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
  - un tentativo di ottenere una derivazione leftmost della stringa in input
  - ad ogni passo espandono il non terminale più a sinistra che non ha figli
- Parser con backtracking
  - possono dover ritornare sulle proprie scelte nel momento in cui si accorgoni 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 simboli (tipicamente 1) necessari per prendere la scelta giusta (simboli di lookahead)

Parser predittivi

- Per ogni non terminale A con alternative $A \longrightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ e per ogni simbolo di lookahead a esiste una sola alternativa di A in grado di generare stringhe che cominciano per a
- Esempio:

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

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

Parser predittivi ricorsivi: un esempio

- Consideriamo la seguente grammatica:

$$
\begin{array}{lcl}
\textit{type} & \longrightarrow & \textit{simple} \\
 & | & \uparrow \textbf{id} \\
 & | & \textbf{array}[\textit{simple}]\ \textbf{of}\ \textit{type} \\
\textit{simple} & \longrightarrow & \textbf{integer} \\
 & | & \textbf{char} \\
 & | & \textbf{num}\quad \textbf{dotdot}\quad \textbf{num}
\end{array}
$$

- Di cosa abbiamo bisogno per scrivere un parser predittivo ricorsivo per la grammatica data? Di una procedura che:
  - matcha con i simboli terminali (e fa scorrere il simbolo di lookahead)
  - per ogni non terminale e per ogni simbolo di lookahead riconosce quale produzione della grammatica utilizzare

La procedura type

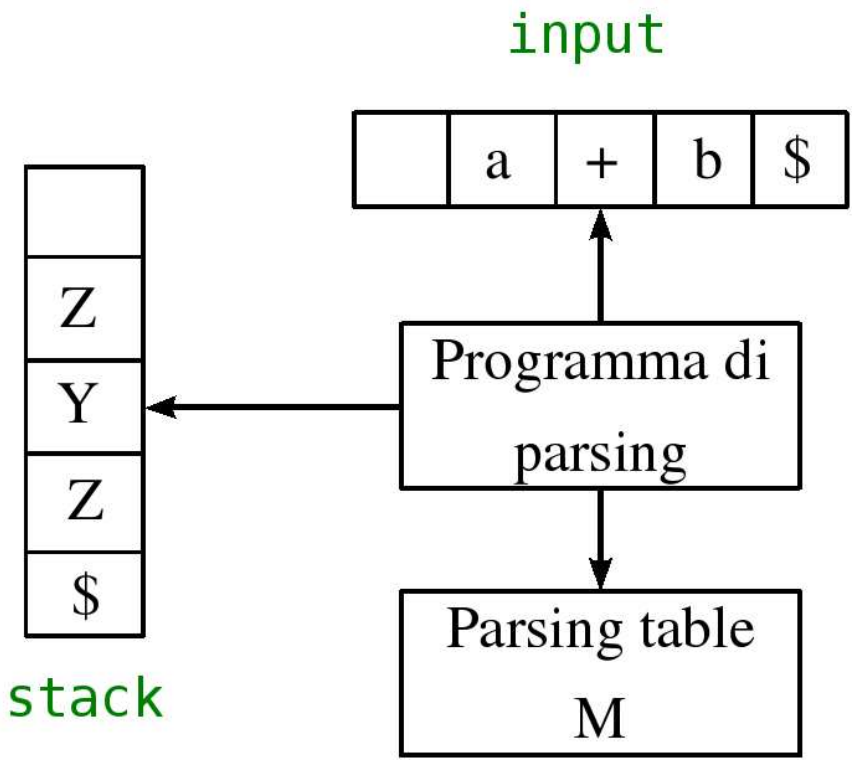
```
procedure type;
begin
    if (lookahead is in {integer,  char,  num}) then
        simple(); //type → simple
    else if (lookahead = '↑') then
        begin //type →↑ id
            match(↑); match(id);
        end
    else if (lookahead = array) then
            begin //type → array[simple] of type
                match(array); match('['); simple(); match(']');
                match(of); type();
            end
            else error;
end;
```

Le procedura simple e match

```
procedure simple;
begin
    if (lookahead = integer) then
        match(integer); //simple → integer
    else if (lookahead = char) then
            match(char);//simple → char
        else if (lookahead = num) then //simple → num dotdot num
                match(num); match(dotdot); match(num);
            else error;
end;

procedura match (t:token);
begin
    if (lookahead = t) then
        lookahead = next_token(); cerca il prossimo token
    else error;
end;
```

