Parsing Top-Down

Maria Rita Di Berardini

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

Parser Top-down

Costruiscono l'albero di derivazione dalla radice alle foglie

- è un tentativo di costruire una **derivazione leftmost** della stringa in input
- ad ogni passo espandono il non terminale più a sinistra che non ha figli

Consideriamo la seguente grammatica

$$\begin{array}{lclcccc} \mathit{type} & \rightarrow & \mathit{simple} & \mid & \uparrow \mathbf{id} & \mid & \mathbf{array}\,[\mathit{simple}]\,\mathbf{of}\ \mathit{type} \\ \mathit{simple} & \rightarrow & \mathbf{integer} & \mid & \mathbf{char} & \mid & \mathbf{num\ dotdot\ num} \end{array}$$

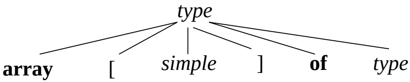
ed esaminiamo come un parser top-down esamina la stringa

$$\mathbf{array}\,[\mathbf{num\ dotdot\ num}]\,\mathbf{of\ char}$$

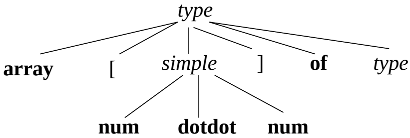
1)


type

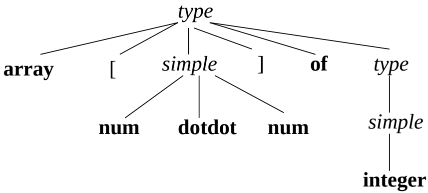
2)


type
array
[
simple
]
of
type

3)


type
array
[
simple
]
of
type
num
dotdot
num

4)


type
array
[
simple
]
of
type
num
dotdot
num
simple
integer

Tipologie di parser top-down

- Il problema principale, è quello di operare una scelta corretta della produzione con cui espandere il nodo selezionato
- Se un albero di derivazione esiste e la grammatica non è ambigua, allora ad ogni passo esiste una ed una sola scelta corretta (una sola derivazione leftmost possibile). Come identifichiamo la scelta corretta?
- **Parser con backtracking**
  - possono dover ritornare sulle proprie scelte se si accorgono di non poter derivare la stringa in input
  - piuttosto inefficienti: nel caso pessimo deve operare tutte le scelta per tutti i possibili non terminali
- **Parser predittivi**
  - sono in grado di "indovinare" ad ogni passo la produzione che porterà alla derivazione della stringa
  - analizzano il minimo numero di terminali (tipicamente 1) necessari per prendere la scelta giusta (**simboli di lookahead**)

Parser predittivi

- Per ogni non terminale A con alternative $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ e per ogni simbolo di lookahead a sono in grado di identificare la sola alternativa di A in grado di generare stringhe che cominciano per a
- In alcuni casi è facile:

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & \mid & \textbf{while } expr \textbf{ do } stmt \\
 & \mid & \textbf{begin } stmt\_list \textbf{ end}
\end{array}
$$

- Parser predittivi ricorsivi
- **Parser predittivi non ricorsivi** (iterativi): utilizzano uno stack

Alcune modifiche alle grammatiche

- Introduciamo delle trasformazioni che permettono di modificare una grammatica in modo da renderla adatta ad un parser top-down predittivo.
- Prima di costruire un parser top-down per una grammatica G bisogna:
  - **elimininare la ricorsione sinistra**:
  - **fattorizzare a sinistra**

Ricorsione a sinistra

- Una CFG si dice **ricorsiva a sinistra** se esiste un non terminale A t.c.

$$A \stackrel{+}{\Rightarrow} A\alpha$$

- **Ricorsione immediata**: esistono delle produzioni della forma $A \longrightarrow A\alpha \mid \beta$
- Per eliminare la ricorsione immediata basta riscrivere le produzioni per A lasciando inalterato il linguaggio generato

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

- Le produzioni $A \longrightarrow A\alpha \mid \beta$ vengono sostituite con

$$\begin{array}{l} A \longrightarrow \beta\ A' \\ A' \longrightarrow \alpha\ A' \mid \epsilon \end{array}$$

