Example (Contd.)

String: “id + *$”

Stack	Input	Error message and action
0	id+*$	Shift
0id2	+*$	Reduce by E->id
0E1	+*$	Shift
0E1+3	*$	“id expected”, pushed id and 2 to stack
0E1+3id2	*$	Reduce by E->id
0E1+3E5	*$	Shift
0E1+3E5*4	$	“id expected”, pushed id and 2 to stack
0E1+3E5*4id2	$	Reduce by E->id
0E1+3E5*4E6	$	Reduce by E->E*E
0E1+3E5	$	Reduce by E->E+E
0E1	$	Accept

State	ACTION				GOTO
	id	+	*	$	E
0	S2	e1	e1	e1	1
1	e2	S3	S4	Acc	
2	e2	R3	R3	R3	
3	S2	e1	e1	e1	5
4	S2	e1	e1	e1	6
5	e2	R1	S4	R1	
6	e2	R2	R2	R2	

LALR Parser Generator - yacc

- yacc – Yet Another Compiler Compiler
- Automatically generates LALR parser for a grammar from its specification
- Input is divided into three sections

...definitions...	Consists of token declarations C code within %{ and %}
%%	
...rules...	Contains grammar rules
%%	
...subroutines...	Contains user subroutines

Example – Calculator to Add and Subtract Numbers

- Definition section

```
%token INTEGER
```

declares an INTEGER token

- Running yacc generates y.tab.c and y.tab.h files
- y.tab.h:

```
#ifndef YYSTYPE
#define YYSTYPE int
#endif
#define INTEGER 258
extern YYSTYPE yylval;
```

- Lex includes y.tab.h and utilizes definitions for token values
- To obtain tokens, yacc calls function yylex() that has a return type of int and returns the token value
- Lex variable yylval returns attributes associated with tokens

Lex Input File

```
%{
#include <stdio.h>
void yyerror(char *);
#include “y.tab.h”
%}
%%
        [0-9]+ {
                        yylval = atoi(yytext);
                        return INTEGER;
        }
        [- + \n]  return *yytext;
        [ \t]  ;                        /* skip whitespace*/
        .               Yyerror(“Invalid character”);
%%
int yywrap(void) {
        return 1;
}
```

yacc Input file

```
%{
        int yylex(void);
        void yyerror(char *);
%}
%token INTEGER
%%
program:
                program expr '\n'       {printf("%d\n", $2);}
                |
                ;
```

yacc input file (Contd.)

```
expr:
                    INTEGER      {$$ = $1;}
                    | expr '+' expr          {$$ = $1 + $3;}
                    | expr '-' expr          {$$ = $1 - $3;}
                    ;
%%
Void yyerror(char *s) {
        fprintf(stderr, "%s\n", s);
        return 0;
}
Int main(void) {
        yyparse();
        return 0;
}
```

- yacc can determine shift/reduce and reduce/reduce conflicts.
- shift/reduce resolved in favour of shift
- reduce/reduce conflict resolved in favour of first rule

Syntax Directed Translation

- At the end of parsing, we know if a program is grammatically correct
- Many other things can be done towards code generation by defining a set of semantic actions for various grammar rules
- This is known as Syntax Directed Translation
- A set of attributes associated with grammar symbols
- Actions may be written corresponding to production rules to manipulate these attributes
- Parse tree with attributes is called an annotated parse tree

Example – generate postfix expression

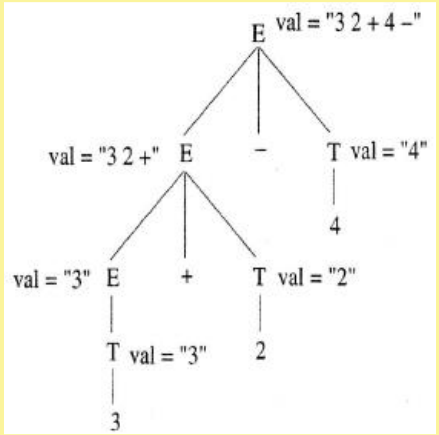
E -> E + T | E – T | T
T -> 0|1|2|…|9

Attribute val of E and T holds the string corresponding to the postfix expression

E -> E1 + T {E.val = E1.val || T.val || '+'}
E -> E1 – T {E.val = E1.val || T.val || '–'}
E -> T {E.val = T.val}
T -> 0|1|2|…|9 {T.val = number}

|| means string concatenation

Input string: "3 + 2 – 4"



Conclusion

- Seen various types of parsers for syntax analysis
- Error detection and recovery can be integrated with parsers
- Parse tree produced implicitly or explicitly by parsers
- Parse tree can be used in the code generation process

FREE ONLINE EDUCATION
swayam
शिक्षित भारत, उन्नत भारत

NPTEL ONLINE CERTIFICATION COURSES

Thank you!