UNIT-4: Part - I

Error Detection and Recovery in PARSERS

M.B.Chandak

Building a parser

- Original grammar:

$$E \rightarrow E+E$$
$$E \rightarrow E*E$$
$$E \rightarrow (E)$$
$$E \rightarrow id$$

- This grammar is left-recursive, ambiguous and requires left-factoring. It needs to be modified before we build a predictive parser for it:

Remove ambiguity:

$$E \rightarrow E+T$$
$$T \rightarrow T*F$$
$$F \rightarrow (E)$$
$$F \rightarrow id$$

Remove left recursion:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E)$$
$$F \rightarrow id$$

Building a parser

- The grammar:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E)$$
$$F \rightarrow id$$

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}
FIRST(E') = {+, ε}
FIRST(T') = {*, ε}
FOLLOW(E) = FOLLOW(E') = {$,)}
FOLLOW(T) = FOLLOW(T') = {+, $,)}
FOLLOW(F) = {*, +, $,)}

Now, we can either build a table or design a recursive descend parser.

Parsing Table

NT	id	+	*	(	)	$
E	E→ TE’			E→TE’		
E’		E’→ +TE’			E’→€	E’→€
T	T→ FT’			T→FT’		
T’		T→ €	T’→*FT’		T→€	T→€
F	F→ id			F→(E)		

Error detection in LL(1) parsing

- An error is detected whenever an empty table slot is encountered.
- We would like our parser to be able to recover from an error and continue parsing.
- Phase-level recovery
  - We associate each empty slot with an error handling procedure.
- Panic mode recovery
  - Modify the stack and/or the input string to try and reach state from which we can continue.

Error recovery in LL(1) parsing

- Panic mode recovery
  - Idea:
    - Decide on a set of synchronizing tokens.
    - When an error is found and there's a nonterminal at the top of the stack, discard input tokens until a synchronizing token is found.
    - Synchronizing tokens are chosen so that the parser can recover quickly after one is found
      - e.g. a semicolon when parsing statements.
    - If there is a terminal at the top of the stack, we could try popping it to see whether we can continue.
      - Assume that the input string is actually missing that terminal.

Error recovery in LL(1) parsing

- Panic mode recovery
  - Possible synchronizing tokens for a nonterminal A
    - the tokens in FOLLOW(A)
      - When one is found, pop A of the stack and try to continue
    - the tokens in FIRST(A)
      - When one is found, match it and try to continue
    - tokens such as semicolons that terminate statements

Modified Parse Table

- Refer FOLLOW information for modifying parsing table.

NT	id	+	*	(	)	$
E	E→ TE’			E→TE’	SYNCH	SYNCH
E’		E’→ +TE’			E’→€	E’→€
T	T→ FT’	SYNCH		T→FT’	SYNCH	SYNCH
T’		T’→ €	T’→*FT’		T’→€	T’→€
F	F→ id	SYNCH	SYNCH	F→(E)	SYNCH	SYNCH

STRING PARSING

STACK	INPUT	REMARKS
$E	) id * + id $	Error, Check E→) ➔ Synch Skip)
$E	id * + id $	E with id ➔ E → T E'
$ E' T	id * + id $	T with id ➔ T → F T'
$ E' T' F	id * + id $	F with id ➔ F → id
$ E' T' id	id * + id $	id with id ➔ pop id
$ E' T'	* + id $	T' with * ➔ T' → * F T'
$ E' T' F *	* + id $	* With * ➔ pop *
$ E' T' F	+ id $	F with + ➔ Synch Skip F
$ E' T'	+ id $	T' with + ➔ T → €
$ E'	+ id $	E' with + ➔ E' → + T E'
Continue ..		

LR PARSERS

- In the parsing table, all the empty cells are considered as error cells.
- The error routines are designed by proper understanding of grammar and string generated by grammar.
- The methodology is called as: Phrase based error detection and recovery.
- The method provides the error messages to be displayed and also suggest necessary corrective actions.

