CS-428A Compiler Construction

Assignment-02
“Top-Down Parser”

M. Salman - 212370003

Instructor: Nazifa Fatima
Last Updated: 2023-07-30

Contents

1 Grammar - 01

After Left Factoring

assignment_question → question_body ?$^{(1)}$
question_body → identifier EQ expression$^{(2)}$ | logical_expression$^{(3)}$ | arithmetic_expression$^{(4)}$
expression → logical_expression$^{(5)}$ | arithmetic_expression$^{(6)}$
logical_expression → logical_term logical_expression_tail$^{(7)}$
logical_expression_tail → OR logical_term logical_expression_tail$^{(8)}$ | $\epsilon^{(9)}$
logical_term → logical_factor logical_term_tail$^{(10)}$
logical_term_tail → AND logical_factor logical_term_tail$^{(11)}$ | $\epsilon^{(12)}$
logical_factor → logical_atom LF′$^{(13)}$
LF′ → NOT$^{(14)}$ | $\epsilon^{(15)}$
logical_atom → identifier$^{(16)}$ | TRUE$^{(17)}$ | FALSE$^{(18)}$ | (logical_expression)$^{(19)}$
arithmetic_expression → term arithmetic_expression_tail$^{(20)}$
arithmetic_expression_tail → term arithmetic_expression_tail AET′$^{(21)}$ | $\epsilon^{(22)}$
AET′ → PLUS$^{(23)}$ | *MINUS*$^{(24)}$
term → factor term_tail$^{(25)}$
term_tail → factor term_tail TT′$^{(26)}$ | $\epsilon^{(27)}$
TT′ → MULTIPLY$^{(28)}$ | *DIVIDE*$^{(29)}$
factor → identifier$^{(30)}$ | number$^{(31)}$ | (arithmetic_expression)$^{(32)}$
identifier → ID$^{(33)}$
number → INTEGER$^{(34)}$ | FLOAT$^{(35)}$

1.1 FIRST Sets

- FIRST(**assignment_question**) = FIRST(**question_body**) = FIRST(identifier) ∪ FIRST(logical_expression) ∪ FIRST(arithmetic_expression) = {ID, TRUE, FALSE, (, INTEGER, FLOAT}
- FIRST(**expression**) = FIRST(logical_expression) ∪ FIRST(arithmetic_expression) = {ID, TRUE, FALSE, (, INTEGER, FLOAT}
- FIRST(**logical_expression**) = FIRST(**logical_term**) = {ID, TRUE, FALSE, (}
- FIRST(**logical_expression_tail**) = {OR, ϵ}
- FIRST(**logical_term_tail**) = {AND, ϵ}
- FIRST(**logical_factor**) = FIRST(logical_atom) = {ID, TRUE, FALSE, (}
- FIRST(**LF′**) = {NOT, ϵ}
- FIRST(**logical_atom**) = FIRST(identifier) ∪ {TRUE, FALSE, (} = {ID, TRUE, FALSE, (}
- FIRST(**arithmetic_expression**) = FIRST(**term**) = {(, ID, INTEGER, FLOAT}
- FIRST(**arithmetic_expression_tail**) = FIRST(term) ∪ {ϵ} {ϵ, (, ID, INTEGER, FLOAT}
- FIRST(**AET′**) = {PLUS, MINUS}
- FIRST(**term**) = FIRST(factor) = {(, ID, INTEGER, FLOAT}
- FIRST(**term_tail**) = FIRST(factor) ∪ {ϵ} {ϵ, (, ID, INTEGER, FLOAT}
- FIRST(**TT′**) = {MULTIPLY, DIVIDE}
- FIRST(**factor**) = FIRST(identifier) ∪ FIRST(number) ∪ {(} = {(, ID, INTEGER, FLOAT}
- FIRST(**identifier**) = {ID}
- FIRST(**number**) = {INTEGER, FLOAT}

1.2 FOLLOW Sets

- FOLLOW(**assignment_question**) = {$}
- FOLLOW(**expression**) = FOLLOW(**question_body**) = {?}
- FOLLOW(**logical_expression**) = FOLLOW(question_body) ∪ FOLLOW(expression) ∪ {)} = {?,)}
- FOLLOW(**logical_expression_tail**) = FOLLOW(**logical_expression**) = {?,)}

