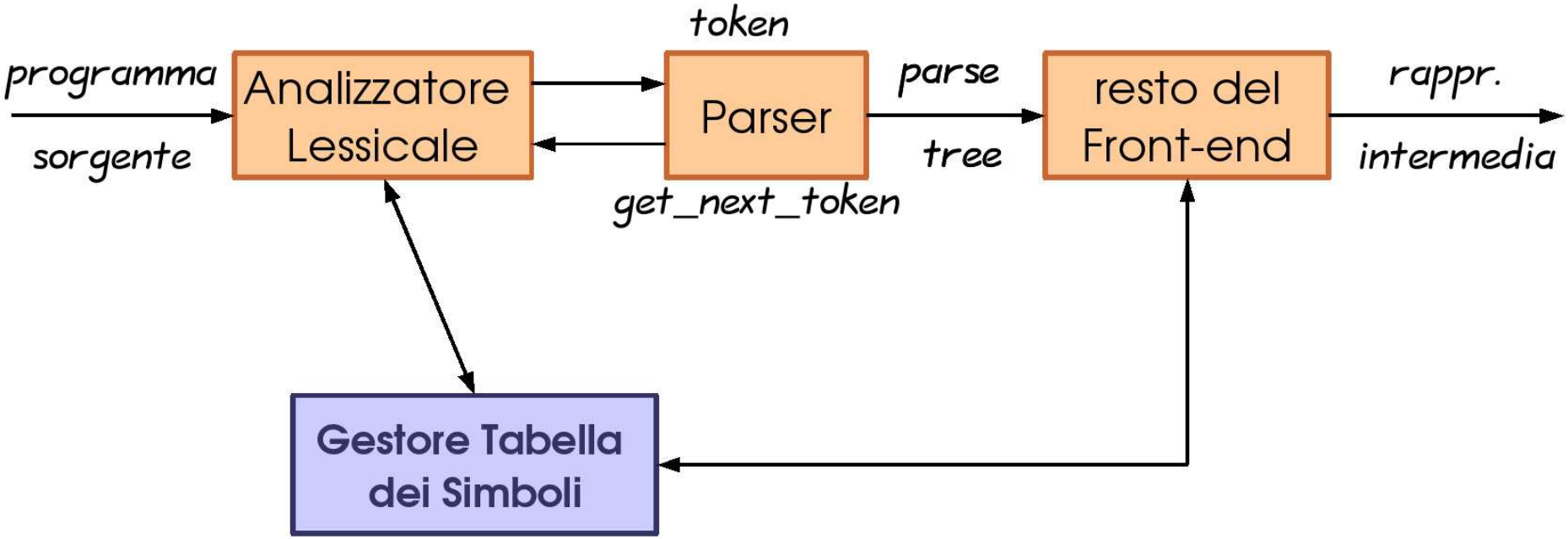
Analisi Sintattica

Maria Rita Di Berardini[1]

[1]Dipartimento di Matematica e Informatica
Universitá di Camerino
mariarita.diberardini@unicam.it

Ruolo dell'analisi sintattica



Tipologie di parsing

- Parsing Universali: consentono di effettuare il parsing di una qualsiasi grammatica (piuttosto inefficienti)
- Top-Down Parsing: costruiscono il parse tree dall'alto (radice) verso il basso (foglie)
- Bottom-Up Parsing: costruiscono il parse tree dal basso verso l'alto
- Top-Down e Bottom-Up Parsing
  - funzionano solo per specifiche sottoclassi di grammatiche (LL e LR)
  - sono sufficientemente espressivi per descrivere la maggior parte dei costrutti dei linguaggi di programmazione
  - per le grammatiche LR esistono dei generatori automatici
  - sono più efficienti

Part I

Grammatiche Libere da Contesto

Grammatiche libere da contesto: definizione

Una grammatica libera da contesto è una tupla $G = \langle \Sigma, V, S, P \rangle$ dove:

1. Σ è un **alfabeto** finito di simboli (detti simboli **terminali**)
2. V è un alfabeto finito di simboli che rappresentano categorie sintattiche (detti anche simboli **non terminali**)
3. $S \in V$ è il simbolo non terminale iniziale
4. P è un insieme finito di **produzioni**, i.e. regole delle forma

$$A \rightarrow X_1 X_2 \ldots X_n$$

dove:

- A è un simbolo non terminale ($A \in V$) detto la **testa** o **parte sinistra** della produzione
- per ogni $j = 1, 2, \ldots, n$, $X_j \in (V \cup \Sigma)$
- la stringa $X_1 X_2 \ldots X_n \in (V \cup \Sigma)^*$ è detta **corpo** o **parte destra** della produzione; può anche essere la stringa vuota ($A \rightarrow \varepsilon$)

Notazioni

- Terminali
  - lettere minuscole dell'alfabeto: $a, b, c, \ldots, a', b', c, \ldots$
  - simboli di operatori: $+, -, *, \ldots$
  - simboli di punteggiatura
  - cifre $0 \ldots 9$
  - stringhe in grassetto: **if**, **id**, ...
- Non terminali: lettere maiuscole $A, B, C, \ldots, X, Y, Z$
- Stringhe in Σ^* : lettere minuscole $u, v, w, x, y, z, \ldots$
- Stringhe in $(V \cup \Sigma)^*$: lettere minuscole $\alpha, \beta, \gamma, \ldots, \alpha_1, \beta_1, \gamma_1, \ldots$
- Se $A \to \alpha_1$, $A \to \alpha_2, \ldots, A \to \alpha_n$ sono tutte le produzioni per un dato non terminale A possiamo usare la notazione equivalente