The Closure Operation -- Example

E' → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → id

closure({E' → ▪ E}) =
{E' → ▪ E
kernel items
E → ▪ E+T
E → ▪ T
T → ▪ T*F
T → ▪ F
F → ▪ (E)
F → ▪ id }

Example

I ={ E' → .E,
E → .E+T,
E → .T,
T → .T*F,
T → .F,
F → .(E),
F → .id }

goto (I, E) = { E' → E., E → E.+T }

goto (I, T) = { E → T., T → T.*F }

goto (I, F) = {T → F. }

goto(I,id) = { F → id. }

goto (I, ( ) = { F → (.E), E → .E+T,
E → .T, T → .T*F, T → .F,
F → .(E), F → .id }

The Canonical LR(0) Collection -- Example

I_0: E' → .E .
E → .E+T

E → .T
T → .T*F

T → .F
F → .(E)
F → .id

I_1: goto(I_0, E)
E' → E.
E → E.+T

I_2: goto(I_0, T)
E → T.
T → T.*F

I_3: goto(I_0, F)
T → F.

I_4 : goto(I_0, ( )
F → (.E)

E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_5: goto(I_0, id)
F → id.

I_6: goto(I_1, +)
E → E+.T
T → .T*F
T → .F
F → .(E)
F → .id

I_7: goto(I_2, *)
T → T*.F

F → .(E)

