

# Parsing top down

---

a.a. 2020-2021

Corso di Fondamenti di Informatica - 1 modulo

Corso di Laurea in Informatica

Università di Roma "Tor Vergata"

Prof. Giorgio Gambosi



In una derivazione sinistra di una stringa, una forma di frase è necessariamente del tipo  $V_T^+(V_T \cup V_N)^*$ .

Esempio: la grammatica

$$\begin{aligned}T &\longrightarrow R \mid aTc \\ R &\longrightarrow RbR \mid \varepsilon\end{aligned}$$

e la produzione sinistra

$$\begin{aligned}T &\Longrightarrow \underline{a}Tc \Longrightarrow \underline{aa}Tcc \Longrightarrow \underline{aa}Rcc \Longrightarrow \underline{aa}RbRcc \Longrightarrow \underline{aa}RbRbRcc \Longrightarrow \\ &\underline{aab}RbRcc \Longrightarrow \underline{aab}RbRbRcc \Longrightarrow \underline{aabb}RbRcc \Longrightarrow \underline{aabbb}Rcc \Longrightarrow \underline{aabbbcc}\end{aligned}$$

# Parsing predittivo

- Nel corso di un parsing predittivo con input  $x$ , alla forma di frase  $wA\alpha$ , con  $w \in V_T^*$ ,  $A \in V_N$ ,  $\alpha \in (V_T \cup V_N)^+$ , corrisponde una situazione in cui il parser ha letto il prefisso  $w$  della stringa  $x$  e deve determinare, sulla base di esso e della rimanente parte  $z$  della stringa  $x = wz$ , quale delle produzioni aventi  $A$  a sinistra applicare.
- Se la produzione selezionata è  $A \rightarrow yB\beta$ , con  $y \in V_T^*$ ,  $B \in V_N$ ,  $\beta \in (V_T \cup V_N)^+$ , la nuova forma di frase è  $wyB\beta\alpha$
- Il parser ad ogni istante fa riferimento alla forma di frase attuale e alla parte di stringa di input ancora da leggere: all'inizio evidentemente queste informazioni sono l'assioma  $S$  e l'intera stringa  $x$  in input

# Parsing predittivo

$$T \longrightarrow R \mid aTc$$

$$R \longrightarrow RbR \mid \varepsilon$$

letta	forma di frase	stringa	operazione
$\varepsilon$	<u>T</u>	aabbbcc	$T \longrightarrow aTc$
a	a <u>T</u> c	abbbcc	$T \longrightarrow aTc$
aa	aa <u>T</u> cc	bbbcc	$T \longrightarrow R$
aa	aa <u>R</u> cc	bbbcc	$R \longrightarrow RbR$
aa	aa <u>R</u> bRcc	bbbcc	$R \longrightarrow RbR$
aa	aa <u>R</u> bRbRcc	bbbcc	$R \longrightarrow \varepsilon$
aab	aab <u>R</u> bRcc	bbcc	-
aab	aab <u>R</u> bRcc	bbcc	$R \longrightarrow \varepsilon$
aabb	aabb <u>R</u> cc	bcc	-
aabb	aabbcc	cc	$R \longrightarrow \varepsilon$
aabbcc	aabbcc	c	-
aabbcc	aabbcc	$\varepsilon$	-

# Parsing a discesa ricorsiva

Implementazione di parser top down: una funzione  $A()$  per ogni  $A \in V_N$ , con la struttura seguente. Il programma inizia da  $S()$

$A()$ :

**for each**  $A \longrightarrow X_1 X_2 \cdots X_k \in P$ :

**for**  $i$  in range( $1, k + 1$ ):

**if**  $X_i \in V_N$ :

**if not**  $X_i()$ :

**break**

**else :**

**if**  $X_i$  uguale al prossimo simbolo  $a$  della stringa:

          avanza al simbolo successivo

**else :**

**break**

**return True**

**return False**

# Parsing a discesa ricorsiva

- Utilizzo del **backtracking**: esplorazione ricorsiva di tutte le possibilità
- Backtracking: trial and error
- La grammatica non può essere **ricorsiva sinistra**

$$A \longrightarrow Aw$$

- Può essere molto inefficiente
- Può essere reso efficiente se la scelta della produzione da considerare può essere guidata dall'esame dei caratteri successivi

