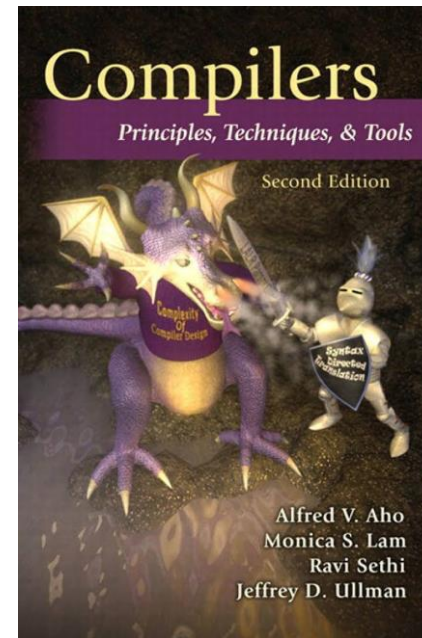


Compiler

Lec 05

Book

Compilers: Principles, Techniques, and Tools is a computer science textbook by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman about compiler construction.



PowerPoint

<http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779>

The screenshot shows a web page for Benha University. The header includes the university logo and name, and a staff search bar with the name 'Ahmed Hassan Ahmed Abu El Atta' and a 'Log out' link. The main content area displays course details for 'Compilers' by 'Ass. Lect. Ahmed Hassan Ahmed Abu El Atta'. The details are organized into several sections:

- Course Information:** A table with the following data:

Course name	Compilers
Level	Undergraduate
Last year taught	2018
Course description	Not Uploaded
- Course password:** A section with a blue header and a white input field.
- Course files:** A section with a blue header and a link to 'add files'.
- Course URLs:** A section with a blue header and a link to 'add URLs'.
- Course assignments:** A section with a blue header and a link to 'add assignments'.
- Course Exams & Model Answers:** A section with a blue header and a link to 'add exams'.

On the left side, there is a navigation menu with links to 'Benha University', 'Home', 'التسفة العربية', 'My C.V.', 'About', 'Courses', 'Publications', 'Inlinks(Competition)', 'Theses', 'Reports', 'Published books', 'Workshops / Conferences', 'Supervised PhD', 'Supervised MSc', 'Supervised Projects', 'Education', 'Language skills', 'Academic Positions', and 'Administrative Positions'. On the right side, there are social media icons for Google, Benha University, RG, LinkedIn, Facebook, Twitter, Google+, YouTube, WordPress, and a general social media icon, along with an '(edit)' link.

Syntax Analysis

PART II

Elimination of Left Recursion

A grammar is left recursive if it has a nonterminal **A** such that there is a derivation **A** \rightarrow **A** α for some string α .

Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion

Direct left recursion

$$A \rightarrow A\alpha$$

Elimination of Left Recursion (cont.)

Direct left recursion

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

Could be replaced by the non-left-recursive productions:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

Example

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \mathbf{id} \end{aligned}$$

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

Elimination of Left Recursion (cont.)

Direct left recursion “general case”

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

Could be replaced by the non-left-recursive productions:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

Example

$S \rightarrow R a \mid A a \mid a$

$R \rightarrow a b$

$A \rightarrow A R \mid A T \mid b$

$T \rightarrow T b \mid a$

Elimination of Left Recursion (cont.)

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

Indirect left recursion :

$$S \Rightarrow Aa \Rightarrow Sda$$

Elimination of Left Recursion (cont.)

Algorithm 4.19: Eliminating left recursion.

INPUT: Grammar G with no cycles or ϵ -productions.

OUTPUT: An equivalent grammar with no left recursion.

- 1) arrange the nonterminals in some order A_1, A_2, \dots, A_n .
- 2) for (each i from 1 to n) {
- 3) for (each j from 1 to $i - 1$) {
- 4) replace each production of the form $A_i \rightarrow A_j\gamma$ by the
 productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \dots \mid \delta_k\gamma$, where
 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ are all current A_j -productions
- 5) }
- 6) eliminate the immediate left recursion among the A_i -productions
- 7) }

Example

$S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \varepsilon$

$S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

$S \rightarrow Aa \mid b$

$A \rightarrow bdA' \mid A'$

$A' \rightarrow cA' \mid adA' \mid \varepsilon$

Example

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow A_1 A_1 \mid a$$

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow A_3 A_1 A_3 A_1 \mid b A_3 A_1 \mid a$$