Parser predittivi non ricorsivi: struttura



Lo stack contiene simboli terminali, non terminali ed il simbolo \$

M è una tabella indicizzata da non terminali e da simboli in $\Sigma \cup \{\$\}$. Dato un non terminale A ed un simbolo a in $\Sigma \cup \{\$\}$, $M[A, a]$ restituisce una produzione della forma $A \rightarrow \alpha$ oppure un errore. $M[A, a]$ indica quale mossa eseguire

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

Il comportamento del parser dipende dal simbolo X in testa allo stack e dal simbolo corrente di input a:

1. Se $X = a = \$$: il parser termina con successo
2. Se $X = a \neq \$$: elimina il simbolo a in testa allo stack (pop(a)) e fa avanzare il simbolo di lookahead
3. Se X è un non terminale consulta l'entrata $M[X, a]$ della tabella di parsing. Abbiamo due possibili casi:
   i. $M[X, a] := X \rightarrow UVW$: elimina X dallo stack ed inserisce i simboli W, V e U. L'ordine di inserimento dei simboli non è casuale: il simbolo più a sinistra (in questo caso U) deve trovarsi in testa alla pila (pop(X); push(W); push(V); push(U)). Stampa la produzione $X \rightarrow UVW$

   ii. $M[X, a] := error$: il parser chiama una procedura di recovery dell'errore

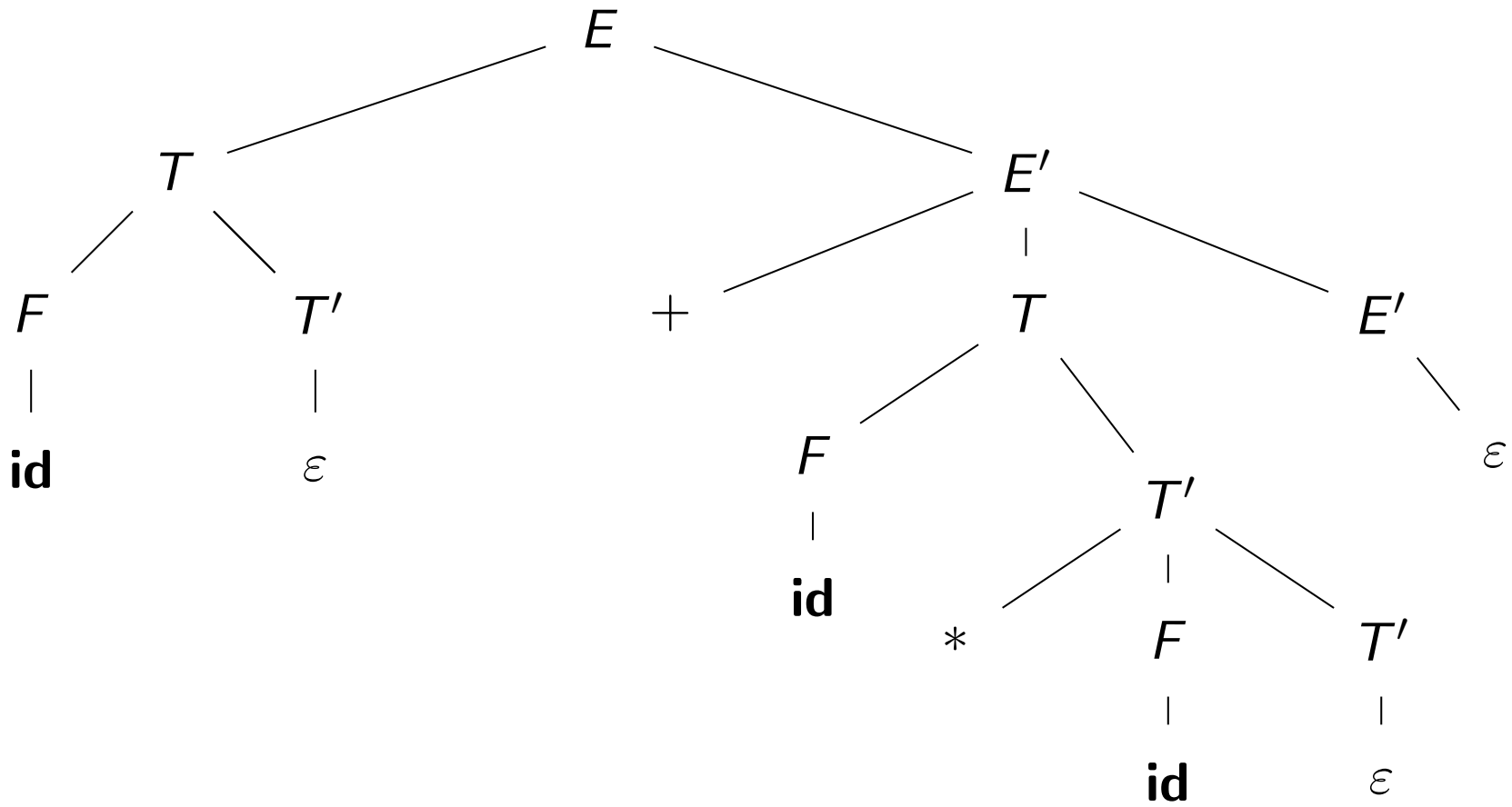
Un Esempio

	id	$+$	$*$	$($	$)$	\$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon'$	$E' \to \varepsilon'$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon'$	$T' \to \varepsilon'$
F	$F \to$ **id**			$F \to (E)$		

Un Esempio

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'	\$	$T' \rightarrow \varepsilon$
\$'	\$	$E' \rightarrow \varepsilon$

Output



Due funzioni ausiliarie: FIRST e FOLLOW

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

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

  - può anche contenere la stringa ε:

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

Due funzioni ausiliarie: FIRST e FOLLOW

- La funzione 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:

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

  - può anche contenere il simbolo \$:

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

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 $X \longrightarrow \varepsilon$, aggiungi ε a FIRST(X)
- Se X è un non terminale ed $X \longrightarrow Y_1 Y_2 \ldots Y_k$ è una produzione per X:

1. aggiungi in FIRST(X) ogni terminale a tale che $a \in$ FIRST(Y_j), con $j \in [1, k]$, ed $\varepsilon \in$ FIRST(Y_1), FIRST(Y_2), $\ldots$, FIRST(Y_{j-1})