$$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

Alberi di derivazione

Sia $G = \langle \Sigma, V, S, P \rangle$ una grammatica libera da contesto

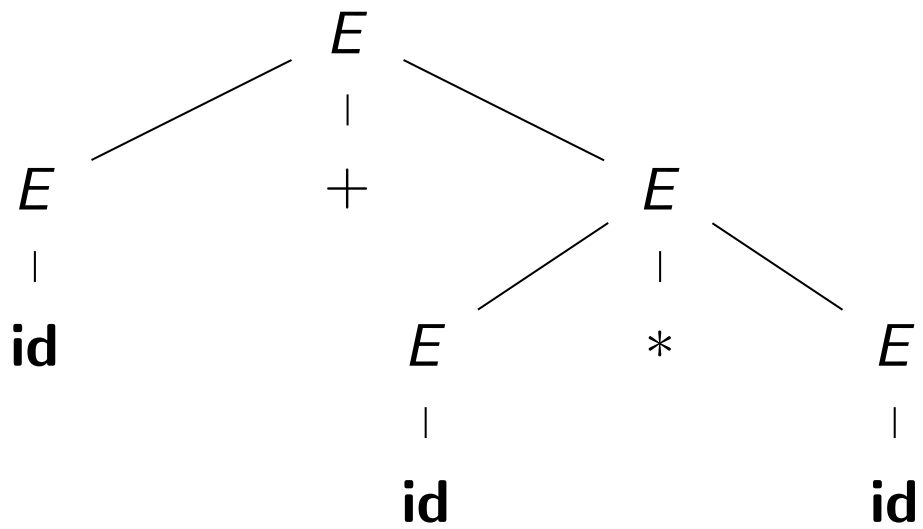
- Un albero di derivazione (**parse tree**) di G è un albero i cui nodi sono etichettati con simboli in $V \cup \Sigma$ e tale che:
  - la radice è etichettata con il simbolo S (simbolo iniziale di G)
  - ogni foglia è etichettata con un terminale
  - ogni nodo interno è etichettato con un non terminale A ed i suoi figli, presi da sinistra verso destra, sono etichettati con i simboli X_1, X_2, ..., X_k della parte destra di una qualche produzione $A \longrightarrow X_1 X_2 \ldots X_k$ per A
- La stringa che si ottiene concatenando i simboli associati alle foglie è detta **stringa associata** all'albero di derivazione
- Un albero è un albero di derivazione per una stringa w se w è la sua stringa associata

Un esempio

Sia $G = \langle \Sigma = \{+, *, (,), -, \mathbf{id}\}, V = \{E\}, S = E, P \rangle$ dove l'insieme P delle produzioni è definito da:

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

Il seguente è un albero di derivazione della stringa è $w = \mathbf{id} + \mathbf{id} * \mathbf{id}$



Linguaggio generato: definizione

Sia $G = \langle \Sigma, V =, S, P \rangle$ una grammatica libera da contesto:

- Il **linguaggio generato** da G denotato da $L(G)$ è l'insieme di tutte le stringhe di terminali $w \in \Sigma^*$ tali che esiste un albero di derivazione la cui stringa associata è w
- Possiamo dare una definizione alternativa introducendo il concetto di derivazione di una stringa a partire da una categoria sintattica
- Intuitivamente, una derivazione è una sequenza (eventualmente vuota) di passi di derivazione
- Un passo di derivazione, denotato con $\Rightarrow_G$, consiste nella riscrittura di un qualche non terminale A con la parte destra di una produzione per A
- Formalmente, sia $\alpha \in (V \cup \Sigma)^*$ contenente almeno un non terminale A ed $A \rightarrow X_1 X_2 \dots X_k \in P$, allora possiamo riscrivere la stringa:

$$\alpha = \beta A \gamma \Rightarrow_G \beta X_1 X_2 \dots X_k \gamma$$

Linguaggio generato: definizione

Sia $G = \langle \Sigma, V =, S, P \rangle$ una grammatica libera da contesto:

- Una derivazione di una stringa $w \in \Sigma^*$ a partire dal simbolo iniziale S è una sequenza

$$S = \alpha_0 \Rightarrow_G \alpha_1 \Rightarrow_G \alpha_2 \ldots \Rightarrow_G \alpha_n = w$$

  dove, per ogni $j = 0, \ldots, n-1$, $\alpha_j \Rightarrow_G \alpha_{j+1}$ è un passo di derivazione
- Se $j < n$, allora $\alpha_j \in (\Sigma \cup V)^*$ generata a partire da S mediante un certo numero di passi di derivazione (**forma sentenziale**)
- Se n=0, allora la derivazione è composta dal solo simbolo S
- $S \stackrel{*}{\Rightarrow}_G \alpha$: la stringa α è derivata da S mediante zero o più passi di derivazione
- $S \stackrel{+}{\Rightarrow}_G \alpha$: la stringa α è derivata da S mediante uno o più passi di derivazione

Linguaggio generato: definizione

Sia $G = \langle \Sigma, V =, S, P \rangle$ una grammatica libera da contesto:

- Il **linguaggio generato** da G è definito come l'insieme

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow}_G w\}$$

- L'insieme delle stringhe ottenute tramite derivazioni ùguale a quello definito tramite alberi di derivazione
- Ad ogni derivazione possiamo associare un albero di derivazione e viceversa

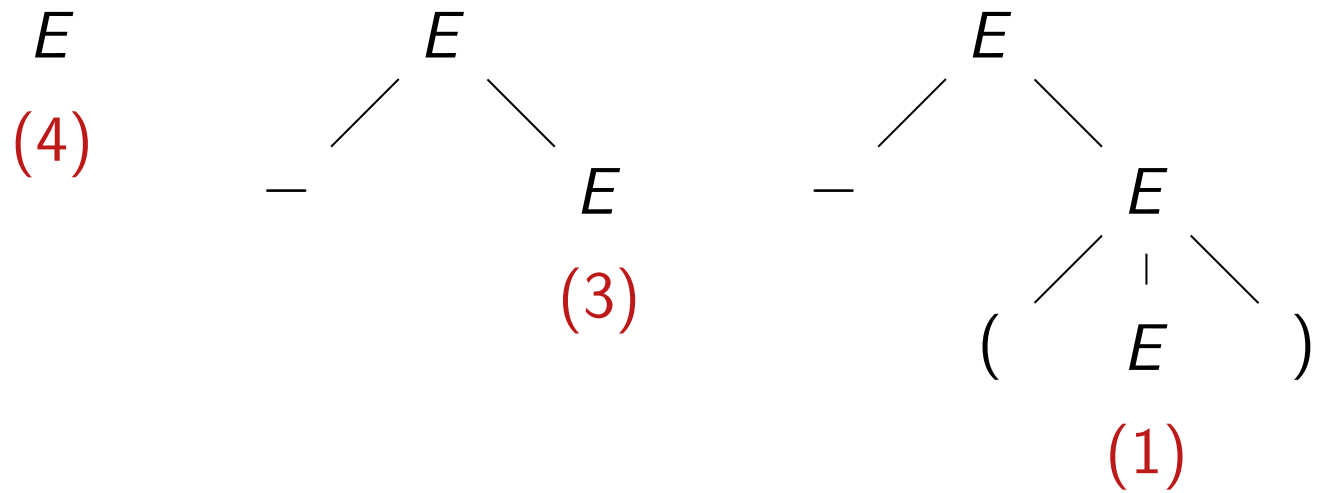
Un esempio

Sia G la grammatica definita da:

$$E \to E + E \ (1) \mid E * E \ (2) \mid (E) \ (3) \mid \ - E \ (4) \mid \mathbf{id} \ (5)$$

e consideriamo la seguente derivazione

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$



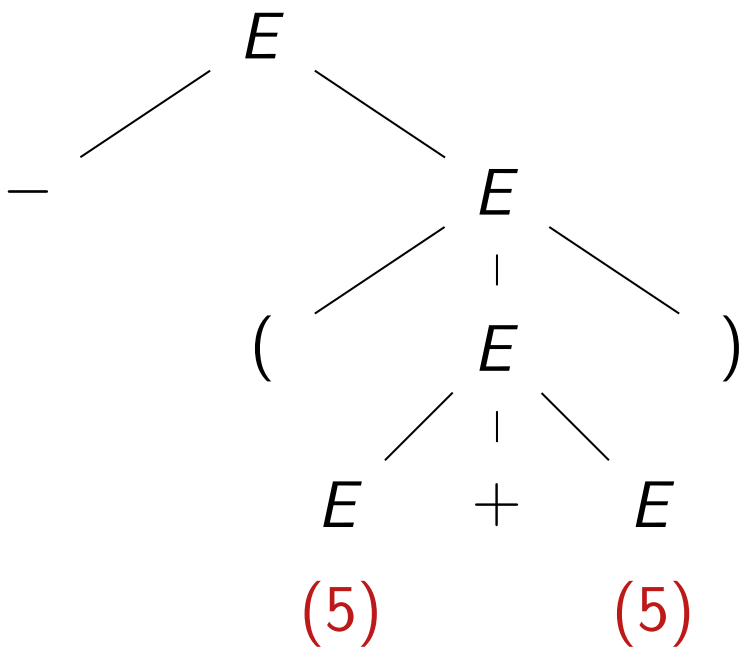
Un esempio

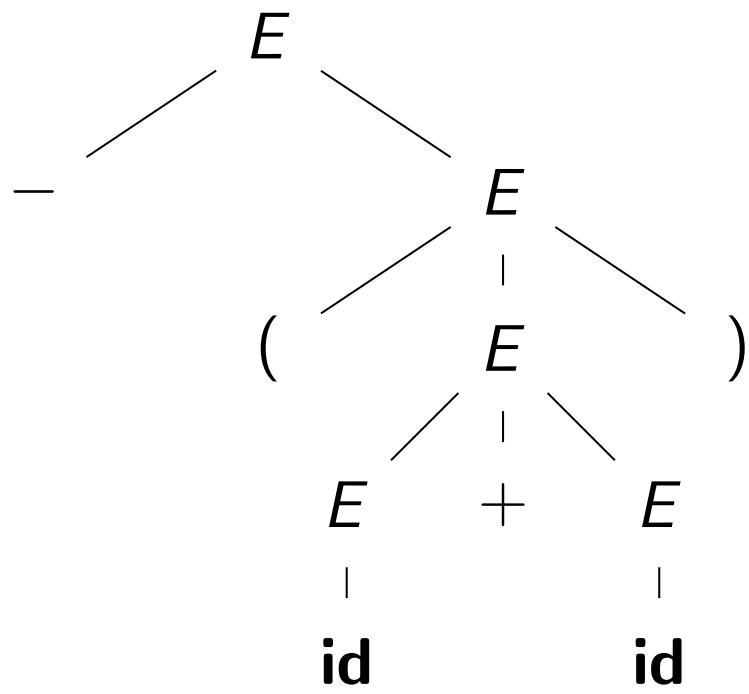
Sia G la grammatica definita da:

$$E \to E + E\ (1) \mid E * E\ (2) \mid (E)\ (3) \mid -E\ (4) \mid \mathbf{id}\ (5)$$

e consideriamo la seguente derivazione

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$





Derivazioni leftmost e rightmost

- Per effettuare un passo di derivazione $\alpha \Rightarrow \alpha'$ dobbiamo:
  - individuare un non terminale A in α, partizionare $\alpha = \beta A \gamma$
  - scegliere una produzione della forma $A \rightarrow \delta$ ed effettuare la riscrittura $\alpha = \beta A \gamma \Rightarrow \beta \delta \gamma = \alpha'$
- Possiamo fissare quale non terminale riscrivere di volta in volta
- Derivazioni leftmost: riscriviamo sempre il non terminale più a sinista
- Forma sentenziale sinistra: una qualsiasi forma sentenziale ottenuta durante una derivazione leftmost
- $\Rightarrow_{lm}$: passo di derivazione leftmost
- $\overset{*}{\Rightarrow}_{lm}$: zero o più passi di derivazione leftmost
- $\overset{+}{\Rightarrow}_{lm}$: uno o più passi di derivazione leftmost

Derivazioni leftmost e rightmost

- Derivazioni rightmost: riscriviamo sempre il non terminale più a destra
- Forma sentenziale destra: una qualsiasi forma sentenziale ottenuta durante una derivazione rightmost
- $\Rightarrow_{rm}$: passo di derivazione leftmost
- $\overset{*}{\Rightarrow}_{rm}$: zero o più passi di derivazione rightmost
- $\overset{+}{\Rightarrow}_{rm}$: uno o più passi di derivazione rightmost

Un esempio

- Sia G la grammatica definita da:

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

- Una possibile derivazione leftmost della stringa $w = \mathbf{id} + \mathbf{id} * \mathbf{id}$:

$$E \Rightarrow_{lm} E+E \Rightarrow_{lm} \mathbf{id}+E \Rightarrow_{lm} \mathbf{id}+E*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*\mathbf{id}$$

- Una possibile derivazione rightmost della stringa $w = \mathbf{id} + \mathbf{id} * \mathbf{id}$:

$$E \Rightarrow_{rm} E+E \Rightarrow_{rm} E+E*E \Rightarrow_{rm} E+E*\mathbf{id} \Rightarrow_{rm} E+\mathbf{id}*\mathbf{id} \Rightarrow_{rm} \mathbf{id}+\mathbf{id}*\mathbf{id}$$

