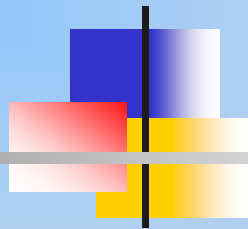
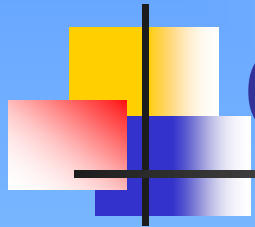


CSE302: Compiler Design



Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

March 27, 2007



Outline

- Recap
 - General/Canonical LR(1) parsing
 - Lookahead LR(1) / LALR(1) parsing
- Yacc
- Syntax-directed translation (Chapter 5)
- Summary and homework



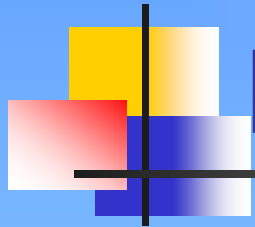
Outline

- Recap
 - General/Canonical LR(1) parsing
 - Lookahead LR(1) / LALR(1) parsing
- **Yacc**
- Syntax-directed translation (Chapter 5)
- Summary and homework



Yacc Introduction

- Yet Another Compiler-Compiler
 - Public domain versions: Bison
- Take a specification file (grammar) and produce an output file for the parser
 - Input: <filename>.y
 - {definitions}
 - %%
 - {productions/rules}
 - %%
 - {auxiliary routines}
 - Output: y.tab.c
 - LALR parser



Definition Section

- Information about tokens, data types
- Any code that must go directly into the output file
 - **#include** directives
- May be empty



Production Section

- Grammar rules with action codes
 - \rightarrow is replaced by :
 - A semicolon ends each production



Auxiliary Routine Section

- Procedure and function declarations that are needed to complete the parser
- May be empty



An Example: Grammar

- $\text{expr} \rightarrow \text{expr} + \text{term} \mid \text{expr} - \text{term} \mid \text{term}$
- $\text{term} \rightarrow \text{term} * \text{factor} \mid \text{factor}$
- $\text{factor} \rightarrow (\text{expr}) \mid \mathbf{\text{number}}$



An Example: Definition Section

- `%{`
`#include <stdio.h>`
`#include <ctype.h>`
`%}`
`%token NUMBER`
- `stdio.h`: `printf()`, `ungetc()`
- `ctype.h`: `isdigit()`



An Example : Production Section

- %%

line : expr	{printf("%d\n", \$1);}	;
expr : expr '+' term	{\$\$ = \$1 + \$3;}	
expr '-' term	{\$\$ = \$1 - \$3;}	
term	{\$\$ = \$1;}	;
term : term '*' factor	{\$\$ = \$1 * \$3;}	
factor	{\$\$ = \$1;}	;
factor : '(' expr ')'	{\$\$ = \$2;}	
NUMBER	{\$\$ = \$1;}	;

- \$\$: attribute value associated with the head
- \$i: attribute value of the *i*th grammar symbol
- Semantic action is performed when reduced



An Example : Auxiliary Routine Section

- %%

```
main () { return yyparse(); }
```

```
int yylex(void) {  
    int c;  
    while ( (c=getchar() ) == ' ');  
    if (isdigit(c)) {  
        ungetc(c,stdin);  
        scanf("%d",&yylval);  
        return(NUMBER);  
    }  
    if(c=='\n') return 0;  
    return(c);  
}
```

```
int yyerror(char *s) {fprintf(stderr, "%s\n", s); return 0;}
```



Compile the Example

- `yacc calc.y`
- `gcc -o calc y.tab.c -ly`



Observe The Parsing Table

- `yacc -v calc.y`
 - Prepares the file `y.output`
 - The kernels of the sets of items
 - A description of the parsing tables
 - A report on conflicts generated by ambiguities



Example Extended

- Evaluate a sequence of expressions
- /
- Floating-point values
 - #define YYSTYPE double
 - #ifndef YYSTYPE
 - typedef int YYSTYPE
 - #endif
 - YYSTYPE yylval
- A parse stack and a value stack