- FOLLOW(**logical_term_tail**) = FOLLOW(**logical_term**) = FIRST(logical_expression_tail) = {OR, ϵ}
- FOLLOW(**LF′**) = FOLLOW(**logical_factor**) = FIRST(logical_term_tail) = {AND, ϵ}
- FOLLOW(**logical_atom**) = FIRST(LF′) = {NOT, ϵ}
- FOLLOW(**arithmetic_expression**) = FOLLOW(question_body) ∪ FOLLOW(expression) ∪ {)} = {?,)}
- FOLLOW(**AET′**) = FOLLOW(**arithmetic_expression_tail**) = FOLLOW(arithmetic_expression) = {?,)}
- FOLLOW(**term**) = FIRST(arithmetic_expression_tail) = {ϵ, (, INTEGER, FLOAT}
- FOLLOW(**TT′**) = FOLLOW(**term_tail**) = FOLLOW(term) = {ϵ, (, INTEGER, FLOAT}
- FOLLOW(**factor**) = FIRST(term_tail) = {ϵ, (, ID, INTEGER, FLOAT}
- FOLLOW(**identifier**) = FOLLOW(logical_atom) ∪ FOLLOW(factor) ∪ {EQ}
  = {EQ, NOT, ϵ, (, ID, INTEGER, FLOAT}
- FOLLOW(**number**) = FOLLOW(factor) = {ϵ, (, ID, INTEGER, FLOAT}

1.3 Parse Table

- The parsing table is shown in Table 2.

1.4 LL(1)

- From Table 2, we can see that the cells have multiple entries of the production rules. Therefore, given grammar is not LL(1).

1.5 Predictive Parser's Moves

Table 1: Moves made by a predictive parser on input '**NUMBER MULTIPLY FLOAT PLUS ID**'

STACK	**INPUT**
assignment_question $	INTEGER MULTIPLY FLOAT PLUS ID $
question_body ? $	INTEGER MULTIPLY FLOAT PLUS ID $
arithmetic_expression ? $	INTEGER MULTIPLY FLOAT PLUS ID $
term arithmetic_expression_tail ? $	INTEGER MULTIPLY FLOAT PLUS ID $
factor term_tail arithmetic_expression_tail ? $	INTEGER MULTIPLY FLOAT PLUS ID $
number term_tail arithmetic_expression_tail ? $	INTEGER MULTIPLY FLOAT PLUS ID $
INTEGER term_tail arithmetic_expression_tail ? $	INTEGER MULTIPLY FLOAT PLUS ID $
~~***INTEGER***~~ term_tail arithmetic_expression_tail ? $	~~***INTEGER***~~ MULTIPLY FLOAT PLUS ID $
term_tail factor term_tail arithmetic_expression_tail ? $	MULTIPLY FLOAT PLUS ID $

- The predictive parser made moves on the input **NUMBER MULTIPLY FLOAT PLUS ID**, but it encountered an error because the given grammar is not LL(1). This issue arose because there is no production rule in the grammar that allows the parser to predict the expansion of the non-terminal "**term_tail**" when it encounters the terminal "**MULTIPLY**" in the input.

Table 2: Parsing Table *A* for Grammar 01

	?	**EQ**	**OR**	**AND**	**NOT**	**TRUE**	**FALSE**	**(**	**)**	**+** [a]	**-** [b]	***** [c]	**/** [d]	**ID**	**INTEGER**	**FLOAT**	**$**
assignment_question						1	1	1						1	1	1	
question_body						3	3	3, 4						2, 3, 4	4	4	
expression						5, 6	5, 6	5, 6						5, 6			
logical_expression						7	7	7						7			
logical_expression_tail	9		8						9								
logical_term						10	10	10						10			
logical_term_tail			12	11													12
logical_facctor						13	13	13						13			
LF′				15	14												15
logical_atom						17	18	19						16			
arithmetic_expression								20						20	20	20	
arithmetic_expression_tail	22							21	22					21	21	21	
AET′										23	24						
term								25						25	25	25	
term_tail								26, 27						26	26, 27	26, 27	27
TT"												28	29				
factor								32						30	31	31	
identifier														33			
number															34	35	

[a] Symbol used for the terminal ′PLUS′.
[b] Symbol used for the terminal ′MINUS′.
[c] Symbol used for the terminal ′MULTIPY′.
[d] Symbol used for the terminal ′DIVIDE′.
The table uses shorter symbols to ensure it fits within the page.

