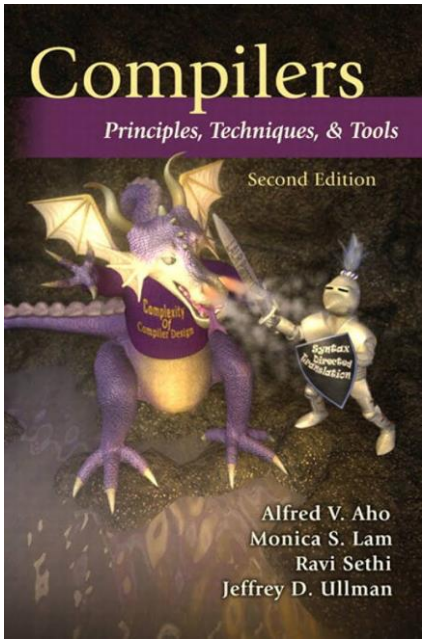
Compiler

Lec 06

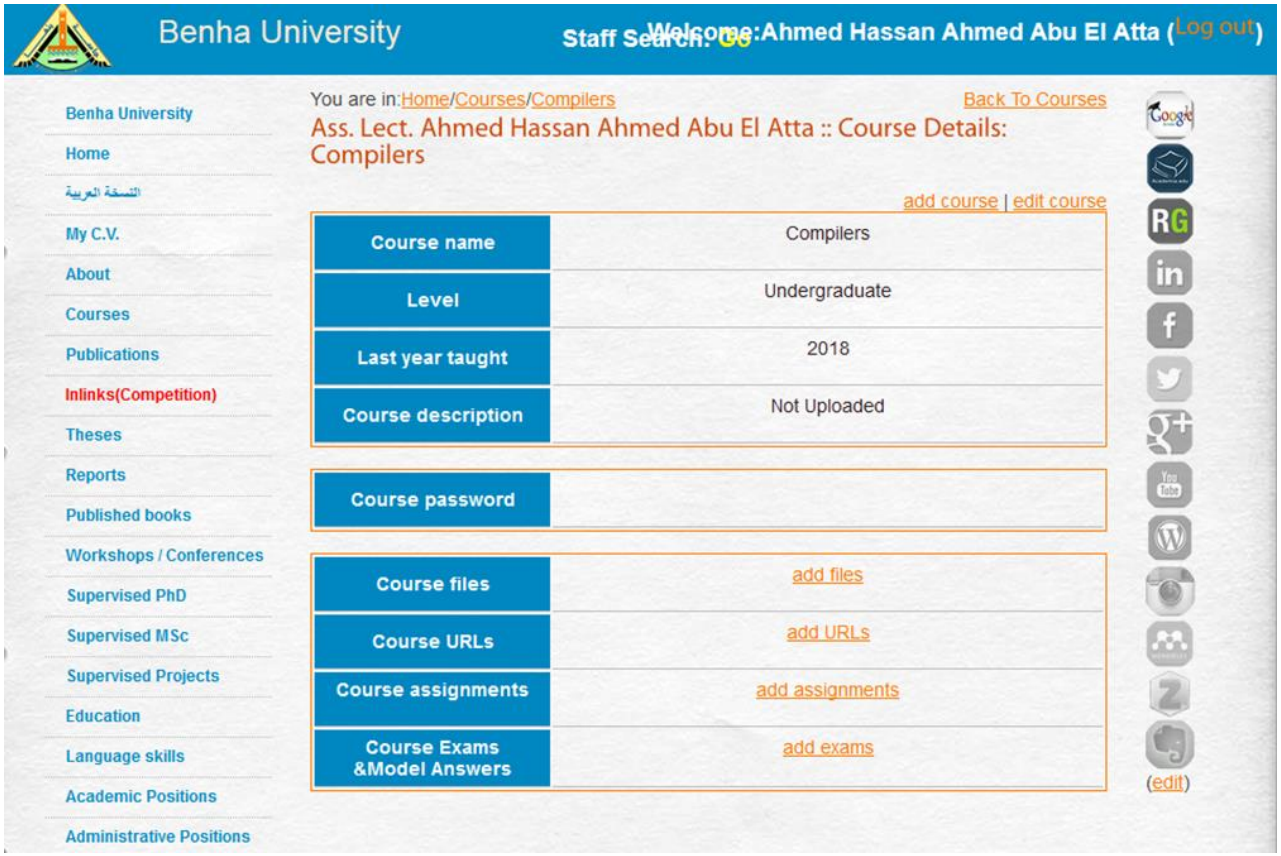
Book

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



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(α) and FIRST(β) are ***disjoint sets***.

2- if ε is in ***FIRST(β)*** , then ***FIRST(α)*** and ***FOLLOW(A)*** are ***disjoint sets***, and likewise if ε is in ***FIRST(α)*** .

Example

E → TE'

T → FT'

F → (E) | id

E' → +TE' | ε

T' → *FT' | ε

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 -> iEtSS’ | a

S’ -> eS | ε

E -> 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

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 ***FIRST***(α) , add A $\rightarrow \alpha$ to M[A, a] .
2. If ε is in ***FIRST***(α) , then for each terminal b in ***FOLLOW***(A) , add A $\rightarrow \alpha$ to M [A, b] . If ε is in ***FIRST***(α) and $\$$ is in ***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 → TE'

T → FT'

F → (E) | id

E' → +TE' | ε

T' → *FT' | ε

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 → TE'

E' → +TE' | ε

T → FT'

T' → *FT' | ε

F → (E) | id

NON - INPUT SYMBOL	TERMINAL					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → id			F → (E)		

Example

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

Example

S → cAd

A→ ab | a

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

X	Follow(X)
S	$
A	d

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

Example

S -> iEtSS’ | a

S’ -> eS | ε

E -> b

X	Follow(X)
S	$,e
S’	$,e
E	t

X	First(X)
iEtSS’	i
a	a
eS	e
S’	e, ε
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 ….;

$S \Longrightarrow D;S \Longrightarrow id = E;S \Longrightarrow id = E;S \Longrightarrow E;S \Longrightarrow$ 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	$E \to TE'$			$E \to TE'$	synch	synch
E'		$E \to +TE'$			$E \to \epsilon$	$E \to \epsilon$
T	$T \to FT'$	synch		$T \to FT'$	synch	synch
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to$ **id**	synch	synch	$F \to (E)$	synch	synch

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