Integrate with Lexer

- `calc.l`

```
%{
%}
number    [0-9]+(\\.[0-9]+)?
%%
[ ]       { /* skip blank */ }
{number} {
    sscanf(yytext, "%lf", &yyval);
    return NUMBER;
}
\\n|.     {return yytext[0];}
```

- `flex calc.l`

- `yacc calc.y`

- `gcc y.tab.c -ly -ll`

Instructor: Dr. Liang Cheng

- `calc.y`

```
%{#include <stdio.h>
#define YYSTYPE double
%}
%token NUMBER
%%
lines : lines expr '\\n' {printf("%g\\n", $2);} | lines '\\n' | ;
expr  : expr '+' term   {$$ = $1 + $3;}
      | expr '-' term   {$$ = $1 - $3;}
      | term             {$$ = $1;}
term  : term '*' factor  {$$ = $1 * $3;}
      | term '/' factor  {$$ = $1 / $3;}
      | factor           {$$ = $1;}
factor : '(' expr ')' {$$ = $2;} | NUMBER {$$ = $1;}
%%
```

#include "lex.yy.c"

CSE302: Compiler Design

03/27/07



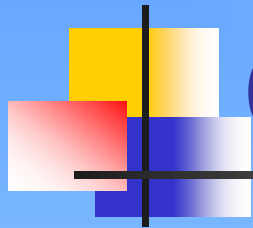
Another Example

- $\langle \text{program} \rangle \rightarrow \langle \text{program} \rangle \langle \text{func} \rangle \mid \langle \text{epsilon} \rangle$
- $\langle \text{func} \rangle \rightarrow \langle \text{type} \rangle \langle \text{ident} \rangle () \langle \text{block} \rangle$
- $\langle \text{block} \rangle \rightarrow \{ \langle \text{statement_list} \rangle \} \mid \langle \text{statement} \rangle$
- $\langle \text{statement_list} \rangle \rightarrow \langle \text{statement_list} \rangle \langle \text{statement} \rangle \mid \langle \text{epsilon} \rangle$
- $\langle \text{statement} \rangle \rightarrow \langle \text{expr} \rangle ; \mid ; \mid \langle \text{strexpr} \rangle ;$
- $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \mid \langle \text{expr} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle$
- $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{term} \rangle / \langle \text{factor} \rangle \mid \langle \text{factor} \rangle$
- $\langle \text{factor} \rangle \rightarrow \text{float} \mid \text{int}$
- $\langle \text{strexpr} \rangle \rightarrow \langle \text{strexpr} \rangle + \text{string} \mid \text{string}$
- $\langle \text{type} \rangle \rightarrow \text{void} \mid \text{string} \mid \text{float} \mid \text{int}$



Compile the Example

- flex part1.l
- yacc part1.y
- gcc -o part1 y.tab.c -ly -ll



Outline

- Recap
- Yacc
- **Syntax-directed translation (Chapter 5)**
- Summary and homework



What Can Be Done So Far?

- Lexing to detect lexical errors
 - Lexical analysis
- Parsing to detect syntax errors
 - Syntax analysis
- How about ...?
 - Semantic analysis
 - Type checking
 - Intermediate code generation
 - Applications of SDT



Syntax-Directed Techniques

- For a BNF grammar
 - Associate grammar symbol X (terminals and non-terminals) with a set of attributes $A(X)$: a_1, \dots, a_n
 - Type information for type checking/conversion
 - Strings for sequences of code in intermediate language
 - Number, table references, ...
 - Attach a **semantic rule** to each production
 - **Syntax-directed definition**
 - Add **program fragment(s)** to some production(s)
 - **Syntax-directed translation**
- The BNF grammar becomes an **attribute grammar**
 - Compute the values of the attributes associated with the symbols in the productions
 - Generate side effects: print results, modify symbol table, ...



Outline

- Recap
- Yacc
- Syntax-directed translation
- **Summary and homework**



Final Exam Reminder

- THURSDAY, MAY 03, 2007,
08:00-11:00AM