Eliminare la ricorsione a sinistra

$$
\begin{array}{lll}
E \rightarrow E + T \mid T & \alpha = +T, \beta = T & E \rightarrow TE' \\
 & & E' \rightarrow +TE' \mid \epsilon \\
T \rightarrow T * F \mid F & \alpha = *F, \beta = F & T \rightarrow FT' \\
 & & T' \rightarrow *FT' \mid \epsilon \\
F \rightarrow (E) \mid \mathbf{id} & & F \rightarrow (E) \mid \mathbf{id}
\end{array}
$$

Eliminare la ricorsione a sinistra

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

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

  dove nessun β_j inizia per A ed ogni $\alpha_i \neq \varepsilon$

- Queste produzioni vengono sostituite con

$$\begin{aligned} A &\longrightarrow \beta_1\,A' \mid \beta_2\,A' \mid \ldots \mid \beta_n\,A' \\ A' &\longrightarrow \alpha_1\,A' \mid \alpha_2\,A' \mid \ldots \mid \alpha_m\,A' \mid \varepsilon \end{aligned}$$

- Questo elimina la ricorsione sinistra immediata

Ricorsione a sinistra non immediata

- La seguente grammatica ha due forme di ricorsione a sinistra

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Sd \mid B \\ B &\rightarrow eB \mid a \end{aligned}$$

- Ricorsione immediata ($A \rightarrow Ac$) e non immediata ($S \Rightarrow Aa \Rightarrow Sda$)
- Se sostiamo la produzione $A \rightarrow Sd$ con le produzioni

$$A \rightarrow Aad \mid bd$$

(abbiamo rimpiazzato l'occorrenza di S nella parte destra di $A \rightarrow Sd$ con il corpo di tutte le produzioni per S) otteniamo una grammatica:

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow Ac \mid Aad \mid bd \mid B \\ B &\rightarrow eB \mid a \end{aligned}$$

con solo ricorsioni immediate

Ricorsione a sinistra non immediata

- Algoritmo per eliminare sistematicamente la ricorsione a sinistra (immediata o meno)
- È corretto solo a grammatiche che non hanno:
  - cicli: situazioni del tipo $A \stackrel{+}{\Rightarrow} A$
  - ϵ-produzioni: produzioni del tipo $A \longrightarrow \epsilon$

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 $A_k \longrightarrow A_j\ \gamma$ con le produzioni
   $$A_k \longrightarrow \delta_1\ \gamma \mid \delta_2\ \gamma \mid \cdots \mid \delta_m\ \gamma \quad \text{dove}$$
   $$A_j \longrightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_m$$
   sono le attuali produzioni per A_j
   end
   elimina la ricorsione immediata per il non terminale A_k
   end

Fattorizzazione a sinistra

- Supponiamo di avere la seguente grammatica:

$$\begin{aligned} S &\longrightarrow \textbf{if } E \textbf{ then } S \mid \textbf{if } E \textbf{ then } S \textbf{ else } S \;\mid \textbf{a} \\ E &\longrightarrow \textbf{b} \end{aligned}$$

  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
- La soluzione di questo problema consiste nel "fattorizzare" in base al prefisso comune delle due alternative, cioè **if** E **then** S

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

- Rimandiamo la scelta a quando avremo esaminato abbastanza input da decidere

Fattorizzazione a sinistra: algoritmo

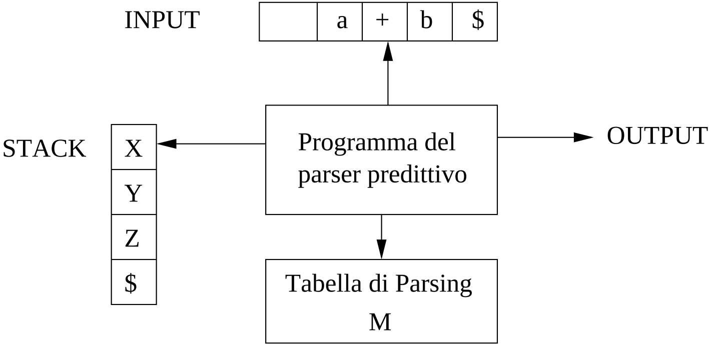
- **Input**: una grammatica G
- **Output**: una grammatica equivalente fattorizzata a sinistra
- Per ogni non terminale A:
  - trova il più lungo prefisso α per le sue alternative
  - se $\alpha \neq \varepsilon$ rimpiazza