Grammatiche Ambigue

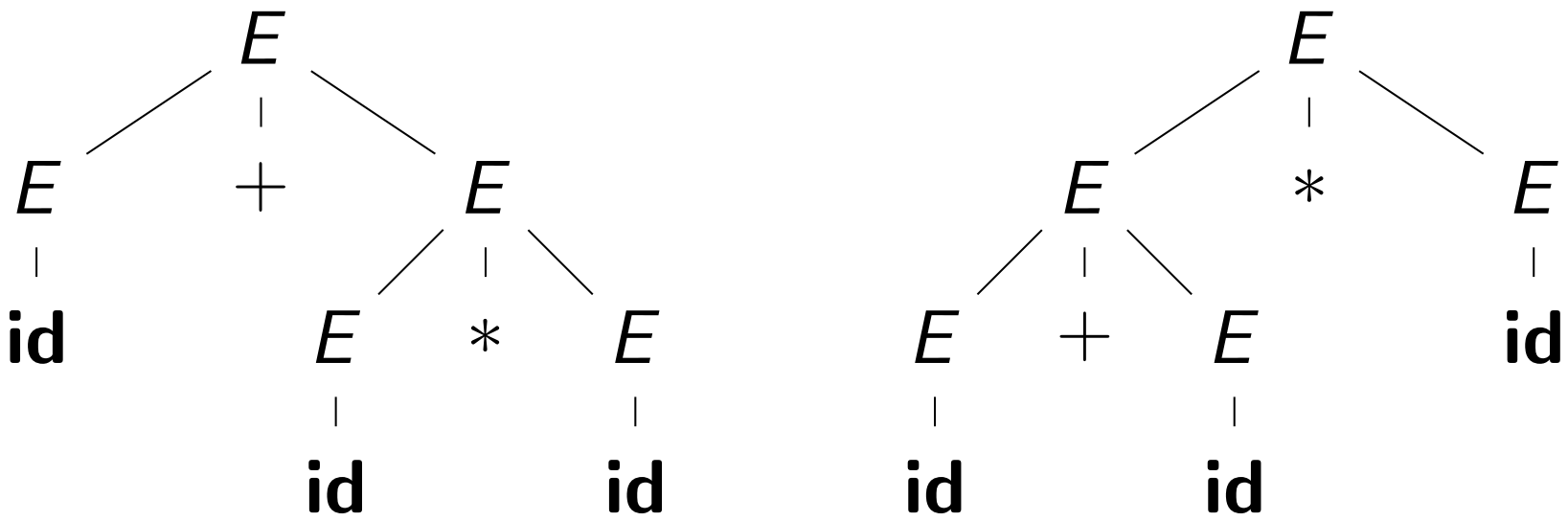
- Grammatica è la definizione di un algoritmo (ricorsivo) per generare le stringhe di un linguaggio; questo algoritmo usa per produzioni per costruire l'albero di derivazione per una certa stringa
- Una grammatica si dice **ambigua** se le sue produzioni permettono di seguire due strade differenti per generare una data stringa
- Esistono due alberi di derivazioni distinti per la stessa stringa o, in maniera alternativa, due derivazioni leftmost o rightmost distinte
- Sia G la grammatica: $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$
- G ammette due diverse derivazioni leftmost per la stringa $\mathbf{id} + \mathbf{id} * \mathbf{id}$

$$E \Rightarrow_{lm} E+E \Rightarrow_{lm} \mathbf{id}+E \Rightarrow_{lm} \mathbf{id}+E*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*\mathbf{id}$$

$$E \Rightarrow_{lm} E*E \Rightarrow_{lm} E+E*E \Rightarrow_{lm} \mathbf{id}+E*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*E \Rightarrow_{lm} \mathbf{id}+\mathbf{id}*\mathbf{id}$$

Grammatiche Ambigue

- I due alberi di derivazioni corrispondenti alle due derivazioni leftmost dell'esempio precedente



Eliminare l'ambiguità

- Per dimostrare che un grammatica è ambigua basta individuare una stringa w per quale esistono due distinti alberi di derivazione la cui stringa associata è w
- Per dimostrare che una grammatica non è ambigua bisogna provare l'unicità dell'albero di derivazione per ogni stringa generata dalla grammatica
- Indicatori di ambiguità:
  - doppia ricorsione: la testa della produzione compare almeno due volte nel corpo della produzione, es $E \longrightarrow E + E$
  - soprattutto in presenza di operatori binari, come nel caso del $+$
- L'ambiguità deve essere eliminata
  - molti algoritmi di parsing falliscono se la grammatica è ambigua
  - riscrittura delle produzioni che lascia inalterato il linguaggio generato

