Teoría de la Computación
Proyecto

Mayo 4 de 2017 - Prof: Rodrigo López

Introducción

Se trata de ejercitar el uso de la técnica del Descenso Recursivo para escribir un analizador sintáctico de expresiones regulares.

Usted debe diseñar una gramática tipo 2 o una BNF y a partir de ella debe programar, en Java, el analizador sintáctico. Es importante tener en cuenta que parte crucial de la evaluación consiste en verificar que, efectivamente, el analizador programado es un reflejo (según el método del descenso recursivo) de la respectiva gramática o BNF. En otras palabras, no se trata simplemente de hacer un analizador sintáctico de expresiones regulares sino que tiene que hacerse usando el método visto en clase.

El vocabulario

El vocabulario utilizado es el conjunto de los caracteres alfabéticos (en minúscula) y los dígitos decimales junto con los caracteres `'+'` y `'*'` que representan los operadores de Unión y Clausura de Kleene respectivamente. Finalmente, la cadena vacía (λ) se representa mediante el caracter `'\'`. El siguiente es un ejemplo de expresión regular válida sobre el vocabulario anterior y su equivalente en la notación del curso

$$\texttt{ab+c*(\+a)} \quad \sim \quad ab \cup c^*(\lambda \cup a)$$

Como es habitual en los lenguajes de programación, los blancos son separadores no significativos; es decir, la siguiente expresión es equivalente a la anterior:

$$\texttt{ab + c*(\ + a)} \quad \sim \quad ab \cup c^*(\lambda \cup a)$$

El analizador léxico

Para que la programación del parser sea lo más parecida a lo visto en clase usted dispondrá del binario del analizador léxico (clase `tcom.ui.RegLexer`) cuyo servicio clave es:

- `Token nextToken()`: Produce un valor de tipo `org.antlr.runtime.Token` correspondiente al siguiente token en la entrada.

Por otro lado, la clase `Token` ofrece los servicios:

- `String getText()`: Cadena de caracteres correspondiente al token.
- `int getLine()`: Número de línea en donde se encuentra el token.
- `int getCharPositionInLine()`: Número de columna (dentro de la línea) en donde se encuentra el token.

- `int getType()`: Tipo del token, codificado mediante un entero. Los códigos definidos para los tokens (por el analizador léxico) son las constantes que aparecen en la siguiente tabla:

Constante	Significado
EOF	Fin de cadena
SYMBOL	Símbolo alfabético o numérico
LAMBDA	'\'
OR	'+'
STAR	'*'
LPAR	Paréntesis izquierdo
RPAR	Paréntesis derecho

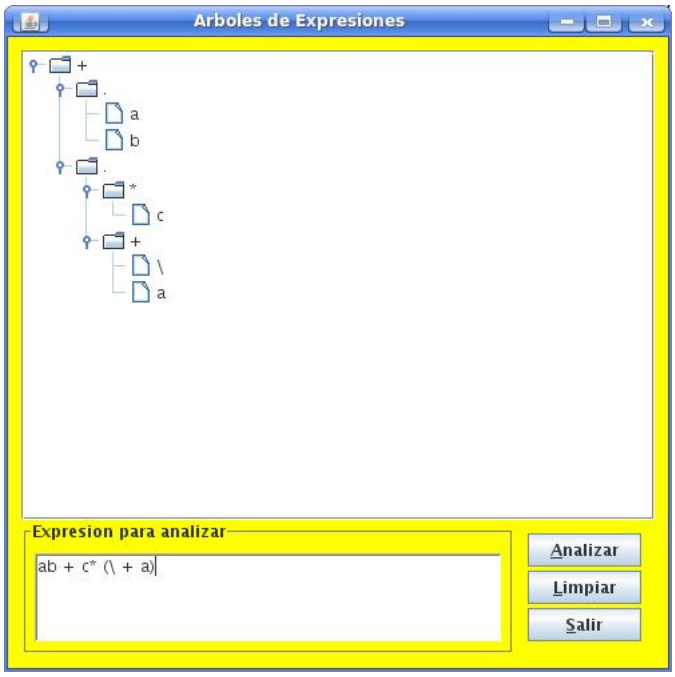
Suponiendo que la variable `token` es de tipo `Token` estas constantes se usan para efectuar verificaciones del estilo ...

```
if(token.getType()==SYMBOL) {
  ...
}

while (LPAR == token.getType()) {
....
}
```

La interfaz de usuario

La interfaz de usuario de su programa está proporcionada por la clase `tcom.ui.VisorExpr` que muestra, en forma de árbol, el resultado del análisis sintáctico de una expresión regular escrita en la zona de texto inferior, como se muestra en la figura:



Note que el operador de concatenación, que no aparece explícitamente en la expresión, se representa con un punto en el nodo correspondiente del árbol. Para poder usar este componente de visualización, es necesario instanciarlo y luego proporcionarle un analizador sintáctico. Los constructores/métodos para esta labor son:

- `public VisorExpr()`: Constructor del componente gráfico.
- `setParserTree(ParserTreeI parser)`: Asocia un parser con el componente.

