Compiladores

YACC

Ing. Adrian Ulises Mercado Martínez

Facultad de Ingeniería, UNAM

1 de Noviembre de 2014

Yacc

Qué es Yacc

- Yacc es un programa que genera analizadores sintácticos. LALR.
- Significa Yet Another Compiler-Compiler.
- Utiliza notación BNF simplificada.
- Fue escrito por Stephen C. Johnson en AT&T para Unix.
- Se ha reescrito para Lisp, EFL, ML, Ada, Java y otros.
- Puede trabajar con un analizador léxico generado por Lex

Estructura de un archivo yacc (*.y)

Estructura de un archivo yacc (*.y)

Declaraciones
%%
Gramatica
%%
Codigo de usuario

Sección Declaraciones

Declaraciones e importar o incluir

Se usa la directiva **%{ %}** para escribir el código que se va importar en el caso de java y las bibliotecas a incluir en el caso de C

java

```
%{
import ....;
%}
```

C

```
%{
#include ...
%}
```

Sección Declaraciones

Tokens

Un símbolo terminal o token se declarara haciendo uso de la instrucción:

```
%token <tipo> id  { id}
```

Ejemplos :

```
%token ID
%token <dval> NUM
```

Nota: se usa notación EBNF para describir a yacc
Estos tokens son los mismos que debe retornar el analizador léxico.

Sección Declaraciones

Tokens

También se puede declarar un terminal usando la definición de la precedencia y asociatividad en caso de ser operador.
La precedencia se toma en yacc de la siguiente manera, entre más abajo del documento aparezca el operador, éste tendra mayor precedencia. Los operadores que aparecen en la misma línea tienen la misma precedencia.
La asociatividad se define con las siguientes instrucciones

```
%left id { id}
%right id { id}
%nonassoc id { id}
```

Nota: en lugar de un id, se puede usar un carater, un caracter es cualquier símbolo encerrado en comillas simples, ejempos: '+', '-', '*', '/'

Sección Declaraciones

No terminales

En general no es necesario declarar los no terminales, pero si se necesita definir un tipo para un no terminal se declara de la siguiente manera

```
%type <tipo> id { id}
```

Símbolo inicial

En general no se necesita declarar el símbolo inicial en caso de requerirse se hace de la manera siguiente.

```
%start id
```

Sección Gramática

Gamática

La forma de expresar las reglas gramáticales es:

Encabezado : cuerpo_producciones ;

El Encabezado es un identificador de un no terminal
El cuerpo_producciones es una combinación de símbolos no terminales y tokens, junto con acciones semánticas, las reglas gramáticales para un mismo no terminal deben estar separadas por |.

Ejemplo

```
exp : exp '+' term { accion semantica}
  | epx '-' term { accion semantica 2};
```

Tipo de los Tokens y no terminales

A los tokens y a los no terminales se les pude asignar un tipo que esta relacionado según el caso.

Java

$\%token\ <dval>\ NUM$

$\%type\ <dval>\ exp$

Donde $< dval >$ esta asociado con la clase ParserVal que se describe a continuación.

Java

En java se asigna el tipo usando la clase ParserVal genearada autómaticamente por yacc.

```
public class ParserVal
{
   public int ival;
   public double dval;
   public String sval;
   public Object obj;
   public ParserVal(int val)
   {
      ival=val;
   }
   public ParserVal(double val)
   {
      dval=val;
   }
```

Java

Para manejar el retorno de los tokens en conjunto con el analizador léxico se usa una variable del tipo ParserVal llamada yylval de la siguiente manera:

```
yylval = new ParserVal(doubleval);
yylval = new ParserVal(integerVal);
u otros casos
yylval = new ParserVal( new OtroObjeto());
```

Sección Gramática

Producciones ε

Una producción ε en yacc se escribe dejando vacía la producción.

Ejemplo:

exp$\rightarrow$ **exp + exp** $\mid \varepsilon$

En yacc:

```
exp : exp '+' exp | ;
```

Tipo de Tokens en yacc para C

Tipo de Tokes y no terminales para yacc en C

Se define el tipo de manera similar que en yacc para java con la diferencia de que los tipos se deben definir usando la directiva %union. En esta directiva el usuario puede definir todas las variables del tipo que quiera.

```
%union{
double dval;
float fval;
}

Ejemplos:
%token <dval> NUM

%type <dval> exp
```

Tipo de Tokens en yacc para C

yylval para C

De la misma manera la interfaz con el analizador léxico usa la variable **yylval** que incluye los datos de la unión definida anteriormente

```
yylval.dval = atod(yytext);
yylval.fval = atof(yytext);
```

Sección Código de Usuario

Sección de Código de Usuario

En código de usuario se escriben las rutinas de análisis léxico o se invoca el analizador léxico generado por lex.
Así como el método o función principal que ejecutará todo el código.
Además de todas las rutinas auxiliares correspondientes al analizador semántico y al generador de código intermedio.
También se suele incluir la rutina que maneja los errores sintácticos.

Acciones

Acciones

Una acción en yacc se representa entre { } y según sea el caso es código en java o en C, en la acción semántica se representa cada símbolo gramatical del cuerpo de la producción con $n; donde n corresponde al número del símbolo comenzando con 1. El encabezado se representa con el símbolo $$

Ejemplo

$$exp \ : \ exp \ MAS \ exp \ \{\$\$ = \$1 + \$3\}$$

Acciones

Acciones

Se pueden insertar acciones no solo al final de las producciones, pero con acciones inscrustradas dentro del cuerpo, yacc realizara un cambio de un esquema con acciones dentro del cuerpo a uno posfijo insertando nuevos no terminales.

Ejemplo

$$exp : \{imprimir();\}\ exp\ MAS\ exp\{\ \$\$.dval = \$1.dval + \$3.dval;\}$$

Compilación

Compilación

Para C

Cualquier archivo que se escriba en yacc se debe guardar con la extensión **.y**

1. Para compilar se usa **yacc archivo.y**
2. Esto genera y.tab.c y y.tab.h
3. Compilar luego el archivo en lex con **lex archivo.l**
4. Esto genera lex.yy.c
5. Finalmente compilar con **gcc y.tab.c lex.yy.c -o nombre -lfl**

Compilación

Compilación

Para Java

Cualquier archivo que se escriba en yacc se debe guardar con la extensión **.y**

1. Para compilar se usa **byaccj -J archivo.y**
2. Esto genera Parser.java y ParserVal.java
3. Compilar luego el archivo en lex con **jflex archivo.flex**
4. Esto genera Yylex.java
5. Finalmente compilar con **javac** todos los archivos .java generados