1 Yacc

A Yacc specification describes a context free grammar (CFG), that can be used to generate a parser.

Elements of a CFG:

1. Terminals: tokens and literal characters,
2. Variables (nonterminals): syntactical elements,
3. Production rules, and
4. Start rule.

Lexical Analyzer scans the input stream and converts sequences of characters into tokens. Identifies tokens in input string (stream).

Format of a production rule:

```
symbol: definition
        {action}
        ;
```

Example: A → Bc is written in yacc as `a: b 'c';`

1.1 `yacc` File Format

definitions

%%

grammar rules (productions and associated actions)

%%

user routines (C functions)

1.1.1 definitions

```
%{
C code
%}

%definition(s)
```

1.1.2 rules

```
rule action
rule action
rule action
```

Declarations—define tokens and their characteristics

`%token`: declare names of tokens

`%prec`: assign precedence to a rule
`%left`: define left-associative operators
`%right`: define right-associative operators
`%nonassoc`:define operators that may not associate with themselves

`%type`: declare the type of variables
`%union`: declare multiple data types for semantic values
`%start`: declare the start symbol (default is the first variable in rules)

`%{`
C declarations directly copied to the resulting C program
`%}` (E.g., variables, types, macros)

Although right-recursive rules can be used in yacc, left-recursive rules are preferred, and, in general, generate more efficient parsers.

Invoking `yacc` on a file, e.g.,

```
yacc yaccFile.y
```

generates two output files:

`y.tab.c` and `y.tab.h`

An application can be built using:

```
cc -o app lex.yy.c y.tab.c -ll -ly
```

1.2 Conflicts

Pointer model: A pointer moves (right) on the RHS of a rule while input tokens and variables are processed.

```
%token A B C
%%
start: A B C /* after reading A: start: A B C */
```

When all elements on the right-hand side are processed (pointer reaches the end of a rule), the rule is *reduced*. The pointer then returns to the rule it was called.

Conflict: There is a conflict if a rule is reduced when there is more than one pointer.

`yytext` is an external variable that contains the matched string.

1.3 Simple Calculator

1.3.1 `calc1.l`

```
%{
/* calc1.l
 */

#include <stdio.h>

#include "y.tab.h"

extern int yylval;
%}

%%
```

```
[0-9]+      {
                yylval = atoi(yytext);
                return INTEGER;
            }

[-+\n]      return *yytext;

[ \t]       { ;  /* skip whitespace */ }

.           yyerror( "invalid character" );

%%

int yywrap( void )
{
    return 1;
}
```

1.3.2 calc1.y

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

%token INTEGER

%%

program:
    program expr '\n'   { printf("= %d\n", $2); }
    |
    ;

expr:
    INTEGER             { $$ = $1; }
    | expr '+' expr     { $$ = $1 + $3; }
    | expr '-' expr     { $$ = $1 - $3; }
    ;
```

```
%%

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

int main()
{
    yyparse();
    return 0;
}
```

$$ left hand side value
$n nth argument on right hand side

1.3.3 Building the Calculator

```
# buildCalc1.sh
#
#  Create simple calculator
#
   #  make y.tab.c and y.tab.h (-d option creates header)
yacc -d calc1.y
   #  make lex.yy.c
lex calc1.l
   #  compile and link C files
cc -o calc1 y.tab.c lex.yy.c -ly -ll

% ./buildCalc1.sh
yacc: 4 shift/reduce conflicts.
```

1.3.4 Running the Calculator

```
% ./calc1
1 + 2
= 3
```

```
2 *3
invalid character
syntax error
```

References

[Johnson79] Stephen C. Johnson, "Yacc: Yet Another Compiler-Compiler," *UNIX Programmer's Manual*, volume 2, Holt, Rinehart, and Winston, pages 353–387, 1979.

[LM&B90] Levine, Mason & Brown, *lex & yacc*, O'Reilly & Associates, 1990.

[S&FB85] Schreiner and Friedman, *Introduction to Compiler Construction with Unix*, Prentice-Hall, 1985.

[LnYP03] The Lex & Yacc Page, `http://dinosaur.compilertools.net/`