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow aK \mid b A_3 A_1 K$$

$$k \rightarrow A_1 A_3 A_1 \mid A_1 A_3 A_1 K \mid \varepsilon$$

Example

$C \rightarrow A \mid B \mid f$

$A \rightarrow Cd$

$B \rightarrow Ce$

$A \rightarrow Cd$

$B \rightarrow Ce$

$C \rightarrow A \mid B \mid f$

$C \rightarrow A \mid B \mid f$

$A \rightarrow BdA' \mid fdA'$

$A' \rightarrow dA' \mid \varepsilon$

$B \rightarrow fdA'eB' \mid feB'$

$B' \rightarrow dA'eB' \mid eB' \mid \varepsilon$

$A \rightarrow Cd$

$B \rightarrow Ce$

$C \rightarrow fC'$

$C' \rightarrow dC' \mid eC' \mid \varepsilon$

Left Factoring

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing.

$$\begin{array}{l} \textit{stmt} \quad \rightarrow \quad \mathbf{if\ expr\ then\ stmt\ else\ stmt} \\ \quad \quad \quad | \quad \quad \mathbf{if\ expr\ then\ stmt} \end{array}$$

Left Factoring (cont.)

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 \mid \beta_2$$

Example

$$S \rightarrow i E t S \mid i E t S e S \mid a$$
$$E \rightarrow b$$

$$S \rightarrow i E t S S' \mid a$$
$$S' \rightarrow e S \mid \epsilon$$
$$E \rightarrow b$$

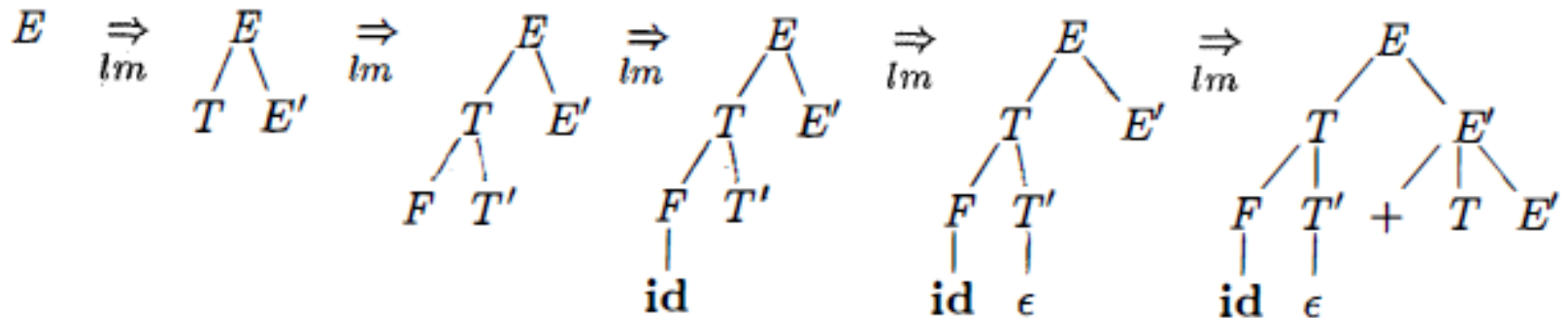
Top-Down Parsing

- Top-down parsing can be viewed as the problem of constructing a parse tree for the input string.
- Starting from the root and creating the nodes of the parse tree in preorder.
- Top-down parsing can be viewed as finding a leftmost derivation for an input string.