## Grammatica

$$E \longrightarrow TE'$$

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

$$T \longrightarrow FT'$$

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

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

## Stringa

id+id\*id

# Parsing predittivo

$\varepsilon$	<u>E</u>	<u>id</u> +id*id	$E \longrightarrow TE'$
$\varepsilon$	<u>TE'</u>	<u>id</u> +id*id	$T \longrightarrow FT'$
$\varepsilon$	<u>FT'E'</u>	<u>id</u> +id*id	$F \longrightarrow \text{id}$
id	id <u>TE'</u>	<u>+</u> id*id	$T' \longrightarrow \varepsilon$
id	id <u>E'</u>	<u>+</u> id*id	$E' \longrightarrow +TE'$
id+	id+ <u>TE'</u>	<u>id</u> *id	$T \longrightarrow FT'$
id+	id+ <u>FT'E'</u>	<u>id</u> *id	$F \longrightarrow \text{id}$
id+id	id+id <u>TE'</u>	<u>*</u> id	$T' \longrightarrow *FT'$
id+id*	id+id* <u>FT'E'</u>	<u>id</u>	$F \longrightarrow \text{id}$
id+id*id	id+id*id <u>TE'</u>	$\varepsilon$	$T' \longrightarrow \varepsilon$
id+id*id	id+id*id <u>E'</u>	$\varepsilon$	$E' \longrightarrow \varepsilon$
id+id*id	id+id*id	$\varepsilon$	-

Ad ogni passo è possibile selezionare una sola produzione, guardando un solo terminale (token)



# Parsing predittivo efficiente

Se per ogni simbolo non terminale da espandere i prossimi  $k$  caratteri della stringa consente di individuare la produzione da applicare, il parser è  $LL(k)$

- Left-to-right: la derivazione è calcolata da sinistra a destra (dalla prima produzione applicata all'ultima)
- Leftmost derivation: la derivazione calcolata è sinistra
- $k$  simboli (di **look-ahead**) da considerare

Un linguaggio CF è  $LL(k)$  se esiste un parser  $LL(k)$  che può effettuare l'analisi sintattica

# Costruzione di un parser LL: la funzione FIRST

Consideriamo il caso  $LL(1)$ , per semplicità.

- Per ogni sequenza  $\alpha \in (V_T \cup V_N)^+$ ,  $FIRST(\alpha)$  è l'insieme dei terminali che possono comparire all'inizio di una forma di frase derivata da  $\alpha$
- quindi,  $c \in FIRST(\alpha)$  se e solo se esiste  $\beta \in (V_T \cup V_N)^*$  e  $\alpha \xRightarrow{*} c\beta$

# Costruzione di un parser LL: la funzione FIRST

- Siamo in particolare interessati a  $FIRST(\alpha)$  se  $\alpha$  è la parte destra di una produzione  $A \longrightarrow \alpha$
- Questo perché se per un qualunque  $c$ , se  $c \in FIRST(\alpha)$  e  $A \longrightarrow \alpha$ , allora una stringa che inizia per  $c$  potrebbe essere derivata a partire da  $A$ , in quanto  $A \Longrightarrow \alpha \xRightarrow{*} c\beta$

# Costruzione di un parser LL: la funzione FIRST

Date le  $A$ -produzioni in  $P$

$$A \longrightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$$

- se ogni terminale appartiene a non più di un insieme  $\text{FIRST}(\alpha_i)$ , allora possiamo sempre individuare quale produzione applicare per riscrivere  $A$ , esaminando il solo prossimo carattere  $c$
- infatti, va applicata  $A \longrightarrow \alpha_i$  se e solo se  $c \in \text{FIRST}(\alpha_i)$
- se non esiste  $\alpha_i$  tale che  $c \in \text{FIRST}(\alpha_i)$ , c'è un errore e la parte di stringa da leggere non è derivabile a partire da  $A$