2. se, per ogni $j \in [1, k]$, $\varepsilon \in$ FIRST(Y_j), aggiungi ε in FIRST(X)

FIRST di simboli

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

1. inizialmente, aggiungiamo a FIRST(X) ogni terminale in FIRST(Y_1)
2. se $\varepsilon \notin$ FIRST(Y_1) non aggiungiamo ulteriori elementi; al contrario, se $\varepsilon \in$ 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 $\varepsilon \notin$ FIRST(Y_2) ...
5. ε viene agginta a FIRST(X) se, per ogni $j = 1, \ldots, k$, $\varepsilon \in$ FIRST(Y_j)

Un esempio

$E \to TE'$
$E' \to +TE' \mid \varepsilon$

$T \to FT'$
$T' \to *FT' \mid \varepsilon$

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

- Dalle produzioni $F \to (E)$ ed $F \to \mathbf{id}$ abbiamo che $\text{FIRST}(F) = \{(, \mathbf{id}\}$
- Dalle produzioni $T' \to *FT$ e $T' \to \varepsilon$ abbiamo che $\text{FIRST}(T') = \{*, \varepsilon\}$
- $T \to FT'$ ed $\varepsilon \notin \text{FIRST}(F) = \{(, \mathbf{id}\}$ implica $\text{FIRST}(T)) = \text{FIRST}(F) = \{(, \mathbf{id}\}$
- Dalle produzioni $E' \to +TE$ ed $E' \to \varepsilon$ abbiamo che $\text{FIRST}(E') = \{+, \varepsilon\}$
- $E \to TE'$ ed $\varepsilon \notin \text{FIRST}(T) = \{(, \mathbf{id}\}$ implica $\text{FIRST}(E)) = \text{FIRST}(T) = \{(, \mathbf{id}\}$

FIRST di stringhe

Sia $\alpha = X_1 X_2 \dots X_n$ una stringa di simboli della grammatica; la FIRST(α) viene calcolato applicando le seguenti regole:

- Aggiungi a FIRST(α) tutti i simboli di FIRST(X_1) tranne ε
- Se $\varepsilon \in$ FIRST(X_1), aggiungi a FIRST(α) tutti i simboli di FIRST(X_2) tranne ε
- Se $\varepsilon \in$ FIRST(X_2), aggiungi a FIRST(α) tutti i simboli di FIRST(X_3) tranne ε

  ...
- Se, per ogni $j = 1, \dots, n$, $\varepsilon \in$ FIRST(X_j), aggiungi ε a FIRST(α)

FOLLOW

- La funzione FOLLOW(B) è costruita a partire da quelle produzioni che contengono il non terminale B nella parte destra in base alle seguenti regole:
  1. inserisci il simbolo speciale \$ in FOLLOW(S), dove S è il simbolo iniziale della grammatica
  2. per ogni produzione della forma $A \rightarrow \alpha\, B\, \beta$, aggiungi in FOLLOW(B) ogni terminale in FIRST(β)
  3. per ogni produzione della forma $A \rightarrow \alpha\, B$ o della forma $A \rightarrow \alpha\, B\, \beta$ con $\beta \stackrel{*}{\Rightarrow} \varepsilon$, 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