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

con

$$\begin{array}{l} A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k \\ A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \end{array}$$

Parser predittivi non ricorsivi: struttura



Parser predittivi non ricorsivi: struttura

- Lo stack contiene simboli terminali e non, più il \$
- M è indicizzata da non terminali (sulle righe) e da simboli in $\Sigma \cup \{\$\}$ (sulle colonne)
- Dato $A \in V_G$ ed $a \in \Sigma \cup \{\$\}$, la entry $M[A, a]$ indica quale mossa eseguire
- In particolare, $M[A, a]$ restituisce una produzione $A \rightarrow \alpha$ oppure un **errore**.
- Il comportamento del parser dipende dal simbolo X in testa allo stack e dal corrente simbolo a in input
- L'output della programma è un albero di derivazione per la stringa in input oppure un messaggio di errore

Programma di parsing

Configurazione iniziale: STACK: $\$S$ (S è non terminale iniziale), INPUT: $w\$$ (w è la stringa da parsare)

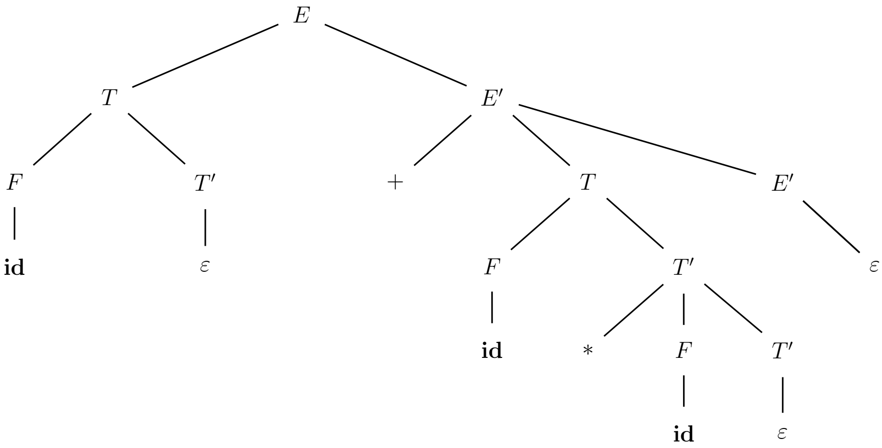
Il comportamento del parser dipende da X (simbolo al top dello stack) ed a (corrente simbolo in input):

1. Se $X = a = \$$, il parser termina con successo
2. Se $X = a \neq \$$, elimina a dallo stack e fa avanzare il simbolo di lookahead
3. Se X è un non terminale consulta l'entry $M[X, a]$ della tabella di parsing. Due possibili casi:
   - 3.1 $M[X, a] := X \rightarrow \alpha$: elimina X dallo stack ed inserisce i simboli di α in maniera tale che il simbolo più a sinistra sia il prossimo simbolo in testa allo stack
     ES: Se $X \rightarrow UVW$ esegue pop(); push(W); push(V); push(U).
     Stampa la produzione $X \rightarrow \alpha$
   - 3.2 $M[X, a] := error$: il parser chiama una procedura di recovery dell'errore

	id	$+$	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

stack	input	output
$E	**id** + **id** * **id** $	
$E'T	**id** + **id** * **id** $	$E \rightarrow TE'$
$E'T'F	**id** + **id** * **id** $	$T \rightarrow FT'$
$E'T'**id**	**id** + **id** * **id** $	$F \rightarrow$ **id**
$E'T'	+ **id** * **id** $	
$E'	+ **id** * **id** $	$T' \rightarrow \varepsilon$
$E'T +	+ **id** * **id** $	$E' \rightarrow +TE'$
$E'T	**id** * **id** $	$E' \rightarrow +TE'$
$E'T'F	**id** * **id** $	$T \rightarrow FT'$
$E'T'**id**	**id** * **id** $	$F \rightarrow$ **id**
$E'T'	* **id** $	
$E'T'F *	* **id** $	$T' \rightarrow *FT'$
$E'T'F	**id** $	
$E'T'**id**	**id** $	$F \rightarrow$ **id**
$E'T'	$	
$E'	$	$T' \rightarrow \epsilon$
$'	$	$E' \rightarrow \epsilon$