2 Grammar - 02

After Left Factoring

sentence → noun_phrase verb_phrase$^{(1)}$
noun_phrase → determiner noun NP′$^{(2)}$ | *proper_noun*$^{(3)}$ | *pronoun*$^{(4)}$
NP′ → $\epsilon^{(5)}$ | adjective$^{(6)}$
verb_phrase → verb VP′$^{(7)}$
VP′ → $\epsilon^{(8)}$ | adverb$^{(9)}$ | noun_phrase VP″ $^{(10)}$
VP″ → $\epsilon^{(11)}$ | preposition$^{(12)}$
adjective → “happy”$^{(13)}$ | “red”$^{(14)}$ | “big”$^{(15)}$
adverb → “quickly”$^{(16)}$ | “carefully”$^{(17)}$
determiner → “the”$^{(18)}$ | “a”$^{(19)}$ | “an”$^{(20)}$
preposition → “in”$^{(21)}$ | “on”$^{(22)}$ | “under”$^{(23)}$
verb → “run”$^{(24)}$ | “jump”$^{(25)}$ | “sing”$^{(26)}$ | “eat”$^{(27)}$
noun → “dog”$^{(28)}$ | “cat”$^{(29)}$ | “apple”$^{(30)}$ | “table”$^{(31)}$
proper_noun → “John”$^{(32)}$ | “London”$^{(33)}$ | “July”$^{(34)}$
pronoun → “he”$^{(35)}$ | “she”$^{(36)}$ | “it”$^{(37)}$ | “they”$^{(38)}$

2.1 FIRST Sets

- FIRST(**sentence**) = FIRST(noun_phrase)
  = {the, a, an, John, London, July, he, she, it, they, run, jump, sing, eat}
- FIRST(**noun_phrase**) = FIRST(determiner) ∪ FIRST(proper_noun) ∪ FIRST(pronoun)
  = {the, a, an, John, London, July, he, she, it, they}
- FIRST(**NP′**) = {ϵ, happy, red, big}
- FIRST(**verb_phrase**) = FIRST(**verb**) = {run, jump, sing, eat}
- FIRST(**VP′**) = {ϵ, quickly, carefully, the, a, an John, London, July, he, she, it, they}
- FIRST(**VP″**) = {ϵ, in, on, under}
- FIRST(**adjective**) = {happy, red, big}
- FIRST(**adverb**) = {quickly, carefully}
- FIRST(**determiner**) = {the, a, an}
- FIRST(**preposition**) = {in, on, under}
- FIRST(**noun**) = {dog, cat, apple, table}
- FIRST(**proper_noun**) = {John, London, July}
- FIRST(**pronoun**) = {he, she, it, they}

2.2 FOLLOW Sets

- FOLLOW(**adverb**) = FOLLOW(**VP′**) = FOLLOW(**verb_phrase**) = FOLLOW(**sentence**) = {\$}
- FOLLOW(**NP′**) = FOLLOW(**noun_phrase**) = FIRST(verb_phrase) = {run, jump, sing, eat}
- FOLLOW(**adjective**) = FOLLOW(NP′) = {run, jump, sing, eat}
- FOLLOW(**determiner**) = FIRST(noun) = {dog, cat, apple, table}
- FOLLOW(**preposition**) = FIRST(**VP″**) = {\$}
- FOLLOW(**verb**) = FIRST(VP′) = {ϵ, quickly, carefully, the, a, an, John, London, July, he, she, it, they}
- FOLLOW(**noun**) = FIRST(NP′) = {ϵ, happy, red, big}
- FOLLOW(**pronoun**) = FOLLOW(**proper_noun**) = FOLLOW(noun_phrase) = {run, jump, sing, eat}

2.3 Parse Table

- The parsing table is shown in Table 4.

2.4 LL(1)

- The given grammar is an LL(1) grammar, as there is a single atomic value in each cell of the parse table. See Table 4.