Assumiamo che $A \longrightarrow \alpha\, B$ sia una produzione della grammatica e sia $a \in$ FOLLOW(A):

- Se $a \in$ FOLLOW(A) allora $S \stackrel{*}{\Rightarrow} \alpha_1\, A\, a\, \beta_1$
- Applicando la produzione $A \longrightarrow \alpha\, B$, abbiamo che

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

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

- In maniera analoga se $A \longrightarrow \alpha\, B\, \beta$ con $\beta \stackrel{*}{\Rightarrow} \varepsilon$ è una produzione della grammatica ed $a \in$ FOLLOW(A) allora:

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

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

Un esempio

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$

$$T \to FT'$$
$$T' \to *FT' \mid \varepsilon$$

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

- Produzioni con E a destra: $F \to (E)$. Applichiamo la seconda regola con $\beta =)$ e FIRST$(\beta) = \{)\}$. Inoltre E è il simbolo iniziale. Allora FOLLOW(E) $= \{), \$\}$
- Produzioni con E' a destra: $E \to TE'$ ed $E' \to +TE'$. Dalle regola 3, tutti i simboli in FOLLOW(E) ed in FOLLOW(E') vanno aggiunti in FOLLOW(E'). Quindi, FOLLOW(E')= FOLLOW(E)= $\{), \$\}$
- Produzioni con T a destra: $E \to TE'$ ed $E' \to +TE'$ con FIRST(E')= $\{+, \varepsilon\}$. 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

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

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

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

- La FOLLOW(T') si ottiene in maniera simile alla FOLLOW(E') perchè T' compare nelle produzioni $T \rightarrow FT$ e $T' \rightarrow *FT'$ come ultimo simbolo a destra. FOLLOW(T') = FOLLOW(T)= $\{+,), \$\}$
- produzioni con F a destra: $T \rightarrow FT'$ con FIRST(T') = $\{*, \varepsilon\}$. Inanzittutto aggiungiamo $*$ a FOLLOW(F). Inoltre, poichè, $T' \rightarrow \varepsilon$, dobbiamo aggiungere a FOLLOW(F) tutti i simboli in FOLLOW(T) = $\{+,), \$\}$. Quindi 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 $a \in \text{FIRST}(\alpha)$, aggiungi $A \rightarrow \alpha$ ad $M(A, a)$
3. Se $\varepsilon \in \mathit{FIRST}(\alpha)$ aggiungi $A \rightarrow \alpha$ ad $M(A, b)$ per ogni terminale $b \in \text{FOLLOW}(A)$; se $\varepsilon \in \mathit{FIRST}(\alpha)$ ed \$ $\in \text{FOLLOW}(A)$ aggiungi $A \rightarrow \alpha$ ad $M(A, \$)$
4. Poni ogni entrata indefinita ad error

Un esempio

$E \to TE'$
$E' \to +TE' \mid \varepsilon$

$T \to FT'$
$T' \to *FT' \mid \varepsilon$

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

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

Consideriamo le seguenti produzioni:

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

Un Esempio

	id	$+$	$*$	$($	$)$	\$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon'$	$E' \to \varepsilon'$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon'$	$T' \to \varepsilon'$
F	$F \to$ **id**			$F \to (E)$		

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 $S' \rightarrow \varepsilon$
- $\text{FIRST}(S) = \{\mathbf{i}, \mathbf{a}\}$, $\text{FIRST}(S') = \{\mathbf{e}, \varepsilon\}$, $\text{FIRST}(E) = \{\mathbf{b}\}$
- $\text{FOLLOW}(S) = \text{FOLLOW}(S') = \{\mathbf{e}, \$\}$, $\text{FOLLOW}(E) = \{\mathbf{t}\}$
- $\text{FIRST}(\mathbf{e}\ S) = \text{FIRST}(\mathbf{e}) = \{\mathbf{e}\}$ implica $M(S', \mathbf{e}) := S' \rightarrow \mathbf{e}\ S$
- $\text{FIRST}(\varepsilon = \{\varepsilon\}$ e $\text{FOLLOW}(S') = \{\mathbf{e}, \$\}$ 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 (Left); la seconda L indica che il parsing produce una derivazione leftmost della stringa; 1 è il numero di simboli di lookahead necessari