Syntactic Analysis

Automatic Parser Generators: The UNIX YACC Tool

Portions of this lecture were adapted from Prof. Pedro Reis Santos's notes for the 2006 Compilers class lectured at IST/UTL

Outline

- YACC File Format
- Lexical Elements
- Priority and Associativity
- Semantic Actions
- Syntactic Analyzer Environment
- Error Handling
- Debugging
- LALR(1) Grammar Conflicts
- Lex and YACC
- Example

YACC Input File Format

- Similar to **lex** input format with 3 regions separated by "**%%**":
  - Declarations: **token**, **union**, **type**, **left**, **right**, **nonassoc**, **start** and base code declaration between "**%{**" and "**}%**"
  - Rules: each production separated by "**:**" (instead of →), alternative productions identified by "**|**" and semantic actions enclosed in braces "{" and "}"
  - Code: implementation of functions some of which declared in the first region

- Declarations
  - **%start**: declares the grammar's non-terminal goal symbol (start symbol)
  - When absent, it is assumed the goal symbol to be the first symbol in the rules section

Lexical Elements, Values and Types

- **%token**: declares the lexical elements (excluding the isolated characters) to be used in the grammar as terminal symbols (e.g., number or identifier). Several **%token** declarations may be present without any relative order

- **%union**: defines the data types to be saved in the parser stack/ These data types are associated with both terminal (generated by the lexical analyzer) or internally by the syntactic analyzer and associated with non-terminal symbols.

- **%type<** *x* **>**: associated the data type to the declared symbols. The "x" elements indicates a specific filed of the union used to represent the data types as in the **%union** directive.

- **%token<** *x* **>**: decleares the token and the corresponding type in a single declaration.

Priority and Associativity

- Possible to impose priority and associativity of rules by explicitly declaration:
  - %left: list of non-terminal symbols that use operators with left associativity
  - %right: list of non-terminal symbols that use operators with right associativity
  - %nonassoc: list of non-terminal symbols that use operators without any associativity
- Order of declaration defines relative priority
  - First declaration have the least priority
  - Last declaration the highest priority
- All symbols in the same declaration have the same relative priority

Semantic Actions

- Values of attributes are referred to by their position in the rule:
  - $$: designates the goal objective of the rule, left-hand side symbol
  - $1: designates the first symbol in the right-hand side of the rule. The remaining symbols are numbers in increasing order.
  - $<*type*>0: designates the first symbol associated with an inherited attribute. The type is needed as the symbol is not explicitly represented in the rule and may not even be uniquely identified. Other symbol in the stack are referred to by negative indices.
  - Semantic actions may return values that can be used in subsequent semantic actions, in the same rule and occupy a position in the rule. For example, X→Y$\underbrace{\{\$\$=3;\}}_{\$2}$Z{$$=$1*$2+$3}

Syntatic Analyzer Environment

- The syntactic analyzer is implemented by the **int yyparse();** function. It returns 0 if no syntactical errors have been found and returns 1 otherwise
- Whenever an error is found, the **int yyerror(char*)**; is invoked and the **int yynerrs;** variable is incremented
- The variables **int yystate;** and **int yychar;** define the internal states of the analyzer according to its LALR(1) table the the current lexical element code
- The **yacc** tool generates the **y.tab.c** file with the parser code and corresponding tables. The **-d** and **-v** switches allow the creation of the files **y.tab.h** (interoperability with **lex**) and **y.output** (LALR(1) table in readable format)
- The file dependency imposed by y.tab.h (with token identification) requires that the parser (using **yacc**) should be generated first and then the lexical analyzer (using **lex**). When linking (*link*), the **yacc** library should be indicated first by **-ly**, followed by the **lex** library, designated by **-ll** (or **-lfl** in case of **flex**)

YACC Error Handling

- The non-terminal symbol (keyword) **error**, allows the parser to handle erro conditions by skipping input terminal symbols until a suitable terminal from its **follow** set is found.
- The **YYERROR** macro may be used in semantic actions to trigger syntactical errors
- The function **yyerror(char*s);** is invoked whenever a syntactical error occurs, even if generated by the **YYERROR** condition
- The **yyerrok** macro can be used in semantic actions to allow the parser to continue after a syntactical error has occurred
- The **yyclearin** macro can be used in the semantic actions to force the analyzer to reread the *lookahead* lexical element for error recovery

Debugging

- **Static Analysis**: generated the syntactic analyzer using the -v switch (yacc -v gram.y) and inspect the y.output file that contains the analyzer states and transitions with the corresponding symbols

- **Dynamic analysis**: compile the syntactical analyzer with the YYDEBUG command line definition (gcc -c -DYYDEBUG y.tab.c, or set #define YYDEBUG 1 in the grammar) and set the yydebug variable to 1 (one) at the beginning of the program execution (in main). Note that YYDEBUG will only exist if it has been defined

LALR(1) Grammar Conflicts

- Production items indicate the internal parser state.
  - For **X → AB** there are 3 possible items: **X → .AB**, **X → A.B** and **X → AB.**, where the '**.**' indicates the parser position in processing this rule
- There is a conflict if in a given state the parser may be compelled to either shift or reduce as suggested by two of the items associated with that state
- YACC reports these conflicts as:
  - **Reduce/Reduce**: there are two distinct items with the '**.**' at the end
  - **Shift/Reduce**: there is an item with the '**.**' in the middle (suggesting a shift) and another item with the '**.**' at the end (suggesting a reduction)
- The *lookahead* token is used to eliminate conflicts in **LARL(1)** grammars

