsilon\}$

(se vedo  $a$  in ingresso e  $A$  è il prossimo simbolo sullo stack, allora espando  $A \rightarrow \varepsilon$ )

# Costruire la tabella di analisi: un esempio

$\text{FIRST}(TE') = \text{FIRST}(T) = \{ (, n \}$   
 $\text{FIRST}(+TE') = \{ + \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}(FT') = \text{FIRST}(F) = \{ (, n \}$   
 $\text{FIRST}(*FT') = \{ * \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}((E)) = \{ ( \}$     $\text{FIRST}(n) = \{ n \}$

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid n$

$\text{FOLLOW}(E) = \{ \$ \} \cup \{ ) \} = \{ \$, ) \}$   
 $\text{FOLLOW}(E') = \text{FOLLOW}(E) = \{ \$, ) \}$   
 $\text{FOLLOW}(T) = \text{FIRST}(E') \cup \text{FOLLOW}(E) = \{ +, \$, ) \}$   
 $\text{FOLLOW}(T') = \text{FOLLOW}(T) = \{ +, \$, ) \}$   
 $\text{FOLLOW}(F) = \text{FIRST}(T') \cup \text{FOLLOW}(T) \cup \text{FOLLOW}(E) = \{ +, \$, ) \}$   
 $\text{FOLLOW}(T') = \{ *, +, \$, ) \}$

	n	+	*	(	)	\$
E	TE'			TE'		
E'		+TE'			$\epsilon$	$\epsilon$
T	FT'			FT'		
T'		$\epsilon$	*FT'		$\epsilon$	$\epsilon$
F	n			(E)		

# Costruire la tabella di analisi: un esempio II

$\text{FIRST}(ABCde) = \{ a, b, c, d \}$     $\text{FIRST}(fAg) = \{ f \}$   
 $\text{FIRST}(ad) = \{ a \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$   
 $\text{FIRST}(b) = \{ b \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$     $\text{FIRST}(C) = \{ b, c, \epsilon \}$   
 $\text{FIRST}(c) = \{ c \}$     $\text{FIRST}(\epsilon) = \{ \epsilon \}$     $\text{FIRST}(B) = \{ b, c, \epsilon \}$

$S \rightarrow ABCde \mid fAg$   
 $A \rightarrow ad \mid \epsilon$   
 $B \rightarrow b \mid \epsilon \mid C$   
 $C \rightarrow c \mid \epsilon \mid B$

$\text{FOLLOW}(S) = \{ \$ \}$   
 $\text{FOLLOW}(A) = \{ b, c, d, g \}$   
 $\text{FOLLOW}(B) = \{ b, c, d \}$   
 $\text{FOLLOW}(C) = \{ b, c, d \}$

	a	b	c	d	e	f	g
S	ABCDe	ABCDe	ABCDe	ABCDe		fAg	
A	ad	$\epsilon$	$\epsilon$	$\epsilon$			$\epsilon$
B		b, C,	C,				
		$\epsilon$	$\epsilon$	$\epsilon$			
C		B	c, B				
		$\epsilon$	$\epsilon$	$\epsilon$			

Più regole sono applicabili

# Analizzatori discendenti e grammatiche LL(1)

- Una grammatica la cui tabella di analisi per un analizzatore discendente non ha definizioni multiple si dice **LL(1)**
  - Left to right (scansione dell'ingrasso)
  - leftmost (si espande il simbolo più a sinistra)
  - 1 simbolo per la previsione
- Non tutte le grammatiche sono LL(1)
  - es. grammatiche ambigue e non fattorizzate a sinistra **non sono LL(1)**
- Occorre
  - Eliminare l'ambiguità
  - eliminare la ricorsione a sinistra
  - Fattorizzare a sinistra
  - ma, **non è detto che si ottenga una grammatica LL(1)**