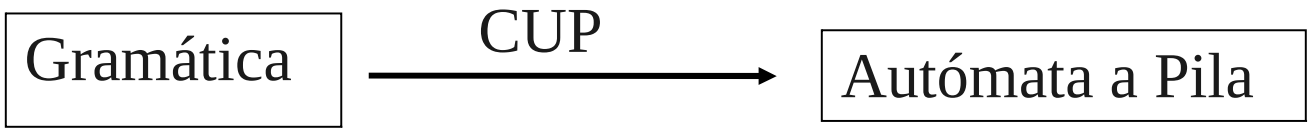
Tabla de Contenidos

CUP (Java Based Constructor of Useful Parsers)

Funcionamiento de CUP I

- Su función principal es: dada una gramática (G) libre de contexto, obtener el Autómata a Pila (AP) que reconoce al lenguaje L(G).



- Cup tendrá como entrada un fichero con extensión .cup y obtiene los siguientes ficheros:
  - Parser.java => Define a la clase que implementa el A.P. El árbol sintáctico se crea de abajo a arriba (método ascendente LALR).
  - Sym.java => Contiene una clase donde se define cada símbolo terminal encontrado en el fichero .cup como un entero.

CUP (Java Based Constructor of Useful Parsers)

Funcionamiento de CUP II

Funcionamiento del parser (Análisis sintáctico)

El parser va construyendo el árbol sintáctico conforme va recorriendo la secuencia de terminales. Si al finalizar la lectura de los terminales ha logrado llegar al símbolo inicial de la gramática, se concluye que la frase es correcta, en caso contrario, la frase no pertenece a L(G).

Relación entre Cup y Jflex

- Cada símbolo terminal de la gramática puede ser definido con una expresión regular. Por tanto, en Jflex se definen todos los terminales de la gramática. El parser no necesita leer carácter a carácter la entrada sino que hace una llamada al método yylex() para que este recorra la entrada y le devuelva el siguiente terminal encontrado.

CUP (Java Based Constructor of Useful Parsers)

Funcionamiento de CUP III

- La clase predefinida Symbol permite definir objetos con la información del terminal que acabamos de reconocer. Como mínimo esta información contiene el token encontrado (entero, identificador etc). Además, podemos insertar un objetos que amplien la información del token encontrado. Por ejemplo, la cadena concreta encontrada.

Ejemplo: Asignación

- Gramática : Σ_T = { id, num, igual, mas por }, Σ_N = {A,E}

  A -> id igual E

  E -> E mas E | E por E | id |num
- Cadena de entrada: a := 34 + b * c
- Secuencia de terminales (tokens) : id igual num mas num por num

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup

- ***fichero.cup***: nombre del fichero que contiene la especificación del analizador.

En una especificación se pueden diferenciar cinco partes:

1º especificaciones de “package” e “imports”

2º componentes de código de usuario

3º lista de símbolos de la gramática (terminales-no terminales)

4º declaraciones de precedencia

5º especificación de la gramática

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup II

1º Sección de sentencias "package" e "imports" (opcional):

Guardan la misma sintaxis y juegan el mismo papel que un programa típico de Java.

Una declaración package indica el nombre del paquete en el que se deben incluir las clases generadas por CUP.

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup III

2º Sección componentes de código de usuario (opcional):

Contiene:

- la declaración de las variables y rutinas utilizadas en el código de usuario embebido.

action code{: .... :}

- la declaración de métodos y variables que se incluyen dentro del código de la clase del analizador generado (p.e.: código para ajustar a unas ciertas necesidades, sobrescribir la rutinas de gestión de errores, ...).

parser code{: ... :}

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup IV

2º Sección componentes de código de usuario (opcional):

...Contiene:

- declaración de código que debe ejecutar el analizador antes de recoger el primer “token”. Por ejemplo, código para inicializar el escáner, tablas o cualquier otro tipo de estructuras de datos:

init with{: ... :}

- la especificación de cómo el analizador debería preguntar por el próximo “token” al escáner:

scan code{: ... :}

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup V

3º Sección lista de símbolos de la gramática (obligatoria):

- Declaración de símbolos terminales:

  terminal [clase] nombre1, nombre2,...,nombreN ;

  Ejemplos: terminal PUNTOYCOMA, SUMA;

  terminal Integer NUMERO;