Output



Due funzioni ausiliarie: FIRST e FOLLOW

L'algoritmo per la costruzione della tabella di parsing si avvale di due funzioni ausiliarie: la FIRST e la FOLLOW

La FIRST

- È definita su stringhe $\alpha \in (V \cup \Sigma)^*$
- $FIRST(\alpha)$ restituisce l'insieme dei **terminali** con cui iniziamo stringhe derivabili da α, ossia:

$$\text{Se } \alpha \stackrel{*}{\Rightarrow} a\beta \text{ allora } a \in FIRST(\alpha)$$

- può anche contenere la stringa ϵ:

$$\alpha \stackrel{*}{\Rightarrow} \epsilon \text{ implica } \epsilon \in FIRST(\alpha)$$

La FOLLOW

- È definita su non terminali della grammatica
- $FOLLOW(A)$ restituisce l'insieme dei **terminali** che compaiono immediatamente a destra di A in qualche forma sentenziale della grammatica

$$\text{Se } S \stackrel{*}{\Rightarrow} \alpha\, A\, a\, \beta \text{ allora } a \in FOLLOW(A)$$

- può anche contenere il simbolo \$:

$$S \stackrel{*}{\Rightarrow} \alpha\, A \text{ implica } \$ \in FOLLOW(A)$$

FIRST di simboli

Sia X un generico simbolo della grammatica. Calcoliamo la $FIRST(X)$ applicando le seguenti regole finchè non è più possibile aggiungere alcun nuovo elemento

- $FIRST(X) = \{X\}$ per ogni terminale $X \in \Sigma$;
- Se X è un non terminale ed $X \rightarrow \epsilon$, aggiungi ϵ a $FIRST(X)$;
- Se X è un non terminale ed $X \rightarrow Y_1 Y_2 \ldots Y_k$ è una produzione per X:
  - se $\epsilon \in FIRST(Y_1), FIRST(Y_2), \cdots, FIRST(Y_{j-1})$ ed $a \neq \epsilon$ è un **terminale** in $FIRST(Y_j)$, aggiungi a in $FIRST(X)$
  - se, per ogni $j \in [1, k]$, $\epsilon \in FIRST(Y_j)$, aggiungi ϵ in $FIRST(X)$

FIRST di simboli

Sia X un non terminale e $X \rightarrow Y_1 Y_2 \ldots Y_k$ una produzione per X. In base alle regole 1 e 2:

1. inizialmente, aggiungiamo $FIRST(X)$ ogni terminale in$T FIRST(Y_1)$
2. se $\epsilon \notin FIRST(Y_1)$ non aggiungiamo ulteriori elementi; se invece $\epsilon \in FIRST(Y_1)$ passiamo a considerare il simbolo Y_2
3. aggiungiamo a $FIRST(X)$ anche ogni terminale in $FIRST(Y_2)$ ed iteriamo
4. se $\epsilon \notin FIRST(Y_2)$ ...
5. ϵ viene agginta a$FIRST(X)$ se, per ogni $j = 1, \ldots, k$, $\epsilon \in FIRST(Y_j)$

Un esempio

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid \mathbf{id}$$

- Dalle produzioni $F \rightarrow (E)$ ed $F \rightarrow \mathbf{id}$, $FIRST(F) = \{(, \mathbf{id}\}$
- Dalle produzioni $T' \rightarrow *FT$ e $T' \rightarrow \epsilon$, $FIRST(T') = \{*, \epsilon\}$
- $T \rightarrow FT'$ ed $\epsilon \notin FIRST(F) = \{(, \mathbf{id}\}$ implica $FIRST(T) = FIRST(F) = \{(, \mathbf{id}\}$
- Dalle produzioni $E' \rightarrow +TE$ ed $E' \rightarrow \epsilon$, $FIRST(E') = \{+, \epsilon\}$
- $E \rightarrow TE'$ ed $\epsilon \notin FIRST(T) = \{(, \mathbf{id}\}$ implica $FIRST(E) = FIRST(T) = \{(, \mathbf{id}\}$

