

# Compiler

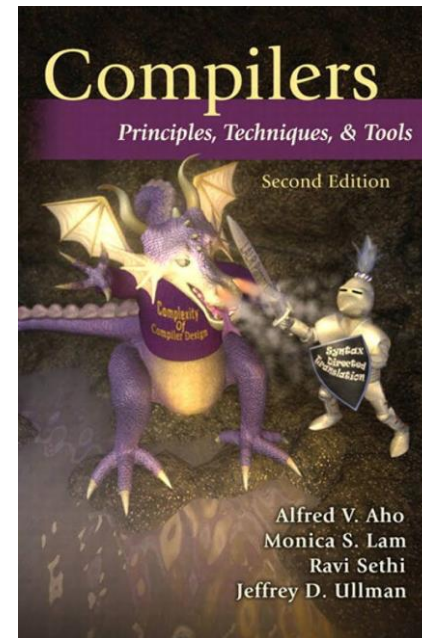
---

Lec 06

# 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 interface for Benha University. The header includes the university logo, the name 'Benha University', and a staff search bar with the name 'Ahmed Hassan Ahmed Abu El Atta' and a 'Log out' link. A navigation menu on the left lists various university-related links. 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: a table for course information, a 'Course password' field, and a list of course-related actions with links to add files, URLs, assignments, and exams. A vertical sidebar on the right contains social media icons and an 'edit' link.

Benha University

Staff Search: **Welcome: Ahmed Hassan Ahmed Abu El Atta** (Log out)

You are in: [Home/Courses/Compilers](#) [Back To Courses](#)

Ass. Lect. Ahmed Hassan Ahmed Abu El Atta :: Course Details:  
Compilers [add course](#) | [edit course](#)

Course name	Compilers
Level	Undergraduate
Last year taught	2018
Course description	Not Uploaded

Course password

Course files	<a href="#">add files</a>
Course URLs	<a href="#">add URLs</a>
Course assignments	<a href="#">add assignments</a>
Course Exams & Model Answers	<a href="#">add exams</a>

(edit)

# Syntax Analysis

---

PART III

# LL ( 1 ) Grammars

---

The first "L" in LL(1) stands for ***scanning*** the ***input*** from ***left to right***,

The second "L" for producing a ***leftmost*** derivation,

The ***"1"*** for using ***one input symbol*** of ***lookahead*** at each step to make parsing action decisions.

# LL ( 1 ) Grammars (cont.)

---

A grammar  $G$  is LL(1) if and only if whenever  $A \rightarrow \alpha \mid \beta$  are two distinct productions of  $G$ , the following conditions hold:

1-  $FIRST(\alpha)$  and  $FIRST(\beta)$  are **disjoint sets**.

2- if  $\epsilon$  is in  $FIRST(\beta)$ , then  $FIRST(\alpha)$  and  $FOLLOW(A)$  are **disjoint sets**, and likewise if  $\epsilon$  is in  $FIRST(\alpha)$ .

# Example

$$E \rightarrow TE'$$

$$T \rightarrow FT'$$

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

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

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

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	{\$, )}	
E'	{\$, )}	follow(E) ⊂ follow(E')
T	{\$, ), +}	follow(E) ⊂ follow(T) follow(E') ⊂ follow(T)
T'	{\$, ), +}	follow(T) ⊂ follow(T')
F	{\$, ), +, *}	follow(T) ⊂ follow(F)

***Is it LL(1) grammar?***

# 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

***Is it LL(1) grammar?***



# 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, $\epsilon$
S	i, a
E	b

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

***Is it LL(1) grammar?***

# LL(1) Parsing Table

---

INPUT: Grammar G.

OUTPUT: Parsing table M.

METHOD : For each production  $A \rightarrow \alpha$  of the grammar, do the following:

1. For each terminal  $a$  in  $\mathbf{FIRST}(\alpha)$ , add  $A \rightarrow \alpha$  to  $M[A, a]$ .
2. If  $\epsilon$  is in  $\mathbf{FIRST}(\alpha)$ , then for each terminal  $b$  in  $\mathbf{FOLLOW}(A)$ , add  $A \rightarrow \alpha$  to  $M[A, b]$ . If  $\epsilon$  is in  $\mathbf{FIRST}(\alpha)$  and  $\$$  is in  $\mathbf{FOLLOW}(A)$ , add  $A \rightarrow \alpha$  to  $M[A, \$]$  as well.

