Práctica 5 - Analizador sintáctico ascendente LALR(1) con CUP

Procesadores de Lenguajes - 4º curso - Grado en Ingeniería Informática

1 Objetivos

- Elaborar una gramática para expresiones numéricas con operandos reales.
- Utilizar CUP[1] y JFLEX para generar un analizador sintáctico ascendente LALR(1) para dicha gramática.

2 Especificaciones del lenguaje

- Operandos: Números reales. La parte entera del número siempre ha de tener al menos un dígito y se admite notación científica.
  Ejemplos de números reales admitidos: *1*, *2.3*, *0.450*, *6e10*, *3.1e−11*, *0.12E+20*
- Operaciones soportadas: suma, resta, producto, división, potenciación y raíz cuadrada. Se admiten los operadores unarios $+$ y $-$.
- Las expresiones utilizan notación infija, respetando la precedencia de las operaciones.
- Se acepta la utilización de paréntesis en múltiples niveles.
- Una expresión finaliza con el carácter punto y coma.

  Ejemplos de expresiones correctas:

  −3;
  +5∗−0.51;
  −(2+3)
  2.5∗(3.1e10−2e−3+1.0)∗2.123;
  2^2^0.3;
  2.01E−1^−2.0;
  sqrt*(5e+3∗2);*

[1] http://www2.cs.tum.edu/projects/cup/

2.1 Especificaciones léxicas

El análisis léxico se encargará de procesar archivos con los siguientes tokens válidos:

FLOAT Número real.

PLUS Signo '+'

MINUS Signo '−'

TIMES Signo '∗'

DIV Signo '/'

POW Signo '^' (potenciación)

SQRT Operación raíz cuadrada (*sqrt*)

LEFT_BRACKET Paréntesis izquierdo ('(')

RIGHT_BRACKET Paréntesis derecho (')')

SEMICOLON Terminador de expresión (';')

El analizador léxico permitirá el uso de mayúsculas y minúsculas, indistintamente, y deberá encargarse de eliminar todos los espacios en blanco del archivo de entrada así como de detectar los errores de tipo léxico.

2.2 Especificación sintáctica

A continuación se propone una gramática para el lenguaje descrito:

$S \rightarrow SE; \mid \epsilon$
$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E\text{^}E \mid \mathbf{sqrt}(E) \mid (E) \mid \mathbf{ufloat}$

3 Instrucciones

1. Construir un analizador léxico que genere una clase Java adecuada para ser utilizada por un analizador sintáctico generado con CUP.

   - La acción para un token válido consiste en devolver un objeto de la clase *Symbol* (*java_cup.runtime.Symbol*) con información del token y la posición en que aparece dentro del archivo. Se pueden utilizar los constructores del ejemplo siguiente para símbolos con y sin atributo:

```
Symbol (sym.PLUS, n_fila , n_columna)
Symbol (sym.FLOAT, n_fila , n_columna , new Float(lexema));
```

- Cuando se detecte un error léxico se mostrará un mensaje con el siguiente formato:

  ```
  [Lex] Error léxico en línea ##, columna ##
  ```

- Los lexemas que den lugar a errores léxicos no se enviarán al analizador sintáctico, por lo que no tendrán efecto en el resultado final.

  Notas: Para que el analizador léxico se adapte a los requisitos de un analizador sintáctico generado con CUP, el archivo JFlex ha de incluir la opción **%cup**.

2. Construir una analizador sintáctico LALR(1) basado en CUP tomando como referencia la gramática propuesta y adaptándola, si fuese necesario, para que cumpla con las especificaciones del lenguaje.

   - El analizador sintáctico mostrará por pantalla el valor numérico resultado de las operaciones en las que no existan errores de sintaxis.
   - Recuperación ante errores de sintaxis. Cuando el analizador sintáctico encuentre un token que no se adecúe a la gramática, mostrará un mensaje con el formato que se indica a continuación e intentará recuperarse tras el siguiente carácter punto y coma.

     ```
     [Parser] Error de sintaxis: 'tipo_token'  en línea ##, columna ##
              Tipos de tokens esperados: [lista_tokens_esperados]
     ```

     Para ello, se puede redefinir la función *syntax_error* que ejecuta el analizador cuando detecta un error sintáctico:

     ```
     import java.util.List;
     import java.util.LinkedList;

     public void syntax_error(Symbol cur_token){
       String msg="[Parser] Error de sintaxis: '"+    \
             symbl_name_from_id(cur_token.sym)+"'  \
             en línea "+cur_token.left+", columna "+cur_token.right+"\n";
       List<Integer> list_of_ids = expected_token_ids();
       LinkedList<String> list = new LinkedList<String>();
       for (Integer token_id : list_of_ids)
                 list.add(symbl_name_from_id(token_id));
       msg += "\tTipos de tokens esperados: " + list +"\n";
       System.out.println(msg);
       }
     ```

3. Ejemplo de funcionamiento:
   Contenido del archivo de entrada:

```
2^2^3;
5+)3.1;
4*3E10;
```

   Salida por pantalla:

```
= 256
[Parser] Error de sintaxis: 'RIGHTBRACKET' en línea 2, columna 3
         Tipos de tokens esperados: [PLUS, MINUS, SQRT, LEFTBRACKET, FLOAT]
= 1.2E11
```

Se entregarán, al menos, los siguientes archivos:

1. Especificación léxica para JFlex (.jflex)
2. Especificación sintáctica para Cup (.cup)
3. Archivos generados por JFlex y Cup (.java)
4. Un archivo de entrada