2.5 Predictive Parser's Moves

Table 3: Moves made by a predictive parser on input '**John run quickly**'

MATCHED	STACK	INPUT
	sentence $	John run quickly $
	noun_phrase verb_phrase $	John run quickly $
	proper_noun verb_phrase $	John run quickly $
	John verb_phrase $	John run quickly $
John	verb_phrase $	run quickly $
John	verb VP' $	run quickly $
John	run VP' $	run quickly $
John run	VP' $	quickly $
John run	adverb $	quickly $
John run	quickly $	quickly $
John run quickly	$	$

Table 4: Parsing Table ***B*** for Grammar 02

	sentence	**noun_phrase**	**NP′**	**verb_phrase**	**VP′**	**VP″**	**adj** [a]	**adv** [b]	**determiner**	**preps** [c]	**verb**	**noun**	**proper_noun**	**pronoun**
$					8	11								
happy			6				13							
red			6				14							
big			6				15							
quickly					9			16						
carefully					9			17						
the	1	2			10				18					
a	1	2			10				19					
an	1	2			10				20					
in						12				21				
on						12				22				
under						12				23				
run			5	7							24			
jump			5	7							25			
sing			5	7							26			
eat			5	7							27			
dog												28		
cat												29		
apple												30		
table												31		
John	1	3			10								32	
London	1	3			10								33	
July	1	3			10								34	
he	1	4			10									35
she	1	4			10									36
it	1	4			10									37
they	1	4			10									38

[a] Symbol used for the terminal ’adjective’.
[b] Symbol used for the terminal ’adverb’.
[c] Symbol used for the terminal ’preposition’.
Note: The first column of the table contains terminal symbols, while the first row contain non-terminals.

3 Grammar - 03

After Removing Left Recursion & Left Factoring

program → decl_list$^{(1)}$
decl_list → declaration decl_list′$^{(2)}$
decl_list′ → declaration decl_list′$^{(3)}$ | $\epsilon^{(4)}$
declaration → var_decl$^{(5)}$ | func_decl$^{(6)}$
var_decl → type ID ;$^{(7)}$
type → int$^{(8)}$ | float$^{(9)}$ | char$^{(10)}$
func_decl → type ID (params) compound_stmt$^{(11)}$
params → param_list$^{(12)}$ | void$^{(13)}$
param_list → param param_list′$^{(14)}$
param_list′ → , param param_list′$^{(15)}$ | $\epsilon^{(16)}$
param → type ID$^{(17)}$
compound_stmt → { local_decls stmt_list }$^{(18)}$
local_decls → ϵ local_decls′$^{(19)}$
local_decls′ → var_decl local_decls′$^{(20)}$ | $\epsilon^{(21)}$
stmt_list → ϵ stmt_list′$^{(22)}$
stmt_list′ → stmt stmt_list′$^{(23)}$ | $\epsilon^{(24)}$
stmt → expr_stmt$^{(25)}$ | compound_stmt$^{(26)}$ | selection_stmt$^{(27)}$ | iteration_stmt$^{(28)}$ |return_stmt$^{(29)}$
expr_stmt → expression ;$^{(30)}$ | ;$^{(31)}$
expression → ID = expression$^{(32)}$ | simple_expression$^{(33)}$
simple_expression → additive_expression SE′ $^{(34)}$
SE′ → relop additive_expression$^{(35)}$ | $\epsilon^{(36)}$
additive_expression → term additive_expression′$^{(37)}$
additive_expression′ → addop term additive_expression′$^{(38)}$ | $\epsilon^{(39)}$
term → factor term′$^{(40)}$
term′ → mulop factor term′$^{(41)}$ | $\epsilon^{(42)}$
factor → (expression)$^{(43)}$ | ID$^{(44)}$ | NUM$^{(45)}$
relop → <$^{(46)}$ | <=$^{(47)}$ | >$^{(48)}$ | >=$^{(49)}$ | ==$^{(50)}$ | !=$^{(51)}$
addop → +$^{(52)}$ | -$^{(53)}$
mulop → *$^{(54)}$ | /$^{(55)}$
selection_stmt → if (expression) stmt SS′$^{(56)}$
SS′ → ϵ $^{(57)}$ | else stmt$^{(58)}$
iteration_stmt → while (expression) stmt$^{(59)}$
return_stmt → return expression ;$^{(60)}$

