Helpful Notes on Yacc

Evan Austin

September 23, 2014

Yacc Specification Structure

```
{ declarations }

%%

{ rules }

%%

{ programs }
```

Common Yacc Values

- *yyparse* - The parser function produced when a Yacc specification is compiled.
- *yyerror* - The function called when a syntax error is detected. Default implementation prints "Syntax error."
- *yylval* - The name of the Yacc's value stack union type, as defined by the %union keyword.
- *yydebug* - Flag that, when set to 1 in the a literate piece of C in the declaration section, triggers debug output code.
  Note: the Yacc specification must be compiled with the t flag to enable debug code.

Declarations

Possible Members of the Declaration Section:

1. Literate C code to be copied over - delimited by %{ braces
2. Definition of Yacc's value stack union type: %union ...
3. Token definitions of the form: %token <union_id> name1 name2 ...
   Used to automatically generate the header file declaring tokens used by Lex and Yacc both.
4. Optional precedence declarations for tokens.
5. The optional definitions of non-terminal symbols:
   %type <union_id> name1 name2 ...
   Generally only used when needing to associate a union type to a non-terminal.
6. The starting non-terminal symbol: %start name

Translation Rules

Form: A : Body ;

Where A is the name of a non-terminal and Body is a (possibly empty) list of names of non-terminals and terminals both.

Semantic actions can also be embedded before/after any symbol in the Body.

Non-terminals with multiple rules can be combined into a single form using the pipe operator:

```
A : B ;
A : C ;
A : D ;
```

OR

```
A : B | C | D ;
```

Semantic Actions

Semantic actions are C code fragments enclosed in curly braces.

Also available are a list of pseudo variables:

- $$ - The return value of the action.
- $X - The value of the symbol at position X.

The default action for a rule is $$ = $1;.

Non-terminals with multiple rules can be combined into a single form using the pipe operator:

```
A : B;
A : C;
A : D;
```

OR

```
A : B | C | D;
```

Useful Notes on Grammars

Yacc deterministically creates a single LALR(1) parser.

When faced with an ambiguous grammar:

- shift /reduce conflicts - Yacc will always shift.
- reduce/reduce conflicts -Yacc will always reduce the first production.

Compiling with the v flag will create a $*$.output file with readable descriptions of the parsing tables and report any conflicts generated from an ambiguous grammar.

If things are not working as expected, check that you don't have any conflicts.