Reduce/Reduce Conflicts

- By default **YACC** performs a reduction using the first of the two offending rules in the input file and ignores the other

- Troubleshooting and Fixing:
  - Identify, in the **y.output** file the offending rules and determine the symbols generating the conflict
  - Try to factor the productions using common productions
  - Alternatively group the productions and resolve the ambiguity in the semantic actions (e.g., as in polymorphic operations)

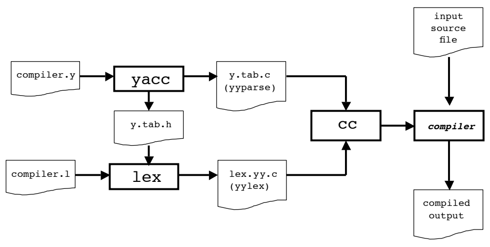
```
S : A | B       S : A | B | X
A : X | Y  ───►  A : Y
B : X | Z       B : Z
```

Shift/Reduce Conflicts

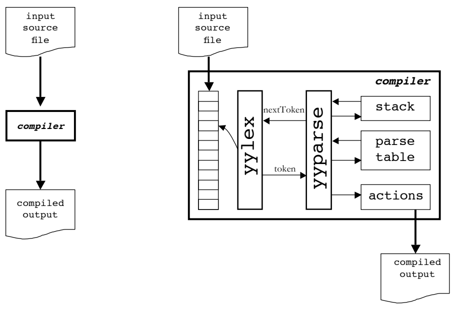
- By default **yacc** shifts over reducing

- Troubleshooting and Fixing:
  - Identify, in the **y.output** file the offending rules and determine the symbols generating the conflict
  - Enforce priorities (*e.g.,* as in expressions or if-else)
  - Enumerate alternatives (*e.g.,* as in function invocation or for construct)
  - Merge rules, using distributive rules, thus avoiding unnecessary reductions (*e.g,* as in declarations and instruction lists)

Lex and Yacc



Lex and Yacc



Example: calc.y

```
%{
#include <stdio.h>
%}

%union{
  int i;
}

%token INT UMINUS
%type<i> expr INT
%start calc

%%

calc: calc expr ';' { printf("%d\n", $2); }
    | expr      ';' { printf("%d\n", $1); }
    | error     ';' { printf("discarded input\n"); yyerrok;}
    ;

expr: expr '+' expr  {$$ = $1 + $3;}
    | expr '-' expr  {$$ = $1 – $3;}
    | expr '*' expr  {$$ = $1 * $3;}
    | expr '/' expr  {$$ = $1 / $3;}
    | expr '%' expr  {$$ = $1 % $3;}
    | '(' expr ')'   {$$ = $2;}
    | '-' expr {$$ = – $2;}
    | '+' expr %prec UMINUS {$$ = $2;}
    | INT {$$ = $1;}
    ;
```

```
%%

int yyerror(char *e){
  fprintf(stderr, "ERRO:%s %d\n;",e, yychar);
}

int main() {

  yyparse();
  return 0;
}
```

y.tab.h

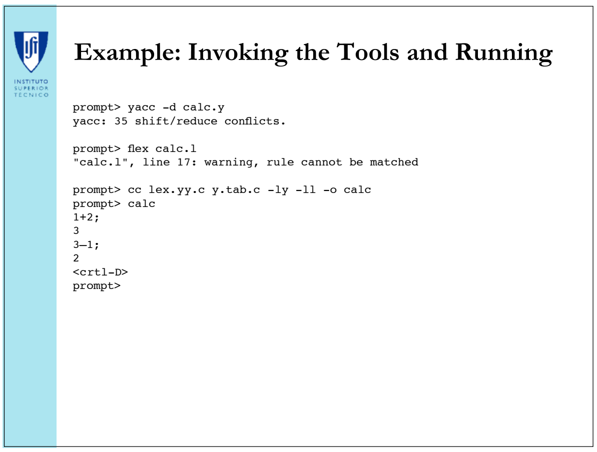
```
#define INT 257
#define UMINUS 258
typedef union{
  int i;
} YYSTYPE;
extern YYSTYPE yylval;
```

Example: calc.l

```
%{
#include "y.tab.h"
%}

%%

"-"     return '-';
"+"     return '+';
"*"     return '*';
"/"     return '/';
"%"     return '%';
";"     return ';';
"("     return '(';
")"     return ')';

[-+/*%;]  return *yytext;

[0–9]+    {yylval.i = atoi(yytext); return INT;}

[ \t\n]+  ;
.         yyerror("unrecognised character");
%%
```

y.tab.h

```
#define INT 257
#define UMINUS 258
typedef union{
  int i;
} YYSTYPE;
extern YYSTYPE yylval;
```



Example: Invoking the Tools and Running

```
prompt> yacc -d calc.y
yacc: 35 shift/reduce conflicts.

prompt> flex calc.l
"calc.l", line 17: warning, rule cannot be matched

prompt> cc lex.yy.c y.tab.c -ly -ll -o calc
prompt> calc
1+2;
3
3–1;
2
<crtl-D>
prompt>
```