Jesse Shaiher
Jay Vang
Cristian Salinas

Compiler Design
Assignment 7 : Predicative Parsing Table

1. Given the following CFG (no programming required).

E-->E + T
E-->T
T-->T * F
T-->F
F-->(E)
F--> a

(a) Eliminate left-recursions
(b) find members of FIRST and FOLLOW

State	FIRST	FOLLOW
E E' T T' F		

(c) Construct its predictive Parsing Table
(d) trace **a*(a+a)$** by showing the content of the stack and the leftover of the input string during tracing: Stack input

Programming part:

1. Given the following CFG and the Predictive Parsing table. Write a program to trace input strings **(1) (i+i)*i$,(2) i*(i-i)$, (3) i(i+i)$** . Show the content of the stack after each match.

E --> E+T; E--> E- T ;E--> T; T-->T*F; T-->T/F; T-->F; F-->i; F—>(E)

states	i	+	-	*	/	(	)	$
E	TQ					TQ		
Q		+TQ	-TQ				λ	λ
T	FR					FR		
R		λ	λ	*FR	/FR		λ	λ
F	i					(E)		

2. Same as problem no. 2, include the S → aW , W → = E rules to the beginning of the grammar so that the grammar becomes this :

S→ aW W→ = E
E→ E+T
E→ E- T
E→ T
T→ T*F T → T/F T→ F
F→ a F → (E)
NOTE: S is the starting state
Construct the parsing table (not exactly the same as above) and modify your program to trace input statements (i) a=(a + a)*a$, (ii) a=a*(a-a)$, (iii) a=(a+a)a$

① a) $E \to E+T$
$E \to T$
$T \to T*F$
$T \to F$
$F \to (E)$
$F \to a$

⟶

$E \to TE'$
$E' \to +TE'$
$E' \to \lambda$

$T \to FT'$
$T' \to *FT'$
$T' \to \lambda$

$F \to (E)$
$F \to a$

b)

state	First	Follow
E	{ (a }	{ #) }
E'	{ + λ }	{ #) }
T	{ (a }	{ + λ #) }
T'	{ * λ }	{ + λ #) }
F	{ (a }	{ + λ #) }

c)

states	a	+	*	(	)	#
E	TE'			TE'		
E'		+TE'			λ	λ
T	FT'			FT'		
T'		λ	*FT'		λ	λ
F	a			(E)		

d)

stack	input
push $ push E stack: $E	a*(a+a)$
pop: E read: a go to [E,a]=TE' push E', push T stack: $E'T	*(a+a)$
pop: T go to [T,a]=FT' push T' push F stack $E'T'F	
pop: F go to [F,a]=a push a stack $E'T'a	
pop: a match w/ input a stack $E'T'	
pop: T' read: * go to [T',*]=*FT' push T', push F, push * stack $E'T'F*	(a+a)$

stack	input
pop: * match w/ input * stack $E'T'F	
pop: F read: (go to [F,(]=(E) push) push E push (stack $E'T')E(	a+a)$
pop: (match w/ input (stack $E'T')E	
pop: E read a go to [E,a]=TE' push E' push T stack $E'T')E'T	+a)$
pop: T go to [T,a]=FT' push T' push F stack $E'T')E'T'F	
next page continued	

stack	input
pop: F go to [F, a] = a push a stack $E'T')E'T'a	
pop: a match w/ input a stack $E'T')E'T'	
pop: T' read: + go to [T', +] = λ stack $E'T')E'	a) $
pop: E' go to [E', +] = +TE' push E', T, + stack $E'T')E'T+	
pop: + match w/ input + stack $E'T')E'T	
pop: T read: a go to [T, a] = FT' push T', F stack $E'T')E'T'F	) $

stak	input
pop: F go to [F, a] = a push a stack $E'T')E'T'a	
pop: a match w/ input a stack $E'T')E'T'	
pop: T' read:) go to [T',)] = λ stack $E'T')E'	$
pop: E' go to [E',)] = λ stack $E'T')	
pop:) match w/ input) stack $E'T'	
pop: T' read: $ go to [T', $] = λ stack $E'	Done w/ input string
pop: E' go to [E', $] = λ stack $	
pop: $ match w/ input $	Accepted

② $S \to aW$ $W \to =E$ → $S \to aw$ $w \to =E$

First (S) = { a }

First (w) = { = }

The rest is the same

states	a	+	-	*	/	(	)	$	=
S	aW								
W									=E
E	TQ					TQ			
Q		+TQ	-TQ				λ	λ	
T	FR					FR			
R		λ	λ	*FR	/FR		λ	λ	
F	a					(E)			