3.1 FIRST Sets

- FIRST(**program**) = FIRST(**decl_list**) = FIRST(**decl_list′**) = FIRST(declaration) = {int,float,char}
- FIRST(**declaration**) = FIRST(var_decl) ∪ FIRST(func_decl) = {int, float, char}
- FIRST(**var_decl**) = FIRST(**func_decl**) = FIRST(**type**) = {int,float,char}
- FIRST(**params**) = FIRST(param_list) ∪ {void} = {int, float, char, void}
- FIRST(**param_list**) = FIRST(**param**) = FIRST(type) = {int, char, float}
- FIRST(**param_list′**) = {, , ϵ}
- FIRST(**compound_stmt**) = {{}
- FIRST(**local_decls**) = FIRST(**stmt_list**) = {ϵ}
- FIRST(**stmt_list′**) = FIRST(stmt) = {;, ID, (, NUM, {, if, while, return}
- FIRST(**local_decls′**) = FIRST(var_decls) ∪ {ϵ} = {int, float, char, ϵ}
- FIRST(**stmt**) = {ID, (, NUM, {, if, while, return, ;}
- FIRST(**expr_stmt**) = FIRST(expression) ∪ {;} = {;, ID, (, NUM)}
- FIRST(**expression**) = {ID} ∪ FIRST(simple_expression) = {ID, NUM, (}

- FIRST(**simple_expression**) = FIRST(**additive_expression**) = FIRST(term) = {(, ID, NUM}
- FIRST(**SE’**) = FIRST(relop) ∪ {ϵ} = {ϵ, <, <=, >, >=,==, !=}
- FIRST(**additive_expression’**) = FIRST(addop) ∪ {ϵ} = {+, -, ϵ}
- FIRST(**term**) = FIRST(factor) = {(, ID, NUM}
- FIRST(**term’**) = FIRST(mulop) ∪ {ϵ} = {ϵ, /, *}
- FIRST(**factor**) = {(, ID, NUM}
- FIRST(**relop**) = {<, <=, >, >=, ==, !=}
- FIRST(**addop**) = {+, -}
- FIRST(**mulop**) = {*, /}
- FIRST(**selection_stmt**) = {if}
- FIRST(**SS’**) = {ϵ, else}
- FIRST(**iteration_stmt**) = {while}
- FIRST(**return_stmt**) = {return}

3.2 FOLLOW Sets

- FOLLOW(**decl_list’**) = FOLLOW(**decl_list**) = FOLLOW(**program**) = {$}
- FOLLOW(**func_decl**) = FOLLOW(**declaration**) = FIRST(decl_list) ∪ FIRST(decl_list’)
  = {int, float, char}
- FOLLOW(**var_decl**) = FOLLOW(declaration) ∪ FIRST(local_decls)
  = {int, float, char, ϵ}
- FOLLOW(**type**) = {ID}
- FOLLOW(**param_list’**) = FOLLOW(**param_list**) = FOLLOW(**params**) = {)}
- FOLLOW(**param**) = FIRST(param_list) ∪ FIRST(param_list’) = {int, char, float, ϵ, ,}
- FOLLOW(**compound_stmt**) = FOLLOW(func_decl) ∪ FOLLOW(stmt)
  = {else, int, float, char, ID, NUM, }, if, while, return, ;}
- FOLLOW(**local_decls’**) = FOLLOW(**local_decls**) = FIRST(stmt_list) = {ϵ}
- FOLLOW(**stmt_list**) = FOLLOW(**stmt_list’**) = {}}
- FOLLOW(**stmt**) = FIRST(stmt_list’) ∪ FOLLOW(selection_stmt) ∪ FOLLOW(iteration_stmt) ∪ {else}
  = {else, ;, ID, NUM, {, if, while, return}
- FOLLOW(**expr_stmt**) = FOLLOW(stmt) = {else, ;, ID, NUM, {, if, while, return}
- FOLLOW(**SE’**) = FOLLOW(**simple_expression**) = FOLLOW(**expression**) = {;,)}
- FOLLOW(**additive_expression’**) = FOLLOW(**additive_expression**) = FIRST(SE’) ∪ FOLLOW(SE’)
  = {<, <=, >, >=, ==, !=, ;,)}
- FOLLOW(**relop**) = FIRST(additive_expression) = {(, ID, NUM}
- FOLLOW(**term’**) = FOLLOW(**term**) = FIRST(additive_expression’) = {+, -, ϵ}
- FOLLOW(**factor**) = FIRST(term’) = {*, /, ϵ}
- FOLLOW(**mulop**) = FOLLOW(**addop**) = FIRST(term) = {(, ID, NUM}
- FOLLOW(**SS’**) = FOLLOW(**selection_stmt**) = FOLLOW(**iteration_stmt**) = FOLLOW(**return_stmt**) = FOLLOW(stmt)
  = {else, ;, ID, NUM, {, if, while, return}