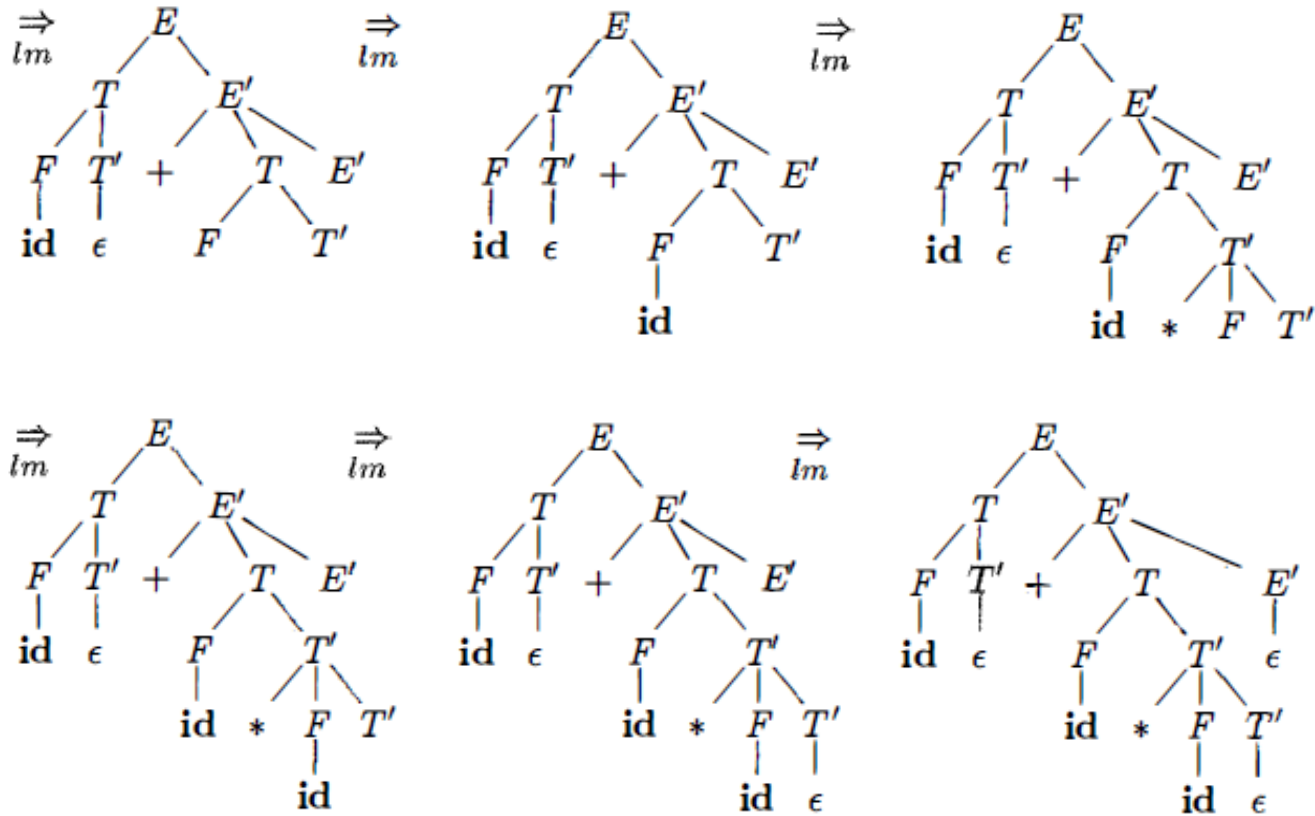
Example

id+id*id

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



Example



FIRST and FOLLOW

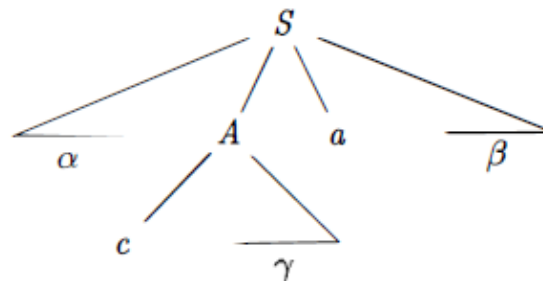
- The construction of both top-down and bottom-up parsers is aided by two functions, **FIRST** and **FOLLOW**, associated with a grammar G .
- During top-down parsing, **FIRST** and **FOLLOW** allow us to choose which production to apply, based on the next input symbol.
- During panic-mode **error recovery**, sets of tokens produced by **FOLLOW** can be used as synchronizing tokens.

FIRST

Define $\text{FIRST}(\alpha)$, where α is any string of grammar symbols, to be the set of terminals that begin strings derived from α .

If $\alpha \Rightarrow^* \varepsilon$, then ε is also in $\text{FIRST}(\alpha)$.

For example, $A \Rightarrow^* c\gamma$, so c is in $\text{FIRST}(A)$.



Terminal “ c ” is in $\text{FIRST}(A)$ and “ a ” is in $\text{FOLLOW}(A)$

FIRST

1. If X is a terminal, then $\text{FIRST}(X) = \{X\}$.
2. If X is a nonterminal and $X \rightarrow Y_1 Y_2 \cdots Y_k$ is a production for some $k \geq 1$, then place a in $\text{FIRST}(X)$ if for some i , a is in $\text{FIRST}(Y_i)$, and ϵ is in all of $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_{i-1})$; that is, $Y_1 \cdots Y_{i-1} \xRightarrow{*} \epsilon$. If ϵ is in $\text{FIRST}(Y_j)$ for all $j = 1, 2, \dots, k$, then add ϵ to $\text{FIRST}(X)$. For example, everything in $\text{FIRST}(Y_1)$ is surely in $\text{FIRST}(X)$. If Y_1 does not derive ϵ , then we add nothing more to $\text{FIRST}(X)$, but if $Y_1 \xRightarrow{*} \epsilon$, then we add $\text{FIRST}(Y_2)$, and so on.
3. If $X \rightarrow \epsilon$ is a production, then add ϵ to $\text{FIRST}(X)$.