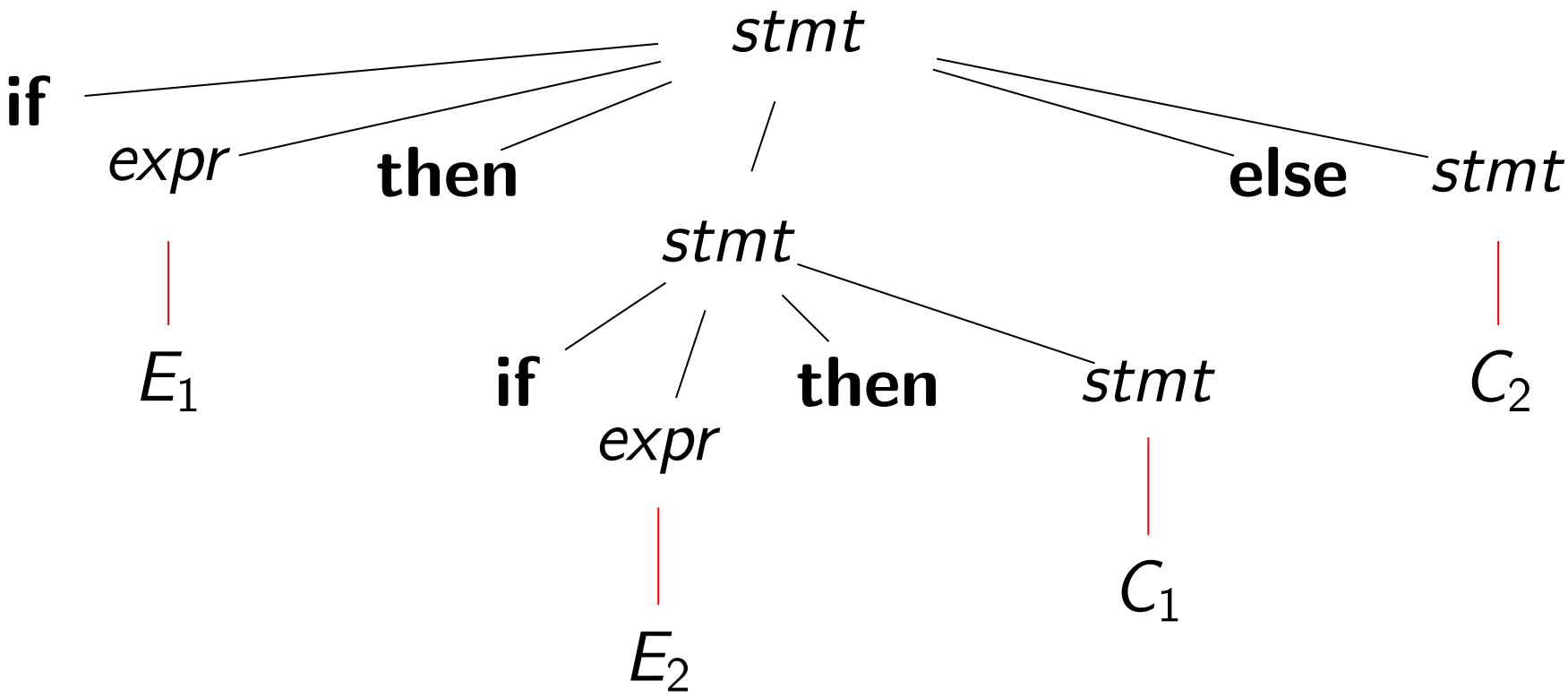
Un costrutto ambiguo: dangling else

$$
\begin{array}{lcl}
\textit{stmt} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\
& | & \textbf{other}
\end{array}
$$

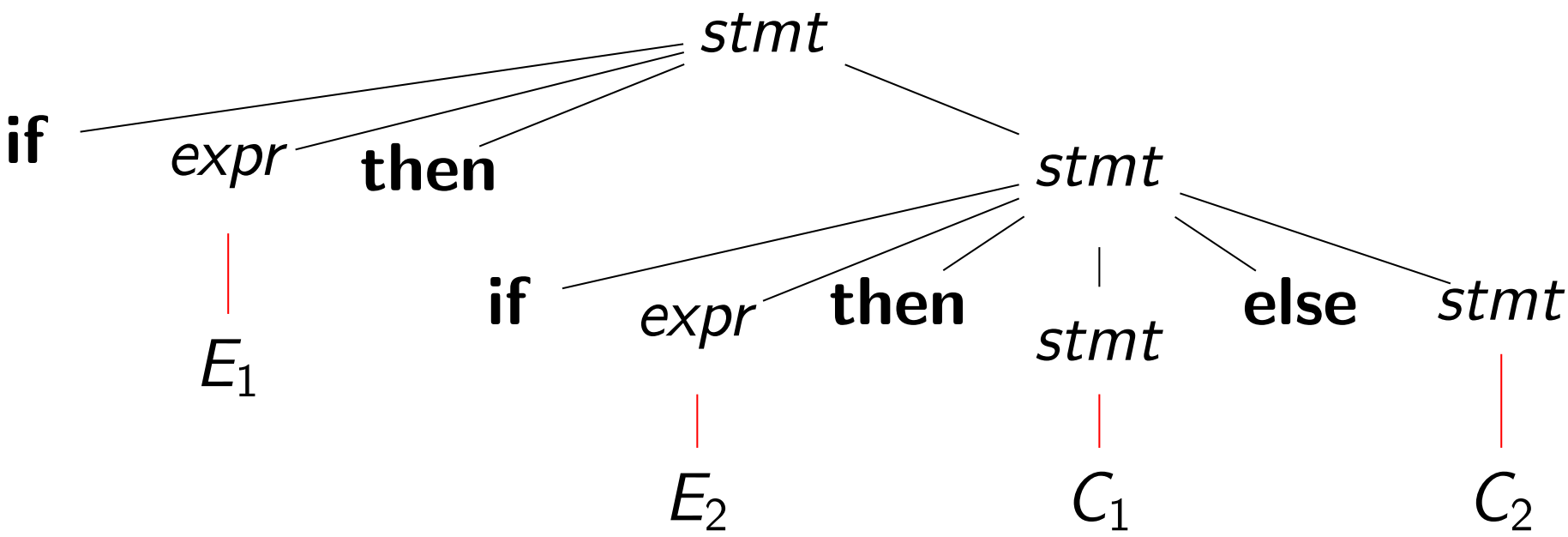
Possiamo scrivere un comando del tipo **if** E_1 **then if** E_2 **then** C_1 **else** C_2, ma, a quale then associare l'else???

- primo then: **if** E_1 **then** (**if** E_2 **then** C_1) **else** C_2
- secondo then **if** E_1 **then** (**if** E_2 **then** C_1 **else** C_2)

Caso (1)