F → .id

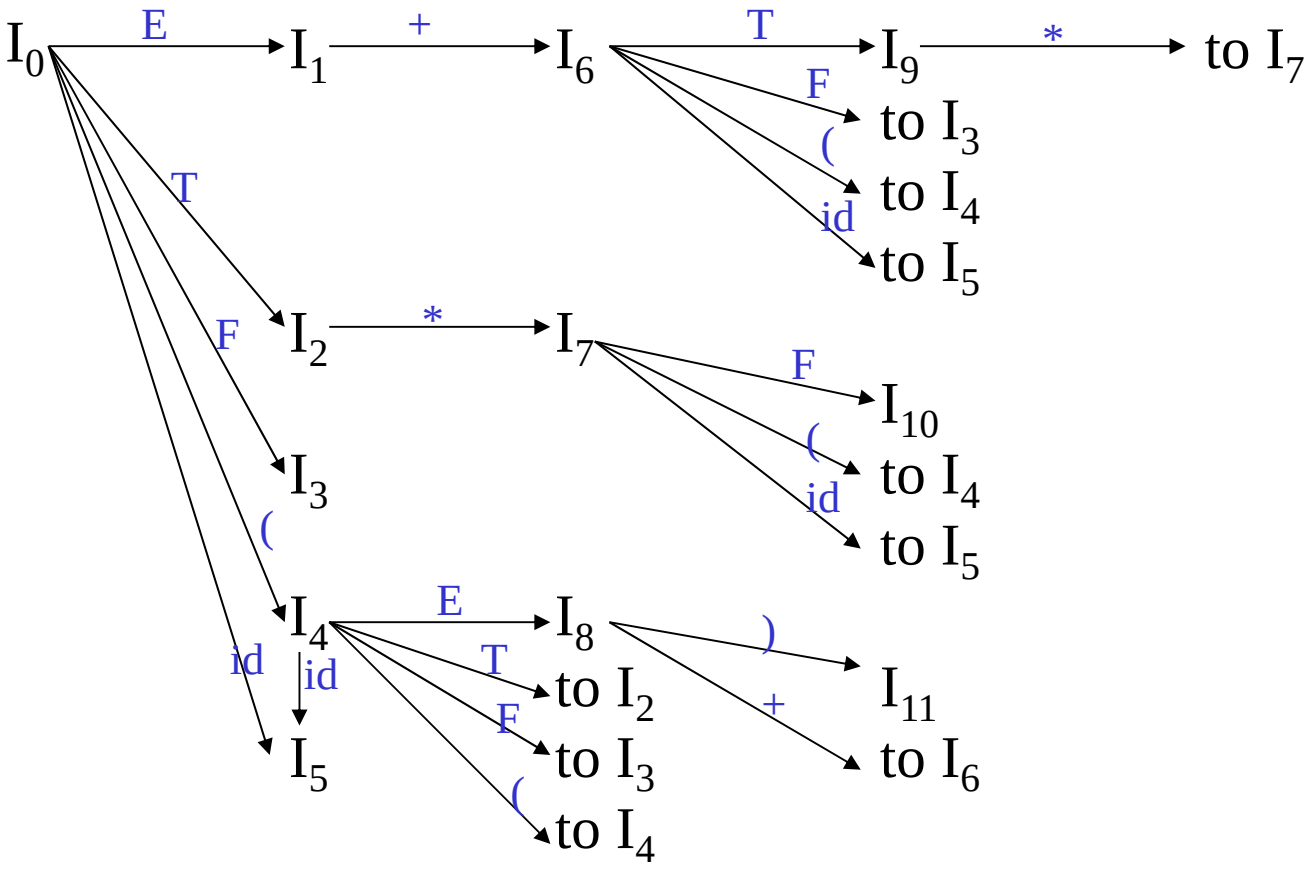
I_8: goto(I_4, E)
F → (E.)
E → E.+T

I_9:goto(I_6, T)
E → E+T.
T → T.*F

I_{10}: goto(I_7, F)
T → T*F.

I_{11} : goto(I_8,))
F → (E).

Transition Diagram (DFA) of Goto Function



Parsing Tables of Expression Grammar

	Action Table							Goto Table		
state	**id**	**+**	*****	**(**	**)**	**$**		**E**	**T**	**F**
0	s5			s4				1	2	3
1		s6				acc				
2		r2	s7		r2	r2				
3		r4	r4		r4	r4				
4	s5			s4				8	2	3
5		r6	r6		r6	r6				
6	s5			s4					9	3
7	s5			s4						10
8		s6			s11					
9		r1	s7		r1	r1				
10		r3	r3		r3	r3				
11		r5	r5		r5	r5				

Example-2

- Consider Grammar:

E → E+E | E*E | (E) | id

After Closure and Goto operation the parsing table generated will be:

State	id	+	*	(	)	$	E
I0	s3	e1	e1	s2	e2	e1	1
I1	e3	s4	s5	e3	e2	Acc	
I2	s3	e1	e1	s2	e2	e1	6
I3	r4	r4	r4	r4	r4	r4	
I4	s3	e1	e1	s2	e2	e1	7
I5	s3	e1	e1	s2	e2	e1	8
I6	e3	s4	s5	e3	s9	e4	
I7	r1	r1	s5	r1	r1	r1	
I8	r2	r2	r2	r2	r2	r2	
I9	r3	r3	r3	r3	r3	r3	

Example

- Originally only shift and reduce entries are added in the table.
- The error entries are designed based on grammar and string knowledge base.
- Description of Error Entries:
- Entry “e1”: For states 0,2,4 and 5

Expected operand found operator: Push an imaginary operand on stack [missing operand error]

- Entry “e2”: For states: 0, 1, 2, 4 and 5

Right parenthesis is found, without left parenthesis, remove it from input.

[unbalanced right parenthesis]

Example

- Entry “e3”: called for states 1 and 6

Expected operator and “id” or right parenthesis is found.

Push operator onto stack [missing operator error]

- Entry “e4”: Called from state 6

End of input is found, but expected is right parenthesis. “missing right parenthesis”

Stack Example

- For the same parser consider string: id +) $

Stack	Input	Remarks
0	id +) $	
0 id 3	+) $	
0 E 1	+) $	
0 E 1 + 4	) $	
	$	CALL E2 REMOVE)
0 E 1 + 4	$	MISSING OPERAND call E1 push id 3 on stack
0 E 1 + 4 id 3	$	
0 E 1 + 4 E 7	$	
0 E 1	$	