FIRST di stringhe

Se $\alpha = X_1X_2 \dots X_n$ è una qualsiasi stringa di simboli grammaticali, la $FIRST(\alpha)$ viene calcolata applicando le seguenti regole:

- Aggiungi a $FIRST(\alpha)$ tutti i simboli in $FIRST(X_1)$ tranne ϵ
- Se $\epsilon \in FIRST(X_1)$, aggiungi a $FIRST(\alpha)$ tutti i simboli in $FIRST(X_2)$ tranne ϵ
- Se $\epsilon \in FIRST(X_2)$, aggiungi a $FIRST(\alpha)$ tutti i simboli in $FIRST(X_3)$ tranne ϵ
- $\dots$
- Se, per ogni $i = 1, \cdots, n$, $\epsilon \in FIRST(X_i)$, aggiungi ϵ a $FIRST(\alpha)$

FOLLOW

La $FOLLOW(B)$ viene costruita, applicando le seguenti regole, a partire dalle produzioni che contengono il non terminale B nella parte destra:

1. aggiungi il \$ in $FOLLOW(S)$, non terminale iniziale della grammatica
2. per ogni produzione della forma $A \longrightarrow \alpha B \beta$, aggiungi in $FOLLOW(B)$ ogni terminale in $FIRST(\beta)$
3. per ogni produzione della forma $A \longrightarrow \alpha B$ o della forma $A \longrightarrow \alpha B \beta$ con $\beta \stackrel{*}{\Rightarrow} \epsilon$, aggiungi in FOLLOW(B) ogni simbolo in $FOLLOW(A)$

La seconda regola è abbastanza intuitiva; infatti, se $A \rightarrow \alpha B \beta$ è una produzione della grammatica, allora tutti i terminali con cui iniziamo stringhe derivabili da β appartengono a $FOLLOW(B)$

Perchè la terza?

FOLLOW

- Se $a \in FOLLOW(A)$ allora a compare a destra di una in una qualche forma sentenziale, ossia:

$$S \stackrel{*}{\Rightarrow} \alpha_1 \, A \, a \, \beta_1$$

- Se $A \rightarrow \alpha \, B$ una produzione, allora

$$S \stackrel{*}{\Rightarrow} \alpha_1 \, A \, a \, \beta_1 \Rightarrow \alpha_1 \, \alpha \, B \, a \, \beta_1$$

  e, quindi, a appartiene anche a $FOLLOW(B)$

- Se $A \rightarrow \alpha \, B \, \beta$ con $\beta \stackrel{*}{\Rightarrow} \epsilon$, allora

$$S \stackrel{*}{\Rightarrow} \alpha_1 \, A \, a \, \beta_1 \Rightarrow \alpha_1 \, \alpha \, B \, \beta \, a \, \beta_1 \stackrel{*}{\Rightarrow} \alpha_1 \, \alpha \, B \, a \, \beta_1$$

  e, di nuovo, a appartiene anche a $FOLLOW(B)$

Un esempio

$$E \longrightarrow TE'$$
$$E' \longrightarrow +TE' \mid \epsilon$$

$$T \longrightarrow FT'$$
$$T' \longrightarrow *FT' \mid \epsilon$$

$$F \longrightarrow (E) \mid \textbf{id}$$

- $FOLLOW(E)$

  $F \rightarrow (E)$

  Per la regola 2 aggiungiamo) a $FOLLOW(E)$. Inoltre, E è il simbolo iniziale e quindi inseriamo anche il \$. Allora: $FOLLOW(E) = \{), \$\}$

- $FOLLOW(E')$

  $E \rightarrow TE'$ ed $E' \rightarrow +TE'$.