- Declaración de símbolos no terminales:

  non terminal [clase] nombre1, nombre2,...,nombreN ;

  Ejemplos: non terminal expr_list, expr_part;

  non terminal Integer expr, factor, term;

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup VI

4º Sección declaración de precedencias (opcional):

- Es útil para el análisis de gramáticas ambiguas.
- Existen tres tipos de declaraciones de precedencia/asociatividad:

  precedence left terminal[, terminal...];

  precedence right terminal[, terminal...];

  precedence nonassoc terminal[, terminal...];
- El orden de precedencia va desde el final al principio, es decir:

  precedence left SUMA, RESTA;

  precedence left MULTIPLICACIÓN, DIVISIÓN;

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup VII

4º Sección declaración de precedencias (opcional):

- Si un terminal no está en la declaración tendrá el menor orden de precedencia.
- Las producciones sin terminales tendrán el menor orden de precedencia, y si presentan algún terminal será el correspondiente a este último.
- La asociatividad es asignada a cada terminal según la declaración realizada(izquierda, derecha y no asociatividad).
- Por defecto, suele ser de izquierda a derecha.
- Si un terminal es declarado “nonassoc” servirá para que cuando aparezcan dos ocurrencias del mismo terminal en la misma producción se genere un error.

CUP (Java Based Constructor of Useful Parsers)

Estructura archivo *.cup VIII

5º *Sección especificación de la gramática(obligatoria):*

- Contiene la gramática junto con las acciones a realizar cuando se reconozca una producción de la misma.
- Declaración del símbolo inicial de la gramática:
  - ***start with*** <simb_No_terminal>;
    - Si esta declaración no aparece, el símbolo inicial se toma como el símbolo de la primera producción definida.
  - **<No terminal> ::= <producción> [| <producción> ...];**
  - declaraciones de acciones: **{: ... :}** . Definida para cada regla.
  - utilización de la clase ***java_cup.runtime.Symbol*** y clase ***sym.java*** para integrar el analizador sintáctico con el analizador léxico (escáner).

CUP (Java Based Constructor of Useful Parsers)

Ejemplo: Expresiones aritméticas I

```
/* -------------------------Seccion codigo-usuario ---------------------- */
import java_cup.runtime.Symbol;

%%
/* ----------------- Seccion de opciones y declaraciones  ----------------- */

%cup

%%
/* ----------------------- Seccion de reglas lexicas ---------------------- */


";" { return new Symbol(sym.PUNTOYCOMA); }
"+" { return new Symbol(sym.MAS); }
"*" { return new Symbol(sym.POR); }
"(" { return new Symbol(sym.PAREN_I); }
")" { return new Symbol(sym.PAREN_D); }
[0-9]+ { return new Symbol(sym.NUMERO, new Integer(yytext())); }
[ \t\r\n\f] { /* ignora delimitadores */ }
. { System.err.println("Caracter Ilegal: "+yytext()); }
```

exprArit.lex

CUP (Java Based Constructor of Useful Parsers)

Ejemplo: Expresiones aritméticas II

```
/* ---------------------- Seccion de declaraciones package e imports--------------------*/
import java_cup.runtime.*;
import java.io.*;
/* ---------------------- Seccion componentes de codigo de usuario --------------------*/
parser code {:
     public static void main(String args[]) throws Exception {
          try{
          new parser(new Yylex(System.in)).parse();  // Inicializamos el scanner (Yylex) para que
          }                                          // la entrada sea por teclado
          catch ( Exception e) {
               System.out.println(" Análisis INCORRECTO !!");
               System.exit(1);}
          System.out.println("Análisis Correcto ");
     }
:}
/* ------------ Declaracion de la lista de simbolos de la gramatica ----------- */
terminal PUNTOYCOMA, MAS, POR, PAREN_I, PAREN_D;
terminal Integer NUMERO;
non terminal lista, expr_p;
non terminal Integer expr;

/* ------------- Declaracion de precedencias ----------- */
precedence left MAS;
precedence left POR;
```

exprArit.cup

CUP (Java Based Constructor of Useful Parsers)

