Syntax Analysis
Part III
Chapter 4

Slides adapted from :
© Robert van Engelen, Florida State University

Yacc and Bison

• *Yacc* (Yet Another Compiler Compiler)
  – Generates LALR(1) parsers
• *Bison*
  – Improved version of Yacc
Creating an LALR(1) Parser with Yacc/Bison

Bison Specification

- A `yacc specification` consists of three parts:
  - Bison declarations, and C declarations within `%{ %}`
  - Translation rules
  - User-defined auxiliary procedures
- The `translation rules` are productions with actions:
  
  ```
  production_1 { semantic action_1 }
  production_2 { semantic action_2 }
  ...
  production_n { semantic action_n }
  ```
Writing a Grammar in Yacc

• Productions in Yacc are of the form

  \[ \text{Nonterminal} : \text{tokens/nonterminals} \{ \text{action} \} \]

  | \text{tokens/nonterminals} \{ \text{action} \}

  \vdots

  ;

• Tokens that are single characters can be used directly within productions, e.g. ‘+’

• Named tokens must be declared first in the declaration part using

  \%token TokenName

Synthesized Attributes

• Semantic actions may refer to values of the synthesized attributes of terminals and nonterminals in a production:

  \[ X : Y_1 Y_2 Y_3 \ldots Y_n \{ \text{action} \} \]

  - \$$ refers to the value of the attribute of \( X \)
  - \$i refers to the value of the attribute of \( Y_i \)

• Example:

  \[
  \text{factor} : \left(\left(\text{expr}\right)\right) \{ \$$=$$2; \}
  \]

\[
\text{factor.val} = x
\]

\[
\text{expr.val} = x
\]

\[
\$$=$$2
\]
Example 1

```c
#include <ctype.h>
#include <stdio.h>

%token DIGIT

%%
line : expr \n
expr : expr \+ term
     | term

term : term \* factor
     | factor

factor : '(' expr ')' 
     | DIGIT

%%
int yylex()
{ int c = getchar();
  if (isdigit(c))
    { yylval = c - '0';
      return DIGIT;
    }
  return c;
}
```

Example of a very crude lexical analyzer invoked by the parser
Example 1 : Complete Code

```c
int yylex() {
    int c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return DIGIT;
    }
    return c;
}

int main() {
    if (yyparse() != 0 )
        fprintf(stderr, "Abnormal exit\n");
    return 0;
}

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

bison calculator.y
gcc calculator.tab.c
./a.out
5+8*(9+2)*3
269
^D

Tokens

- Bison represents literal tokens using the corresponding C character constant (ASCII code)
- Bison represents symbolic tokens as numbers higher than any possible character’s code, so they will not conflict with any literal tokens
- Can be overridden by `%token NUMBER 621`
Tokens

• When using symbolic tokens, run Bison with –d option to create a C header file with definitions

• If Bison is combined with Flex, add #include xxx.tab.h in lexer file, where xxx.y is the source file

Symbol Values

• Both tokens and nonterminals have an associated (semantic) value
  – For token, the value is stored in the C variable yylval
  – For nonterminals, the value is stored in the $$, $1, … pseudo-variables

• The associated semantic value is of type YYSTIPE, declared as int by default

• YYSTIPE can be redefined using the C instruction #define
Dealing with Ambiguous Grammars

- A description of parsing action conflicts can be obtained using the \(-v\) option, which produces an additional file \(y.output\)
- Reduce/reduce conflicts solved by using the conflicting production listed first
- Shift/reduce conflicts resolved in favor of shift

Dealing with Ambiguous Grammars

- We can also deal with ambiguous grammars by defining operator precedence levels and left/right associativity of the operators
- Example:

  \[
  \begin{align*}
  \%left & \text{'+' '}' '-'' '}' \\
  \%left & \text{'*'} '}/' '}' \\
  \%right & \text{MINUS}
  \end{align*}
  \]
Dealing with Ambiguous Grammars

• Productions are also given precedence and associativity, inherited from their rightmost nonterminal
• Example: item \( E \rightarrow E + E \) and lookahead + resolved with reduction (+ left associative)
• Example: item \( E \rightarrow E + E \) and lookahead * resolved with shift (* higher precedence)

Dealing with Ambiguous Grammars

• Can force different precedence by attaching to a production \( \#\text{prec} \) \langle terminal \rangle
• Symbol \langle terminal \rangle can be a placeholder: this terminal is never used by lexical analyzer, but indicates a precedence (see Example 2 later)
Combining Bison with Flex

Yacc or Bison specification
`yacc.y`

Lex or Flex specification
`lex.l`

and token definitions
`y.tab.h`

Terminal placeholder

Double type for nonterminal attributes and `yylval`

Example 2: Bison

```c
{%
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double
%
%token NUMBER
%left '+' '-'
%left '*' '/'
%right UMINUS
%
lines : lines expr '\n' { printf("%g\n", $2); }  
| lines '\n'  
| /* empty */
| ;
expr : expr '+' expr { $5 = $1 + $3; }
| expr '-' expr { $5 = $1 - $3; }
| expr '*' expr { $5 = $1 * $3; }
| expr '/' expr { $5 = $1 / $3; }
| '(' expr ')' { $5 = $2; }
| '-' expr %prec UMINUS { $5 = -$2; }
| NUMBER
| ;
%
```
Example 2: Bison (cont’d)

```c
int main()
{ if (yyparse() != 0)
    fprintf(stderr, "Abnormal exit\n");
    return 0;
} int yyerror(char *s)
{ fprintf(stderr, "Error: %s\n", s);
}
```

- Run the parser
- Invoked by parser to report parse errors

Example 2: Flex

```flex
%option noyywrap
%
#include "example2.tab.h"
extern double yylval;
%
number [0-9]+\.?|[0-9]*\.[0-9]+%
%
[ \t]  { /* skip blanks */ }
{number}  { sscanf(yytext, "%lf", &yylval);
             return NUMBER;
} 
\n|  { return yytext[0]; }
```

Generated by Bison, contains `#define NUMBER xxx`  
Defined in `example2.tab.c`
Error Recovery in Yacc

{%
...
%
%}

lines : lines expr \n' | lines \n' | /* empty */ |
| error \n' { printf("%g\n", $2; }
|
| /* empty */ |
| error \n' { yyerror("reenter last line: ");
| yyerrok; }

...;

Error production:
set error mode and
skip input until newline

Reset parser to normal mode

Symbol Values

- If several types are needed for grammar symbols, a union type must be defined.
- The %union declaration identifies all of the possible C types that a symbol value can have.
- The field declarations are copied verbatim into a C union declaration of the type YYSTYPE.
- In the absence of a %union declaration, Bison defines YYSTYPE to be int.
Symbol Values

• Associate the types declared in \texttt{union} with specific grammar symbols using
  – the \texttt{type} declaration for nonterminal
  – the \texttt{token} declaration for tokens

Example

\begin{verbatim}
%union {
    double dval;
    char *sval;
}
...
%token <dval> REAL    // token
%token <sval> STRING
%type  <dval> expr    // nonterminal
\end{verbatim}