El analizador sintáctico

El analizador sintáctico o *parser* debe implementar la interfaz `tcom.ui.ParserTreeI` que solamente contiene un método:

- `public AstI analizar(String src) throws VisorException`: Analiza la cadena `src` y retorna el árbol de parsing correspondiente.

Se requiere además que el árbol construído por el parser satisfaga la interfaz `tcom.ui.AstI`, la cual contiene los métodos:

- `public int getChildCount()`: Retorna el número de hijos de un nodo. Una hoja del árbol tiene 0 hijos.
- `public AstI getChild(int n)`: Retorna el n-ésimo hijo de este nodo. El primer hijo corresponde al índice 0.
- `public String getName()`: Retorna el nombre de este nodo.

Importante: Todas las clases e interfaces descritas hasta ahora se encuentran en los archivos `antlr-runtime-3.1.3.jar` y `tcomui.jar` que pueden bajarse de Moodle y deben usarse tanto para compilar como para ejecutar su programa.

El paquete `tcom.expreg`

Para facilidad en la corrección del proyecto, todo lo que usted programe debe hacer parte del paquete `tcom.expreg`. Si esta condición no se cumple, su proyecto no será corregido!! y la calificación será 0. Dos observaciones importantes:

- El método `main` invocado para iniciar el programa deberá encontrarse dentro de la clase `RegExpParser` del mencionado paquete. Lo natural es que esta clase instancie al visor y le proporcione el analizador sintáctico.
- En la implementación del método `analizar` de la interfaz `ParserTreeI` se debe seguir el siguiente protocolo que permite conectarse con el analizador léxico y reaccionar ante los errores.

```
RegLexer lexer; // Atributo para manejar el analizador léxico
Token token; // Token vigente en el analisis.
    ......

public AstI analizar(String srcexpr) throws VisorException{
   AstI tree = null;
   try {

      //Instanciar el analizador léxico y conectarlo con una cadena de caracteres.
      ANTLRInputStream in =
         new ANTLRInputStream(new ByteArrayInputStream(srcexpr.getBytes()));
      lexer = new RegLexer(in);

       //Iniciar el proceso de parsing
       tree = parse();

   }catch(IllegalArgumentException e){
      // Caracter inválido en la entrada
      throw new VisorException(" ... mensaje apropiado ...");
   } catch (UnsupportedEncodingException e) {
      //Problemas con la codificación de caracteres
     throw new VisorException(" ... mensaje apropiado ...");
   } catch (IOException e) {
      //Error de I/O inesperado
     throw new VisorException(" ... mensaje apropiado ...");

   }catch (RegParsingException e) {
      // Error detectado por su parser
      throw new VisorException(" ... mensaje apropiado ...");
   } catch (Exception e) {
      //Otros errores
      throw new VisorException(" ... mensaje apropiado ...");
   }

   return tree;
}
```

De todas las excepciones atrapadas la única que es responsabilidad suya es `RegParsingException`. Las demás son producidas por el entorno usado por su programa. En cualquier caso, es deseable producir mensajes de error apropiados.

Para que no pierdan tiempo

Para evitarles perder tiempo, la clase `RegExpParser` puede comenzar así:

```
package tcom.expreg;

import static tcom.ui.RegLexer.LAMBDA;
import static tcom.ui.RegLexer.LPAR;
import static tcom.ui.RegLexer.OR;
import static tcom.ui.RegLexer.RPAR;
import static tcom.ui.RegLexer.STAR;
import static tcom.ui.RegLexer.SYMBOL;
import static tcom.ui.RegLexer.EOF;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;

import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.Token;

import tcom.ui.AstI;
import tcom.ui.ParserTreeI;
import tcom.ui.RegLexer;
import tcom.ui.VisorException;
import tcom.ui.VisorExpr;
```

y el método `main` debe ser:

```
public static void main(String[] args) {
   VisorExpr visor = new VisorExpr();
   visor.setParserTree(new RegExpParser());
   visor.setVisible(true);
}
```

Entregables

- Un documento en el que aparezca la gramática o BNF utilizada para construir el parser, junto con el algoritmo de parsing, usando las convenciones del curso. Este documento puede ser tipo `.doc[x]` o `.pdf`.
- Todos los fuentes java implementados por usted dentro del paquete `tcom.expreg`. Cada uno de estos fuentes debe comenzar con un comentario con el nombre completo de quienes programaron. Además, los métodos que correspondan a los no-terminales de la gramática (o de la BNF) deben incluir como comentario la producción (regla) de donde provienen.

Todo lo anterior debe empacarse dentro de un archivo `.zip` cuyo nombre debe ser su apellido, sin tildes ni eñes y no debe contener subdirectorios. Por ejemplo, si quien programa es Rodrigo López, el archivo correspondiente debería llamarse `lopez.zip` (note que se omite la tilde). No se corrigen tareas en formatos diferentes a `.zip`.

Se puede trabajar en grupos de 2 personas, máximo.

Fecha de Entrega

21 de mayo de 2017