  Dalla regola 3, i simboli in $FOLLOW(E)$ ed in $FOLLOW(E')$ vanno aggiunti in $FOLLOW(E')$. Quindi:
  $FOLLOW(E') = FOLLOW(E) = \{), \$\}$

Un esempio

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid \textbf{id}$$

- $FOLLOW(T)$
  $E \rightarrow TE'$ ed $E' \rightarrow +TE'$ con $E' \Rightarrow \epsilon$
  Per la regola 2, aggiungiamo in $FOLLOW(T)$ tutti i terminali in $FIRST(E')$ (cioè $+$).
  Per la regola 3 aggiungiamo in $FOLLOW(T)$ tutti i simboli in $FOLLOW(E) = FOLLOW(E') = \{), \$\}$
  Quindi: $FOLLOW(T) = \{+,), \$\}$

Un esempio

$$E \longrightarrow TE'$$
$$E' \longrightarrow +TE' \mid \epsilon$$

$$T \longrightarrow FT'$$
$$T' \longrightarrow *FT' \mid \epsilon$$

$$F \longrightarrow (E) \mid \textbf{id}$$

- $FOLLOW(T')$
  $T \rightarrow FT'$ e $T' \rightarrow *FT'$
  $FOLLOW(T') = FOLLOW(T) = \{+,), \$\}$
- $FOLLOW(F)$
  $T \rightarrow FT'$ con $T' \Rightarrow \epsilon$.
  Per la 2, ogni terminale in $FIRT(T') = \{*, \epsilon\}$ viene aggiunto in $FOLLOW(F)$
  Per la 3, aggiungiamo anche i simboli in $\text{FOLLOW}(T) = \{+,), \$\}$
  Quindi $\text{FOLLOW}(F) = \{*, +,), \$\}$

Costruzione della tabella

Input: una grammatica G

Output: una parsing table M per la grammatica G

Metodo:

1. Per ogni produzione $A \rightarrow \alpha$ applica i passi 2 e 3
2. Per ogni terminale $a \in FIRST(\alpha)$, aggiungi $A \rightarrow \alpha$ in $M(A, a)$
3. Se $\epsilon \in FIRST(\alpha)$ aggiungi $A \rightarrow \alpha$ in $M(A, b)$ per ogni simbolo (anche il \$) $b \in FOLLOW(A)$
4. Poni ogni entrata indefinita ad error

Un esempio

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' \mid \epsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' \mid \epsilon \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$

	$FIRST$	$FOLLOW$
E	$\{(, \mathbf{id}\}$	$\{), \$\}$
E'	$\{+, \epsilon\}$	$\{), \$\}$
T	$\{(, \mathbf{id}\}$	$\{+,), \$\}$
T'	$\{*, \epsilon\}$	$\{+,), \$\}$
F	$\{(, \mathbf{id}\}$	$\{*, +,), \$\}$

- $E \rightarrow TE'$ e $FIRST(TE') = FIRST(T) = \{(, \mathbf{id}\}$ implica $M(E, () := M(E, \mathbf{id}) := E \rightarrow TE'$
- $E' \rightarrow +TE'$: $\mathrm{FIRST}(+TE') = \mathrm{FIRST}(+) = \{+\}$ implica $M(E, +) := E' \rightarrow +TE'$
- $E' \rightarrow \varepsilon$: poichè $\mathrm{FOLLOW}(E') = \{), \$\}$, $M(E',)) := M(E', \$) := E' \rightarrow \varepsilon$

Ricapitolando

	id	$+$	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \textbf{id}$			$F \rightarrow (E)$		

Grammatiche LL(1)

- Il procedimento appena descritto per la costruzione della tabella di parsing può essere applicato ad una qualsiasi grammatica contex-free
- Tuttavia, per alcune grammatiche, M può avere delle entrate indefinite (più di un valore nella stessa casella $M(A, a)$ della tabella)
- Se G è ambigua o ricorsiva sinistra allora M avrà almeno un entrata indefinita
- Consideriamo la seguente grammatica che astrae il costrutto if-then-else

$$\begin{array}{lcl} S & \rightarrow & \mathbf{i}\; E\; \mathbf{t}\; S\; S' \mid \mathbf{a} \\ S' & \rightarrow & \mathbf{e}\; S \mid \varepsilon \\ E & \rightarrow & \mathbf{b} \end{array}$$

e le produzioni $S' \rightarrow \mathbf{e}\; S$ e $S' \rightarrow \varepsilon$

Grammatiche LL(1)