Ejemplo: Expresiones aritméticas III

```
/* -------------- Declaricon de la gramatica  ----------- */

lista ::= lista expr_p | expr_p;
expr_p ::= expr PUNTOYCOMA;
expr      ::= NUMERO
            | expr MAS expr
            | expr POR expr
            | PAREN_I expr PAREN_D ;
```

Para que la entrada al analizador sea a través de un fichero en vez de a través de teclado, la definición de “parser code” debería ser la siguiente:

```
parser code {:

     public static void main(String args[]) throws Exception
{
     FileInputStream fichero=new FileInputStream(args[0]);
     DataInputStream entrada =new DataInputStream(fichero);
     try{
     new parser(new Yylex(entrada)).parse();
     }
     catch ( Exception e) {
          System.out.println(" Análisis INCORRECTO !!");
          System.exit(1);}
     System.out.println("Análisis Correcto ");
     }
:}
```

expArit_fich.cup

CUP (Java Based Constructor of Useful Parsers)

Ejemplo: Expresiones aritméticas IV

- Clase "sym" generada al ejecutar la herramienta CUP...

```
/** CUP generated class containing symbol constants. */
public class sym {
  /* terminals */
  public static final int NUMERO = 7;
  public static final int PUNTOYCOMA = 2;
  public static final int PAREN_D = 6;
  public static final int MAS = 3;
  public static final int error = 1;
  public static final int PAREN_I = 5;
  public static final int POR = 4;
  public static final int EOF = 0;
}
```

sym.java

CUP (Java Based Constructor of Useful Parsers)

Instalación

- Disponible para los sistemas operativos Windows y Linux.
- En Windows descomprimir el archivo de instalación.
- Java previamente instalado.
- Incluir el directorio que contiene a java_cup en la variable CLASSPATH
- Compilar java_cup de la siguiente forma (opcional):
  - javac java_cup/*.java java_cup/runtime/*.java

CUP (Java Based Constructor of Useful Parsers)

Instalación

- Instalación en Ubuntu:

Explicación en http://shakaran.es/blog/2009/02/instalar-jflex-y-cup-en-ubuntu/

- Instalación de Jflex y Cup
  - sudo apt-get install jflex cup (o usando synaptic, buscar ”jflex” “cup” y aplicar)
- Poner en el Classpath los ficheros jar:
- echo "export CLASSPATH=$CLASSPATH:/usr/share/java/cup.jar:/usr/share/java/JFlex.jar" | tee -a ~/.bashrc
- O bien, incluirlo solo para la sesión actual:
  - export CLASSPATH=$CLASSPATH:/usr/share/java/cup.jar:/usr/share/java/JFlex.jar

En general , en Linux:

Incluir los ficheros jar en :/usr/share/java

Añadir al classpath la ruta

CUP (Java Based Constructor of Useful Parsers)

Uso

- Para crear y usar el analizador sintáctico se deberá:

  1º Crear los ficheros *.flex y . cup, utilizando las clases: java_cup.runtime.Symbol y clase Sym.

  2º Ejecutar Jflex (***Jflex nombre.flex***) generando Yylex.java (o nombre del la clase definida por %class)

  3º Ejecutar cup con el archivo .cup (***cup nombre.cup en Ubuntu***) o

  (**java java_cup.Main <nombre.cup>** en Windows y Fedora8)

  4º Compilar los archivos *.java. (***javac Yylex.java sym.java parser.java***)

  4º Usar el parser. (***java parser <fichero o entrada teclado>***)

CUP (Java Based Constructor of Useful Parsers)

Uso: Ejemplo

- Pruebas... : **java parser** [<*nombre_archivo_pruebas*>]
  - Recordar: Las expresiones aritméticas deben acabar en punto y coma...
  - Ejemplo de salidas correctas:
    - 1+2+3;
    - 2*3+(90+4);
    - (3+5+6)*4;
  - Ejemplo de salidas no correctas
    - 1+2+3
    - 1+++2;
    - 3+(3+4;

CUP (Java Based Constructor of Useful Parsers)

Ejercicios propuestos

- Ampliar el ejemplo de las expresiones aritméticas, añadiendo la definición de los operadores división y resta. Añadir también el terminal identificador como parte de una expresión.
- Modificar el ejercicio anterior para que el lenguaje contenga a una secuencia de instrucciones de asignación.
- Practicar, en general, con las gramáticas vistas en clase