FOLLOW

To compute **FOLLOW(A)** for all non-terminals **A**, apply the following rules until nothing can be added to any FOLLOW set:

1. Place **\$** in **FOLLOW(S)** , where **S** is the **start symbol**, and **\$** is the **input right endmarker** .
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in **FIRST(β)** except ϵ is in **FOLLOW(B)** .
3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$, where **FIRST(β)** contains ϵ , then everything in **FOLLOW(A)** is in **FOLLOW(B)** .

Example

$S \rightarrow cAd$

$A \rightarrow ab \mid a$

X	First(X)
cAd	c
ab	a
a	a
A	a
S	c

X	Follow(X)
S	\$
A	d

Example

$$E \rightarrow TE'$$

$$T \rightarrow FT'$$

$$F \rightarrow (E) \mid id$$

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

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

X	First(X)	X	First(X)
id	{id}	T	{(, id}
(E)	{(}	+TE'	{+}
F	{(, id}	E'	{+, ε}
FT'	{}	TE'	{(, id}
T'	{*, ε}	E	{(, id}
FT'	{(, id}		

X	Follow(X)
E	{\$,)}

Example

$E \rightarrow TE'$

Match case 3 **twice** $A \rightarrow \alpha B$ and $A \rightarrow \alpha B\beta$, where **FIRST**(β) contains ε

$A \rightarrow \alpha B$ if $A = E$ and $B = E' \Rightarrow \text{follow}(E) \subset \text{follow}(E')$

$A \rightarrow \alpha B\beta$ if $A = E$ and $B = T$ and $\beta = E'$

$\Rightarrow \text{follow}(E) \subset \text{follow}(T)$

X	Follow(X)	
E	{\$, }	
E'	{\$, }	follow(E) \subset follow(E')
T	{\$, }	follow(E) \subset follow(T)

Example

$E \rightarrow TE'$

Also match case 2

$A \rightarrow \alpha B \beta$, then everything in $\text{FIRST}(\beta)$ except ϵ is in $\text{FOLLOW}(B)$.

Where $A = E$ and $B = T$ and $\beta = E'$

$\text{FIRST}(E')$ except ϵ is in $\text{FOLLOW}(T)$

X	Follow(X)	
E	{\$,)}	
E'	{\$,)}	$\text{follow}(E) \subset \text{follow}(E')$
T	{\$,), +}	$\text{follow}(E) \subset \text{follow}(T)$

Example

$$E' \rightarrow +TE'$$

Case 2 : $\text{FIRST}(E')$ except ε is in $\text{FOLLOW}(T)$ (not new)

Case 3: twice

- $\text{follow}(E') \subset \text{follow}(E')$ (not new)
- $\text{follow}(E') \subset \text{follow}(T)$

X	Follow(X)	
E	{\$,)}	
E'	{\$,)}	$\text{follow}(E) \subset \text{follow}(E')$
T	{\$,), +}	$\text{follow}(E) \subset \text{follow}(T)$ $\text{follow}(E') \subset \text{follow}(T)$

Example

$T \rightarrow FT'$

Case 2 : $FIRST(T') = \{*, \varepsilon\}$ except ε is in $FOLLOW(F)$

Case 3: twice

- $follow(T) \subset follow(T')$
- $follow(T) \subset follow(F)$

X	Follow(X)	
E	{\$,)}	
E'	{\$,)}	follow(E) \subset follow(E')
T	{\$,), +}	follow(E) \subset follow(T) follow(E') \subset follow(T)
T'	{\$,), +}	follow(T) \subset follow(T')
F	{*, \$,), +}	follow(T) \subset follow(F)

Example

$T' \rightarrow *FT'$

Case 2 : $FIRST(T') = \{*, \varepsilon\}$ except ε is in $FOLLOW(F)$
(not new)

Case 3: twice

- $follow(T') \subset follow(T')$ **(not new)**
- $follow(T') \subset follow(F)$

X	Follow(X)	
E	{\$,)}	
E'	{\$,)}	$follow(E) \subset follow(E')$
T	{\$,), +}	$follow(E) \subset follow(T)$ $follow(E') \subset follow(T)$
T'	{\$,), +}	$follow(T) \subset follow(T')$
F	{\$,), +, *}	$follow(T) \subset follow(F)$

Example

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

X	First(X)
iEtSS'	i
a	a
eS	e
S'	e, ϵ
S	i, a
E	b

X	Follow(X)
S	\$, e
S'	\$, e
E	t