3.3 LL(1)

- From Table 6, we can see there are multiple values in the cells. Therefore, grammar is not LL(1).

3.4 Predictive Parser's Moves

Table 5: Moves made by a predictive parser on input **'float ID; int ID (void) { }'**

MATCHED	STACK	INPUT
	program $	float ID; int ID (void) { } $
	decl_list $	float ID; int ID (void) { } $
	declaration decl_list′ $	float ID; int ID (void) { } $
	var_decl func_decl decl_list′ $	float ID; int ID (void) { } $
	type ID; func_decl decl_list″ $	float ID; int ID (void) { } $
	float ID; func_decl decl_list′ $	float ID; int ID (void) { } $
float	ID; func_decl decl_list′ $	ID; int ID (void) { } $
float ID	; func_decl decl_list′ $	; int ID (void) { } $
float ID;	func_decl decl_list′ $	int ID (void) { } $
float ID;	type ID (params)compound_stmt decl_list′ $	int ID (void) { } $
float ID;	int ID (params) compound_stmt decl_list′$	int ID (void) { } $
float ID; int	ID (params) compound_stmt decl_list′$	ID (void) { } $
float ID; int ID	(params) compound_stmt decl_list′ $	(void) { } $
float ID; int ID (	params) compound_stmt decl_list′ $	void) { } $
float ID; int ID (	void) compound_stmt decl_list′ $	void) { } $
float ID; int ID (void	) compound_stmt decl_list′ $	) { } $
float ID; int ID (void)	compound_stmt decl_list′ $	{ } $
float ID; int ID (void)	{ local_decls stmt_list } decl_list′ $	{ } $
float ID; int ID (void) {	local_decls stmt_list′ } decl_list′ $	} $

- The grammar is not LL(1) due to the presence of multiple values in a single cell of the parse table. There is no production rule in the grammar that allows the parser to predict the expansion of the non-terminal "**local_decls**" when it encounters the terminal "}" in the input, thus halting.

3.5 Parse Table

Table 6: Parsing Table ***C*** for Grammar 03

	<	<=	>	>=	==	!=	+	-	*	/	$
SE′	35	35	35	35	35	35					
additive_expression′	39	39	39	39	39	39	38	38			39
term′							42	42	41	41	42
relop	46	47	48	49	50	51					
addop							52	53			
mulop									54	55	

**** The table has been split, and certain non-terminals like additive_expression′ and SE′ have production rules that span across both parts of the table (partial).***

Table 6: Parsing Table ***C*** for Grammar 03 (...Continued...)

	ID	;	int	float	char	(	)	void	{	}	NUM	if	else	while	return	$
program			1	1	1											
decl_list			2	2	2											
decl_list′			3	3	3											4
declaration			5, 6	5, 6	5, 6											
var_decl			7	7	7											
type			8	9	10											
func_decl			11	11	11											
params			12	12	12			13								
param_list			14	14	14											
param_list′			15	15	15		16									
param			17	17	17											
compound_stmt									18							
local_decls																19
local_decls′			20	20	20											21
stmt_list									22							
stmt_list′	23	23				23			23	24	23	23		23	23	
stmt	25	25				25			26		25	27		28	29	
expr_stmt	30	31				30					30					
expression	32	33				33					33					
simple_expression		34				34					34					
SE′		36					36									
additive_expression	37					37					37					
additive_expression′		39				39										
term	40					40					40					
factor	44					43					45					
selection_stmt												56				
SS′	57	57							57		57	57	57, 58	57	57	
iteration_stmt														59		
return_stmt															60	