# Costruzione della funzione FIRST

Per la costruzione di  $\text{FIRST}(\alpha)$  va utilizzato il predicato  $\text{Nullable}(\beta)$  definito come  $\text{Nullable}(\beta) = \text{TRUE}$  se e solo se  $\beta$  è **annullabile**, cioè se e solo se esiste una derivazione  $\beta \xRightarrow{*} \varepsilon$ . Una produzione  $B \rightarrow \beta$  è **annullabile** se e solo se  $\beta$  è annullabile.

La costruzione di  $\text{Nullable}(\beta)$  è basata sulle seguenti proprietà

$$\text{Nullable}(\varepsilon) = \text{TRUE}$$

$$\text{Nullable}(a) = \text{False} \quad \forall a \in V_T$$

$$\text{Nullable}(\alpha\beta) = \text{Nullable}(\alpha) \wedge \text{Nullable}(\beta)$$

$$\text{Nullable}(A) = \bigvee_i \text{Nullable}(\alpha_i) \quad \forall A \in V_N, \forall A \rightarrow \alpha_i \in P$$

# Costruzione della funzione FIRST

La costruzione di  $\text{FIRST}(\alpha)$  avviene in modo simile.

Consideriamo in primo luogo la costruzione di  $\text{FIRST}(X)$ , dove  $X$  è un simbolo della grammatica,  $X \in V_T \cup V_N$

1. Se  $X \in V_T$ , allora  $\text{FIRST}(X) = \{X\}$
2. Se  $X \in V_N$ , per ogni  $X \longrightarrow Y_1 Y_2 \cdots Y_k \in P$  ( $k \geq 1$ ):
  - 2.1  $\text{FIRST}(Y_1) \subseteq \text{FIRST}(X)$
  - 2.2 Per  $i = 2, \dots, k$ , se  $\text{Nullable}(Y_1 \cdots Y_{i-1})$  allora  $\text{FIRST}(Y_i) \subseteq \text{FIRST}(X)$

# Costruzione della funzione FIRST

Costruzione di  $\text{FIRST}(X_1 \cdots X_n)$  da  $\text{FIRST}(X)$  per ogni  $X$ :

- $\text{FIRST}(X_1) \subseteq \text{FIRST}(X_1 \cdots X_n)$
- Per  $i = 2, \dots, n$ , se  $\text{Nullable}(X_1 \cdots X_{i-1})$  allora  $\text{FIRST}(X_i) \subseteq \text{FIRST}(X_1 \cdots X_n)$

# La funzione FIRST

Da quanto detto, se

- $x = cy$  è la stringa da leggere
- $A$  è il terminale da riscrivere

allora le possibili produzioni da applicare sono tutte le  $A \rightarrow \alpha_i$  tali che  $c \in \text{FIRST}(\alpha_i)$ .

Se in tutti i casi possibili c'è al più una di tali produzioni, abbiamo un parser  $LL(1)$ .



# La funzione FIRST

Errore! In realtà, se  $A$  è annullabile,  $cy$  potrebbe essere prodotta in modo diverso:

- Supponiamo che la forma di frase attuale sia  $ABw$ , con  $w \in (V_T \cup V_N)^*$
- dato che  $A$  è annullabile, esiste una derivazione  $A \xRightarrow{*} \varepsilon$

allora,  $x = cy$  potrebbe essere ancora derivabile se  $c \in \text{FIRST}(B)$  (e quindi se  $B \xRightarrow{*} c\beta$ ) in quanto

$$ABw \xRightarrow{*} Bw \xRightarrow{*} c\beta w$$

# La funzione FOLLOW

Definiamo la funzione FOLLOW nel modo seguente:

- Per ogni non terminale  $A \in V_N$ ,  $\text{FOLLOW}(A)$  è l'insieme dei terminali che possono comparire subito dopo  $A$  in una forma di frase derivata da  $S$
- Quindi, dati  $A \in V_N$  e  $c \in V_T$ ,  $c \in \text{FOLLOW}(A)$  se e solo se esistono  $\alpha, \beta \in (V_T \cup V_N)^*$  tali che  $S \xRightarrow{*} \alpha A c \beta$
- In realtà, la funzione  $\text{FOLLOW}(A)$  riveste interesse soltanto se  $\text{Nullable}(A) = \text{TRUE}$ , quindi se esiste una derivazione  $A \xRightarrow{*} \varepsilon$

# La funzione FOLLOW

Durante il parsing:

- Siano  $A$  il non terminale da riscrivere e  $c$  il simbolo attualmente letto
- Se  $c \in \text{FOLLOW}(A)$  e  $\text{Nullable}(A) = \text{TRUE}$

allora la derivazione  $A \xRightarrow{*} \varepsilon$  può portare all'annullamento di  $A$

# Costruzione della funzione FOLLOW

Per tener conto del caso in cui  $A$  potrebbe essere l'ultimo simbolo di una forma di frase, cioè in cui  $S \xRightarrow{*} \alpha A$ , estendiamo la grammatica con:

- un non terminale  $\$$  di fine stringa
- un nuovo assioma  $S'$
- una produzione  $S' \longrightarrow S\$$

Evidentemente,  $A$  può comparire a fine stringa nella prima grammatica se e solo se  $\$ \in \text{FOLLOW}(A)$  nella nuova grammatica.

# Costruzione della funzione FOLLOW

FOLLOW viene costruita a partire da un insieme di vincoli derivati dalle produzioni.

- $\$ \in \text{FOLLOW}(S)$
- Se  $A \rightarrow \alpha B \beta \in P$ , allora  $\text{FIRST}(\beta) \subseteq \text{FOLLOW}(B)$
- se  $A \rightarrow \alpha B \beta \in P$  e  $\text{Nullable}(\beta)$ , allora  $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$
- se  $A \rightarrow \alpha B \in P$  allora  $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$

Grammatica

$$\begin{aligned}E &\longrightarrow TE' \\E' &\longrightarrow +TE' \mid \varepsilon \\T &\longrightarrow FT' \\T' &\longrightarrow *FT' \mid \varepsilon \\F &\longrightarrow (E) \mid \text{id}\end{aligned}$$

$\text{Nullable}(E') = \text{Nullable}(T') = \text{TRUE}$

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

# Esempio

- $\$ \in \text{FOLLOW}(E)$
- $\text{FIRST}(E') = \{+\} \subseteq \text{FOLLOW}(T)$
- $\text{FIRST}(T') = \{*\} \subseteq \text{FOLLOW}(F)$
- $\text{FIRST}(')') = \{)\} \subseteq \text{FOLLOW}(E)$
- $\text{FOLLOW}(E) \subseteq \text{FOLLOW}(E')$
- $\text{FOLLOW}(E) \subseteq \text{FOLLOW}(T)$
- $\text{FOLLOW}(T) \subseteq \text{FOLLOW}(T')$
- $\text{FOLLOW}(E') \subseteq \text{FOLLOW}(T)$
- $\text{FOLLOW}(T') \subseteq \text{FOLLOW}(F)$



Da cui deriva

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

# Tabella di parsing predittivo

Associa ad ogni coppia  $(a, X)$ ,  $a \in V_T$ ,  $X \in V_N$ , un insieme di produzioni (1 se  $LL(1)$ ) da applicare nel caso in cui  $X$  sia il non terminale da riscrivere e  $a$  sia il simbolo letto in input.