If, after performing the above, there is no production at all in  $M[A, a]$ , then set  $M[A, a]$  to error. (empty entry)

# 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	{\$, )}	
E'	{\$, )}	follow(E) ⊂ follow(E')
T	{\$, ), +}	follow(E) ⊂ follow(T) follow(E') ⊂ follow(T)
T'	{\$, ), +}	follow(T) ⊂ follow(T')
F	{\$, ), +, *}	follow(T) ⊂ follow(F)

# Example

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

NON - INPUT SYMBOL	TERMINAL					
	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	$F \rightarrow id$			$F \rightarrow (E)$		

# Example

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\text{id} + \text{id} * \text{id}\$$	
	$TE'\$$	$\text{id} + \text{id} * \text{id}\$$	output $E \rightarrow TE'$
	$FT'E'\$$	$\text{id} + \text{id} * \text{id}\$$	output $T \rightarrow FT'$
	$\text{id} T'E'\$$	$\text{id} + \text{id} * \text{id}\$$	output $F \rightarrow \text{id}$
$\text{id}$	$T'E'\$$	$+ \text{id} * \text{id}\$$	match $\text{id}$
$\text{id}$	$E'\$$	$+ \text{id} * \text{id}\$$	output $T' \rightarrow \epsilon$
$\text{id}$	$+ TE'\$$	$+ \text{id} * \text{id}\$$	output $E' \rightarrow + TE'$
$\text{id} +$	$TE'\$$	$\text{id} * \text{id}\$$	match $+$
$\text{id} +$	$FT'E'\$$	$\text{id} * \text{id}\$$	output $T \rightarrow FT'$
$\text{id} +$	$\text{id} T'E'\$$	$\text{id} * \text{id}\$$	output $F \rightarrow \text{id}$
$\text{id} + \text{id}$	$T'E'\$$	$* \text{id}\$$	match $\text{id}$
$\text{id} + \text{id}$	$* FT'E'\$$	$* \text{id}\$$	output $T' \rightarrow * FT'$
$\text{id} + \text{id} *$	$FT'E'\$$	$\text{id}\$$	match $*$
$\text{id} + \text{id} *$	$\text{id} T'E'\$$	$\text{id}\$$	output $F \rightarrow \text{id}$
$\text{id} + \text{id} * \text{id}$	$T'E'\$$	$\$$	match $\text{id}$
$\text{id} + \text{id} * \text{id}$	$E'\$$	$\$$	output $T' \rightarrow \epsilon$
$\text{id} + \text{id} * \text{id}$	$\$$	$\$$	output $E' \rightarrow \epsilon$

# Example

---

$S \rightarrow cAd$

$A \rightarrow ab \mid a$

X	Follow(X)
S	\$
A	d

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

NON - INPUT SYMBOL	TERMINAL				
	a	b	c	d	\$
S			$S \rightarrow cAd$		
A	$A \rightarrow ab$ $A \rightarrow a$				

# Example

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

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

X	First(X)
iEtSS'	i
a	a
eS	e
S'	e, $\epsilon$
S	i, a
E	b

NON - TERMINAL	INPUT SYMBOL					
	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

# Error Handle

---

Simple way:

Error message with the name of the **missing token**.



# Error Recovery in Predictive Parsing

---

1. As a starting point, place all symbols in **FOLLOW(A)** into the **synchronizing set** for nonterminal **A**. If we **skip tokens until** an element of **FOLLOW(A)** is seen and **pop A** from the stack, it is likely that parsing can **continue**.

# Error Recovery in Predictive Parsing (cont.)

2- It is not enough to use  $FOLLOW(A)$  as the synchronizing set for A.

For example, if semicolons terminate statements, as in C, then keywords that begin statements may not appear in the FOLLOW set of the nonterminal representing expressions.

$S \rightarrow D;S \mid \varepsilon$        $D \rightarrow id = E \mid int\ id$        $E \rightarrow TE'$

A missing semicolon after an assignment may therefore result in the keyword beginning the next statement being skipped.

$id = int \dots;$

$S \Rightarrow D;S \Rightarrow id = E;S \Rightarrow id = E;S \Rightarrow E;S \Rightarrow skip\ input\ until\ found\ ;\ pop\ E\ and\ contain.$

# Error Recovery in Predictive Parsing (cont.)

---

There is a **hierarchical structure on constructs** in a language; for example, **expressions appear within statements**, **which appear within blocks**, and **so on**.

For example, we might **add keywords** that begin statements to the **synchronizing sets** for the **nonterminals generating expressions**. (more information (Lookahead))

# Error Recovery in Predictive Parsing (cont.)

---

3- If we add symbols in  $FIRST(A)$  to the synchronizing set for nonterminal  $A$ , then it may be possible to resume parsing according to  $A$  if a symbol in  $FIRST(A)$  appears in the input. (skip until first of  $A$ )

# Example

NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
$E'$		$E \rightarrow +TE'$			$E \rightarrow \epsilon$	$E \rightarrow \epsilon$
$T$	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$	$F \rightarrow \text{id}$	synch	synch	$F \rightarrow (E)$	synch	synch

STACK	INPUT	REMARK
$E \$$	) id * + id \$	error, skip )
$E \$$	id * + id \$	id is in $\text{FIRST}(E)$
$TE' \$$	id * + id \$	
$FT'E' \$$	id * + id \$	
id $T'E' \$$	id * + id \$	
$T'E' \$$	* + id \$	
* $FT'E' \$$	* + id \$	
$FT'E' \$$	+ id \$	error, $M[F, +] = \text{synch}$
$T'E' \$$	+ id \$	$F$ has been popped
$E' \$$	+ id \$	
+ $TE' \$$	+ id \$	
$TE' \$$	id \$	
$FT'E' \$$	id \$	
id $T'E' \$$	id \$	
$T'E' \$$	\$	
$E' \$$	\$	
\$	\$	