4 Grammar - 04

$S \rightarrow AaBbCc^{(1)} \mid dDeEfFgG^{(2)} \mid hH^{(3)} \mid \epsilon^{(4)}$
$A \rightarrow aA^{(5)} \mid \epsilon^{(6)}$
$B \rightarrow bB^{(7)} \mid \epsilon^{(8)}$
$C \rightarrow cC^{(9)} \mid \epsilon^{(10)}$
$D \rightarrow dD^{(11)} \mid \epsilon^{(12)}$
$E \rightarrow eE^{(13)} \mid \epsilon^{(14)}$
$F \rightarrow fF^{(15)} \mid \epsilon^{(16)}$
$G \rightarrow gG^{(17)} \mid \epsilon^{(18)}$
$H \rightarrow hH^{(19)} \mid \epsilon^{(20)}$

4.1 FIRST Sets

- FIRST(**S**) = {d, h, ϵ} $\cup$ FIRST(A) = {a, d, h, ϵ}
- FIRST(**A**) = {a , ϵ}
- FIRST(**B**) = {b , ϵ}
- FIRST(**C**) = {c , ϵ}
- FIRST(**D**) = {d , ϵ}
- FIRST(**E**) = {e , ϵ}
- FIRST(**F**) = {f , ϵ}
- FIRST(**G**) = {g , ϵ}
- FIRST(**H**) = {h , ϵ}

4.2 FOLLOW Sets

- FOLLOW(**S**) = FOLLOW(**G**) = FOLLOW(**H**) = {$}
- FOLLOW(**A**) = {a}
- FOLLOW(**B**) = {b}
- FOLLOW(**C**) = {c}
- FOLLOW(**D**) = {e}
- FOLLOW(**E**) = {f}
- FOLLOW(**F**) = {g}

4.3 Parse Table

- The parsing table is shown in Table 7.

4.4 LL(1)

- Given grammar is not an LL(1) grammar as there are multiple values in single cell of parse table.

4.5 Predictive Parser's Moves

Table 7: Parsing Table ***D*** for Grammar 04

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**$**
S	1			2				3	4, 1
A	5, 6								
B		7, 8							
C			9, 10						
D				11	12				
E					13	14			
F						15	16		
G							17		18
H							20	19	

Table 8: Moves made by a predictive parser on input **'aabbcc'**

MATCHED	**STACK**	**INPUT**	**ACTION**
	S$	aabbcc$	
	AaBbCc$	aabbcc$	output S → AaBbCc
	aAaBbCc$	aabbcc$	output A → aA | ϵ
a	AaBbCc$	abbcc$	output A → aA | ϵ
a	aAaBbCc$	abbcc$	match a
aa	AaBbCc$	bbcc$	entry D[A, b] is blank

- The grammar is not LL(1) because there are multiple values present in a single cell of the parse table. Specifically, there is no unique production to predict when encountering terminal 'b' after matching 'aa'.

5 Grammar - 05

S → aA$^{(1)}$ | Bb$^{(2)}$ | cC$^{(3)}$
A → d$^{(4)}$ | $\epsilon^{(5)}$
B → eB$^{(6)}$ | f$^{(7)}$
C → gC$^{(8)}$ | h$^{(9)}$ | $\epsilon^{(10)}$

5.1 FIRST Sets

- FIRST(**S**) = {a, c} ∪ FIRST(B) = {a, e, f, c}
- FIRST(**A**) = {d, ϵ}
- FIRST(**B**) = {e, f}
- FIRST(**C**)= {g, h, ϵ}

5.2 FOLLOW Sets

- FOLLOW(**S**) = FOLLOW(A) = FOLLOW(C) = {$}
- FOLLOW(**B**) = {b}

5.3 Parse Table

Table 9: Parsing Table ***E*** for Grammar 05

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**$**
S	1		3		2	2			
A				4					5
B					6	7			
C							8	9	10

5.4 LL(1)

- Given grammar is LL(1) as only atomic values are present in each cell of the parse table.

5.5 Predictive Parser’s Moves

Table 10: Moves made by a predictive parser on input **‘efb‘**

MATCHED	**STACK**	**INPUT**	**ACTION**
	S$	efb$	
	Bb$	efb$	output S → Bb
	eBb$	efb$	ouput B → eB
e	Bb$	fb$	match e
e	fb$	fb$	output B → f
ef	b$	b$	match f
efb	$	$	match b