Costruzione della tabella  $M$ :

Per ogni produzione  $A \rightarrow \alpha \in P$ :

- se  $\alpha \neq \varepsilon$ , per ogni  $a \in \text{FIRST}(A)$  aggiungi  $A \rightarrow \alpha$  a  $M[A, a]$
- se  $\text{Nullable}(\alpha)$ , per ogni  $b \in \text{FOLLOW}(A)$  aggiungi  $A \rightarrow \alpha$  a  $M[A, b]$

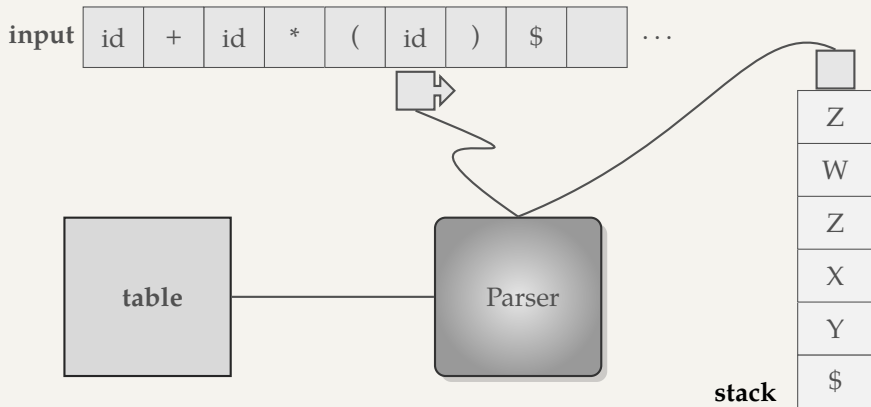
# Esempio

Per la grammatica precedente

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

# Parsing predittivo non ricorsivo

Utilizza uno stack (pila) in modo esplicito, invece che implicitamente, simulando una derivazione sinistra della stringa.



# Parsing predittivo non ricorsivo

```
input.first()
stack.push(S$)
while stack.top() != $:
    if stack.top() == input.current():
        stack.pop()
        input.next()
    elif table[stack.top(), input.current()] != Null:
        Let table[stack.top(), input.current()] be  $X \rightarrow Y_1 \cdots Y_k$ 
        output stack.top()  $\rightarrow Y_1 \cdots Y_k$ 
        stack.pop()
        stack.push( $Y_1 \cdots Y_k$ )
    else:
        error
```

# Parsing predittivo non ricorsivo

Esempio di parsing di  $\text{id} + \text{id} * \text{id}$

Matched	Stack	Input	Action
	E\$	id+id*id\$	
	TE'\$	id+id*id\$	output $E \rightarrow TE'$
	FT'E'\$	id+id*id\$	output $T \rightarrow FT'$
	idT'E'\$	id+id*id\$	output $F \rightarrow \text{id}$
id	T'E'\$	+id*id\$	match id
id	E'\$	+id*id\$	output $T' \rightarrow \varepsilon$
id	+TE'\$	+id*id\$	output $E' \rightarrow +TE'$
id+	TE'\$	id*id\$	match +
id+	FT'E'\$	id*id\$	output $T \rightarrow FT'$
id+	idT'E'\$	id*id\$	output $F \rightarrow \text{id}$
id+id	T'E'\$	*id\$	match id

# Parsing predittivo non ricorsivo

Matched	Stack	Input	Action
id+id	*FT'E'\$	*id\$	output $T' \rightarrow *FT'$
id+id*	FT'E'\$	id\$	match *
id+id*	idT'E'\$	id\$	output $F \rightarrow id$
id+id*id	T'E'\$	\$	match id
id+id*id	E'\$	\$	output $T' \rightarrow \varepsilon$
id+id*id	\$	\$	output $E' \rightarrow \varepsilon$

# Parsing predittivo non ricorsivo

Ne risulta la derivazione sinistra

$E \Rightarrow TE'$	$\Rightarrow FT'E'$
$\Rightarrow idT'E'$	$\Rightarrow idE'$
$\Rightarrow id + TE'$	$\Rightarrow id + FT'E'$
$\Rightarrow id + idT'E'$	$\Rightarrow id + id * FT'E'$
$\Rightarrow id + id * idT'E'$	$\Rightarrow id + id * idE'$
$\Rightarrow id + id * id$	



# Parsing predittivo non ricorsivo

E l'albero sintattico