Caso (2): quello buono

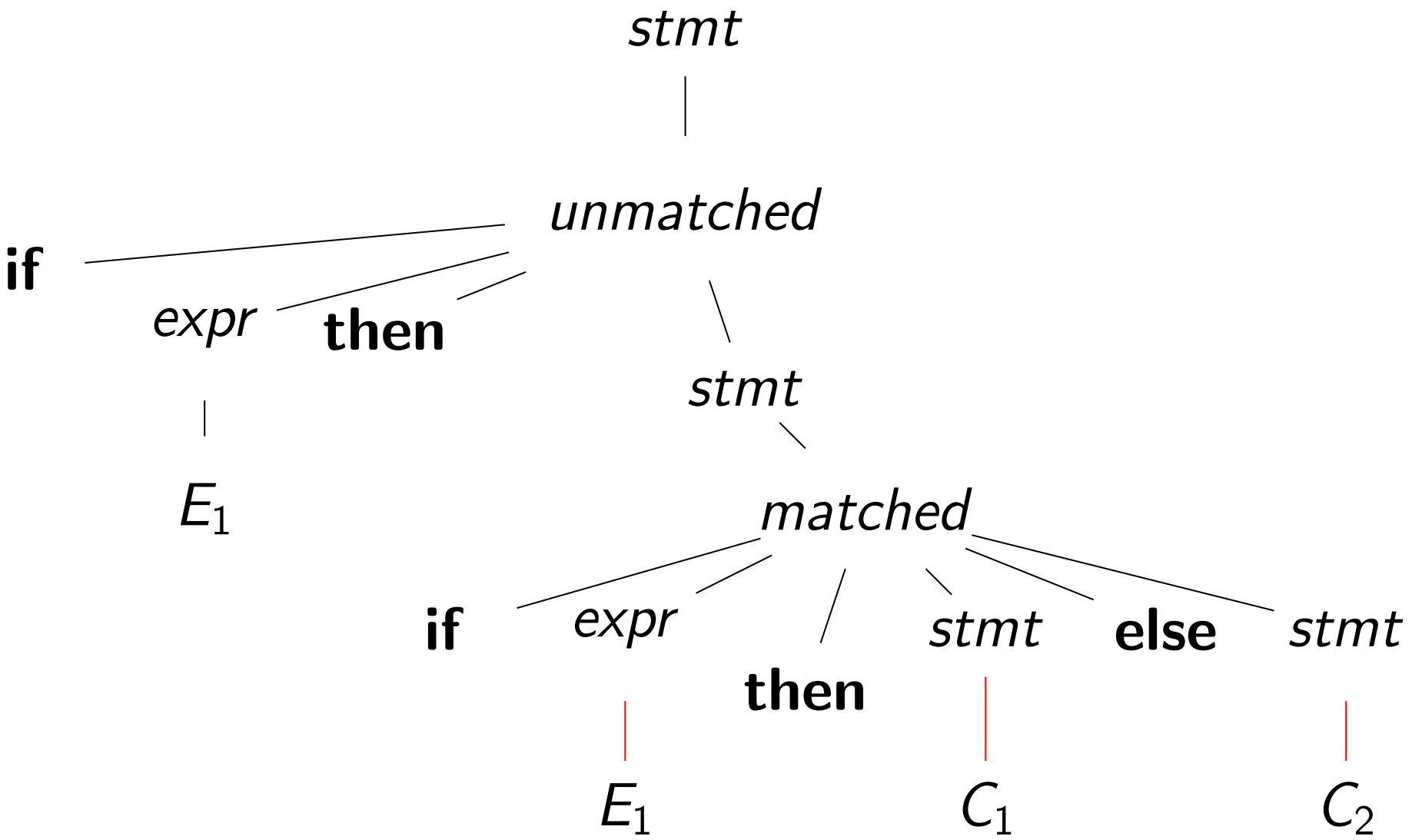


La nuova grammatica

Modifichiamo la grammatica dei comandi come segue:

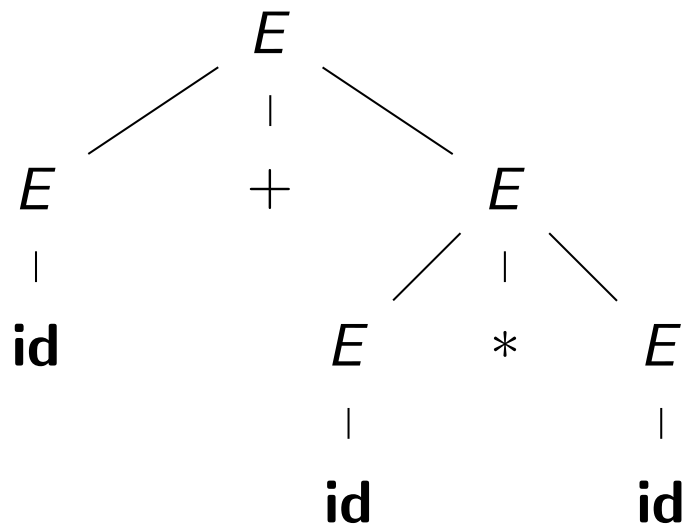
$$
\begin{array}{lcl}
\textit{stmt} & \longrightarrow & \textit{matched} \mid \textit{unmatched} \\
\textit{matched} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{matched} \textbf{ else } \textit{matched} \\
 & \mid & \textbf{other} \\
\textit{unmatched} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
 & \mid & \textbf{if } \textit{expr} \textbf{ then } \textit{matched} \textbf{ else } \textit{unmatched}
\end{array}
$$

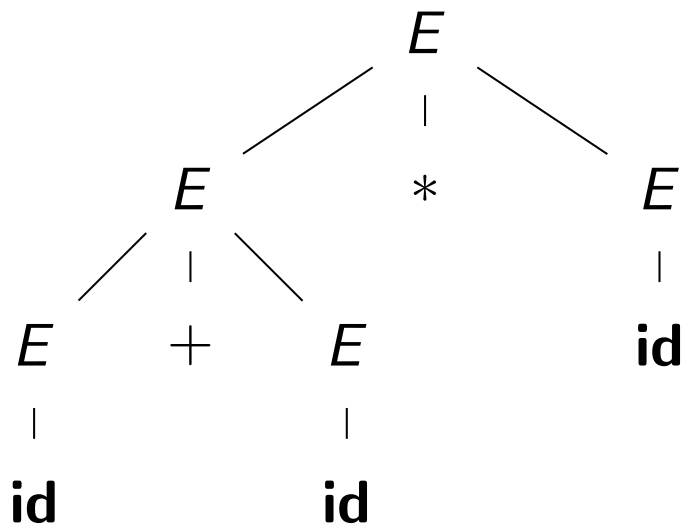
Un solo albero di derivazione



Associatività e precedenze degli operatori

Consideriamo la grammatica $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$ ed i due seguenti alberi di derivazione per la stringa $w = \mathbf{id} + \mathbf{id} * \mathbf{id}$





Associatività e precedenze degli operatori

- Gli operatori aritmetici hanno delle precedenze: $3 + 5 * 7 = 3 + (5 * 7)$ e non $(3 + 5) * 7$
- In base alle regole semantiche il secondo albero è sbagliato
- Associatività degli operatori: si consideri l'espressione $3 + 5 + 7$
  - a quale $+$ associamo il 5?
  - se il $+$ associa a sinistra, allora il 5 viene associato con il $+$ alla sua sinistra, e quindi $3 + 5 + 7 = (3 + 5) + 7$
  - se il $+$ associa a destra: il 5 viene associato con il $+$ alla sua destra, e quindi $3 + 5 + 7 = 3 + (5 + 7)$
- Nel caso dell'espressione 2 - 1 + 3
  - associatività a sinistra: $2 - 1 + 3 = (2 - 1) + 3 = 4$
  - associatività a destra: $2 - 1 + 3 = 2 - (1 + 3) = -2$