- Consideriamo la seguente grammatica che astrae il costrutto if-then-else

$$\begin{array}{lcl} S & \rightarrow & \mathbf{i}\ E\ \mathbf{t}\ S\ S' \mid \mathbf{a} \\ S' & \rightarrow & \mathbf{e}\ S \mid \varepsilon \\ E & \rightarrow & \mathbf{b} \end{array}$$

  e le produzioni $S' \rightarrow \mathbf{e}\ S$ e $S' \rightarrow \varepsilon$
- $\mathrm{FIRST}(S) = \{\mathbf{i}, \mathbf{a}\}$, $\mathrm{FIRST}(S') = \{\mathbf{e}, \varepsilon\}$, $\mathrm{FIRST}(E) = \{\mathbf{b}\}$
- $\mathrm{FOLLOW}(S) = \mathrm{FOLLOW}(S') = \{\mathbf{e}, \$\}$, $\mathrm{FOLLOW}(E) = \{\mathbf{t}\}$
- $\mathrm{FIRST}(\mathbf{e}\ S) = \mathrm{FIRST}(\mathbf{e}) = \{\mathbf{e}\}$ implica $M(S', \mathbf{e}) := S' \rightarrow \mathbf{e}\ S$
- $\mathrm{FIRST}(\varepsilon) = \{\varepsilon\}$ e $\mathrm{FOLLOW}(S') = \{\mathbf{e}, \$\}$ implicano $M(S', \mathbf{e}) := S' \rightarrow \varepsilon$

Grammatiche LL(1)

- GRAMMATICHE LL(1): una grammatica si dice LL(1) se esiste una tabella per il parsing predittivo che non ha entrate multiple
- LL(1): la prima L indica che l'input viene scandito da sinistra verso destra (<u>L</u>eft); la seconda L indica che il parsing produce una derivazione leftmost della stringa; 1 è il numero di simboli di lookahead necessari

Un esempio

Consideriamo la seguente grammatica

$$
\begin{aligned}
S &\rightarrow B \\
B &\rightarrow T \vee B \mid T \mid [B \Rightarrow B; B] \\
T &\rightarrow F \wedge T \mid F \\
F &\rightarrow (B) \mid t \mid f
\end{aligned}
$$

e calcoliamo *FIRST* e *FOLLOW* dei suoi non terminali

	FIRST	FOLLOW
S	$\{(, t, f, [\ \}$	$\{\$\}$
B	$\{(, t, f, [\ \}$	$\{\$, \Rightarrow, ;,],)\ \}$
T	$\{(, t, f\}$	FOLLOW(B) $\cup\{\vee\} = \{\$, \Rightarrow, ;,],), \vee\}$
F	$\{(, t, f\}$	FOLLOW(T) $\cup\{\wedge\} = \{\$, \Rightarrow, ;,],), \vee, \wedge\}$

Un esempio

Se applichiamo la procedura per la costruzione della tabella di parsing otteniamo delle entrante multidefinite sulla riga del non terminale B come indicato di seguito

	t	f	$\vee$	$\wedge$	(	)	[	]	;	$\Rightarrow$	\$
S	$S \to B$	$S \to B$			$S \to B$		$S \to B$				
B	$B \to T \vee B$ $B \to T$	$B \to T \vee B$ $B \to T$			$B \to T \vee B$ $B \to T$						

Un esempio

Problema: la grammatica non è nel formato "giusto", dobbiamo fattorizzare a sinistra per eliminare le cause di questa ambiguità:

Otteniamo così la seguente grammatica G'

$$
\begin{aligned}
&S \longrightarrow B \\
&B \longrightarrow TB' \mid [B \Rightarrow B; B] \\
&B' \longrightarrow \vee B | \varepsilon \\
&T \longrightarrow FT' \\
&T' \longrightarrow \wedge T \mid \varepsilon \\
&F \longrightarrow (B) \mid t \mid f
\end{aligned}
$$

Homework:

1. calcolare la *FIRST* e la *FOLLOW* dei non terminali della grammatica G
2. e costruire la relativa tabella (vedi Esercizio 2.2 su Analisi Sintattica)