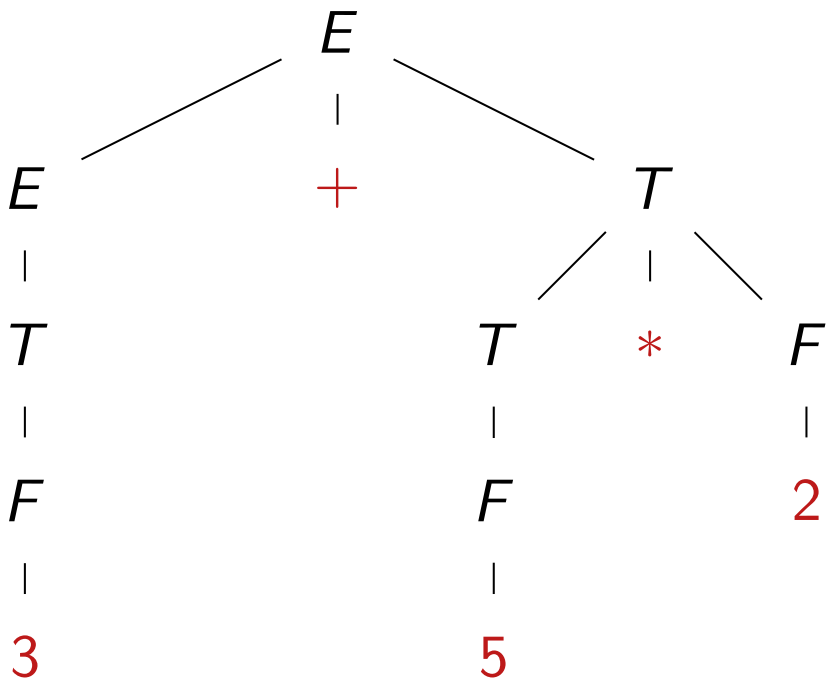
Associatività e precedenze degli operatori

- Come facciamo ad assegnare precedenze ed associatività fra operatori binari attraverso le produzioni della grammatica
  - definire vari livelli di precedenza degli operatori e per ogni livello si crea un simbolo non terminale
  - per ogni livello si indica se l'associatività e a destra o a sinistra
- Per la grammatica che genera il linguaggio delle espressioni fra naturali abbiamo quattro operatori ($+$, $-$, $*$ e $/$) e tre livelli di precedenza:
  1. fattori (F): gli operandi, livello di precedenza di precedenza più alto
  2. termini (T): operatori $*$ and $/$, livello di precedenza inferiore a quello dei fattori, associatività a sinistra
  3. espressioni (E): operatori $+$ and $-$, livello di precedenza inferiore a quello dei termine e dei fattori, associatività a sinistra

Una grammatica non ambigua per i numeri naturali

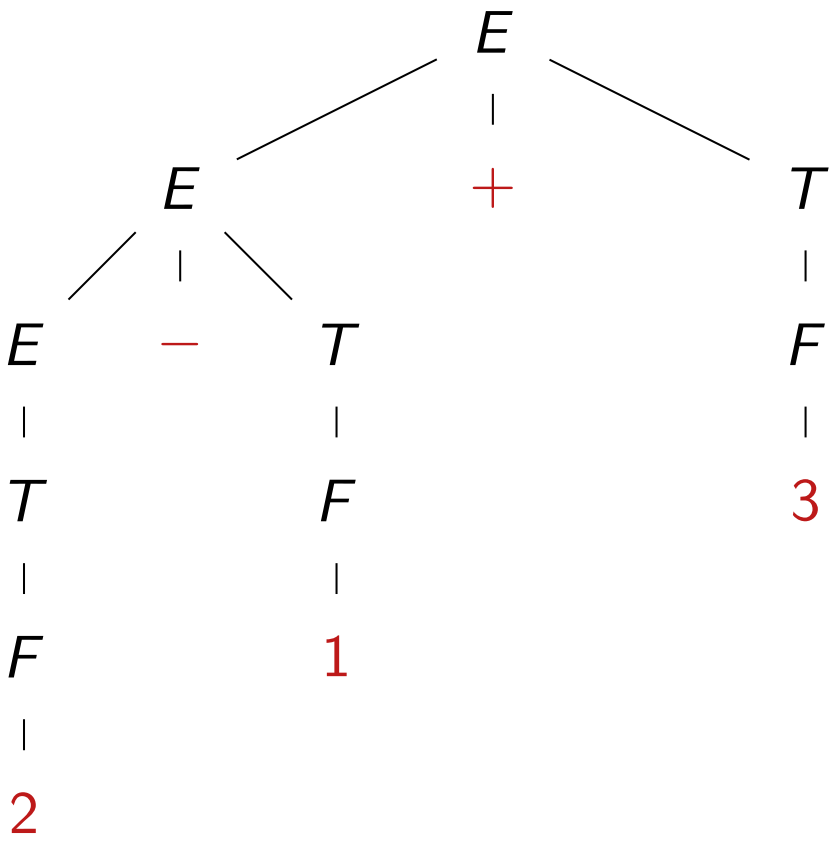
$$
\begin{aligned}
F &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \mid (E) \\
T &\rightarrow T * F \mid T/F \mid F \\
E &\rightarrow E + T \mid E - T \mid T
\end{aligned}
$$

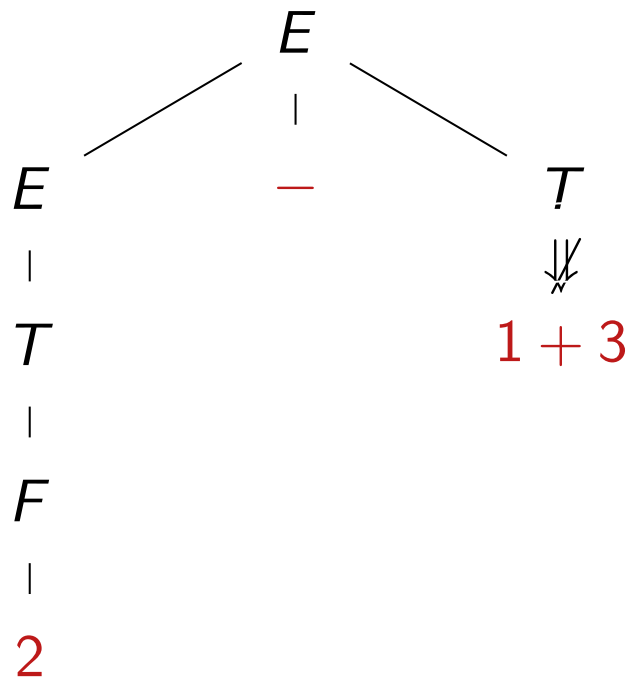
$w = 3 + 5 * 2$



Una grammatica non ambigua per i numeri naturali

$$
\begin{aligned}
F &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \mid (E) \\
T &\rightarrow T * F \mid T/F \mid F \\
E &\rightarrow E + T \mid E - T \mid T
\end{aligned}
$$





Ricorsione a sinistra

- Una grammatica si dice **ricorsiva a sinistra** se esiste un non terminale A tale che $A \stackrel{+}{\Rightarrow} A\alpha$ per qualche stringa di simboli α
- A si riscrive in uno o più passi nella stringa $A\alpha$
- **Ricorsione immediata**: esistono produzioni della forma

$$A \to A\alpha \mid \beta$$

- Dal non terminale A riusciamo a derivare stringhe della forma

$$A \Rightarrow A\alpha \Rightarrow A\alpha\alpha \Rightarrow A\alpha\alpha\alpha \ldots \Rightarrow A\alpha \ldots \alpha\alpha\alpha \Rightarrow \beta\alpha \ldots \alpha\alpha\alpha$$

- Il primo passo genera l'ultima α, il secondo la penultima e così via; l'ultimo genera la β

Eliminare la ricorsione a sinistra

- Riscrivere le produzioni per il non terminale A lasciando inalterato il linguaggio generato

$$A \to A\,\alpha \mid \beta \qquad A \to \beta\,A' \quad A' \to \alpha\,A' \mid \varepsilon$$

$$E \to E + T \mid T \qquad \alpha = +T, \beta = T \qquad E \to TE' \quad E' \to +TE' \mid \varepsilon$$

$$T \to T * F \mid F \qquad \alpha = *F, \beta = F \qquad T \to FT' \quad T' \to *FT' \mid \varepsilon$$

$$F \to (E) \mid \mathbf{id} \qquad F \to (E) \mid \mathbf{id}$$

Eliminare la ricorsione a sinistra

- In generale, possiamo avere un insieme di produzioni della forma:

$$A \to A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

  dove nessun β_j inizia per A ed ogni $\alpha_i \neq \varepsilon$
- Queste produzioni vengono sostituite con

$$A \to \beta_1\,A' \mid \beta_2\,A' \mid \ldots \mid \beta_n\,A'$$
$$A' \to \alpha_1\,A' \mid \alpha_2\,A' \mid \ldots \mid \alpha_m\,A' \mid \varepsilon$$

- Questo elimina la ricorsione sinistra immediata

Ricorsione a sinistra non immediata

- Consideriamo la seguente grammatica

$$S \longrightarrow Aa \mid b$$
$$A \longrightarrow Ac \mid Sd \mid c$$

- Ricorsione immediata: $A \longrightarrow Ac \mid c$
- Ricorsione non immediata: $S \Rightarrow Aa \Rightarrow Sda$
- Algoritmo per eliminare sistematicamente la ricorsione a sinistra (immediata o meno)
- È garantito funzionare per grammatiche che non hanno:
  - cicli: situazioni del tipo $A \overset{+}{\Rightarrow} A$
  - ε-produzioni: produzioni del tipo $A \longrightarrow \varepsilon$

Algoritmo per eliminare la ricorsione

- **Input**: una grammatica senze cicli ed ε-produzioni
- **Output**: una grammatica equivalente senza ricorsione a sinistra

1. Ordina i non terminali: $A_1, A_2, \ldots, A_n$
2. **for** $k := 1$ **to** n **do begin**
   for $j := 1$ **to** $k - 1$ **do begin**
   sostituisci ogni produzione della forma $A_k \to A_j\ \gamma$ con le produzioni $A_k \to \delta_1\ \gamma \mid \delta_2\ \gamma \mid \ldots \mid \delta_m\ \gamma$ dove $A_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_m$ sono le attuali produzioni per il non terminale e A_j
   end
   elimina la ricorsione immediata per il non terminale A_k

end

Un esempio

- Consideriamo la seguente grammatica

$$S \to Aa \mid b \qquad A_1 \to A_2 a \mid b$$
$$A \to Ac \mid Sd \mid c \qquad A_2 \to A_2 c \mid A_1 d \mid c$$

- primo passo $k = 1$
  - **for** $j := 1$ **to** $1 - 1 = 0$ il for più interno non viene eseguito
  - A_1 non ha ricorsione immediata
- secondo passo $k = 2$
  - **for** $j := 1$ **to** $2 - 1 = 1$: cerchiamo produzioni della forma $A_2 \to A_1\, \gamma$
  - abbiamo una sola produzione di quella forma $A_2 \to A_1 d$ $(\gamma = d)$
  - le produzioni correnti per A_1 sono: $A_1 \to A_2 a \mid b$ $(\delta_1 = A_2 a, \delta_2 b)$
  - $A_2 \to A_1 d$ viene sostituita con le produzioni

$$A_2 \to A_2 ad(\delta_1\ \gamma) \mid bd(\delta_2\ \gamma)$$

  - Dobbiamo eliminare la ricorsione sinistra per A_2

Un esempio

- secondo passo $k = 2$
  - ...
  - eliminare la ricorsione per A_2: $A_2 \to A_2 ad \mid bd$($\alpha = ad, \beta = bd$):

$$\begin{aligned} A_2 &\to bd\,A_2' \\ A_2' &\to ad\,A_2' \mid \varepsilon \end{aligned}$$

- Otteniamo così la seguente grammatica senza ricorsione

$$A_1 \to A_2 a \mid b$$

$$\begin{aligned} A_2 &\to bd\,A_2' \\ A_2' &\to ad\,A_2' \mid \varepsilon \end{aligned}$$

Fattorizzazione a sinistra

- Supponiamo di avere la seguente grammatica:

$$S \longrightarrow \textbf{if } E \textbf{ then } S \mid \textbf{if } E \textbf{ then } S \textbf{ else } S \mid \textbf{a}$$
$$E \longrightarrow \textbf{b}$$

  e di aver riconosciuto sulla stringa in input il token **if**; quale delle due produzioni usare per espandere S?
- Vorrei poter effettuare questa scelta in maniera predittiva (cioè, senza backtracking)
- La soluzione di questo problema consiste nel "fattorizzare" in base al prefisso comune delle due alternative, cioè **if** E **then** S

$$S \longrightarrow \textbf{if } E \textbf{ then } S\ S' \mid \textbf{a}$$
$$S' \longrightarrow \textbf{else } S \mid \varepsilon$$
$$E \longrightarrow \textbf{b}$$

- In questo modo rimandiamo la scelta a quando avremo esaminato abbastanza input da decidere

Fattorizzazione a sinistra: algoritmo

- **Input**: una grammatica G
- **Output**: una grammatica equivalente fattorizzata a sinistra
- Per ogni non terminale A:
  - trova il più lungo prefisso α per le sue alternative
  - se $\alpha \neq \varepsilon$ rimpiazza

$$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k$$

con

$$A \to \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k$$
$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$