Ejercicios Laboratorio 2

Lenguajes de programación y procesadores de lenguaje

24 de Octubre de 2024

En el Campus Virtual de la asignatura en el apartado de "Laboratorio 2: Análisis Sintáctico" encontraréis un enlace para obtener CUP como archivo `cup.jar` y otro para obtener el código Java para generar el analizador sintáctico `AnalizadorSintacticoCUP.zip`. También encontraréis un ejemplo de especificación de la gramática usando CUP en `Tiny.cup`. Como ejemplo de entrada para el procesamiento final vamos a usar los mismos `input.txt` y `input1.txt` de la anterior sesión. Descargad todos estos archivos en una carpeta y descomprimid el fichero `AnalizadorSintacticoCUP.zip`.

Para más información o versiones más recientes de CUP visitad `http://www2.cs.tum.edu/projects/cup/`.

Podéis instalar CUP en vuestra cuenta usando el archivo `.jar` o lo podéis usar directamente en la línea de comandos. También podéis utilizar el entorno para desarrollo de código Java que prefiráis. Deberéis definir los caminos necesarios en el `CLASSPATH` o usar `-cp` al llamar a Java.

Ejercicio 1: Primer ejemplo con CUP.

En primer lugar, vamos a mirar el archivo `Tiny.cup`. En él se define la gramática incontextual de nuestro lenguaje. Podéis ver tanto la lista de terminales y no terminales como las reglas gramaticales del lenguaje. Para procesar el archivo `Tiny.cup` que contiene las definiciones para generar un analizador sintáctico con CUP, tenemos que ejecutar:

```
$ java -cp cup.jar java_cup.Main -parser AnalizadorSintacticoTiny
      -symbols ClaseLexica -nopositions Tiny.cup
```

Esto generará los archivos `AnalizadorSintacticoTiny.java` y `ClaseLexica.java` cuyo contenido debería coincidir con los archivos con el mismo nombre que ya existían en la carpeta `asint` de la carpeta `AnalizadorSintacticoCUP`. La carpeta `alex` contiene el analizador léxico que se trata como vimos en la

anterior sesión. Desde la carpeta `AnalizadorSintacticoCUP`, compilamos todo el código Java:

```
$ javac -cp "../cup.jar" */*.java
```

y ya podemos usar el analizador generado. Vamos a probarlo sobre el ejemplo siguiente que está en el fichero `input.txt` de la sesión anterior:

```
evalua

-456.12 * vacas + 1.34 * (ovejas + cabras)

donde
vacas = 567,
ovejas = 456,
cabras = 10
```

Para hacerlo tenéis que ejecutar (desde la carpeta `AnalizadorSintacticoCUP`):

```
$ java -cp ".:../cup.jar" asint.Main ../input.txt
```

No mostrará nada porque la entrada forma parte del lenguaje. Modificad la entrada introduciendo algún error sintáctico y volved a ejecutarlo.

Ejercicio 2: Lenguaje de Listas.

Vamos a continuar con el lenguaje de listas que comenzamos a desarrollar en la sesión anterior. Ahora nuestro objetivo es hacer un analizador sintáctico para el lenguaje `NumListLang`. El siguiente ejemplo muestra el lenguaje (en la entrada podrá haber comentarios que van después de `//`):

```
L = [] // L contiene la lista vacía
L2 = [1,2,3] // L2 contiene una lista con 3 números
L3 = L#L2 // L3 es la concatenación de L y L2, es decir L2
m4 = [[[1,2],3],4] // m4 es una lista heterogénea
L5 = lreduce + m4 // L5 tiene un único elemento: la suma de los números de m4
L6 = lmap - 1 m4 // L6 se obtiene restando 1 a los números de m4
print L5 // Se imprime [10]
print L6 // Se imprime [[[0,1],2],3]
L7 = lfilter != 1 m4 // L7 es una copia de L4 eliminando los números iguales a 1
print L7 // Se imprime [[[2],3],4]
L8 = lfilter > 2 L7 // L8 contiene [[[],3],4]
```

En cualquier sitio que se espere una lista podemos encontrar una expresión que tenga como resultado una lista, pero en algunos casos será necesario el uso de paréntesis para romper la ambigüedad. Por ejemplo, podemos tener:

```
L9 = lfilter != 1 (L4 # L2)
```

Considerad distintas posibilidades según la prioridad que refleje vuestra gramática. Por otro lado, si no es necesario no hay que obligar a poner paréntesis. Por ejemplo, hay que aceptar:

```
fil2 = lfilter != 1 [1,2,3]
```

La operación `lfilter` tiene como parámetros una operación relacional (`>,<,==, !=`), un número y una lista. La operación `lmap` tiene como parámetros una operación aritmética (`+,-, *, /`), un número y una lista. La operación `lreduce` tiene como parámetros una operación aritmética (`+,-, *, /`) y una lista. Las instrucciones del programa están separadas por saltos de línea.

Para construir el analizador sintáctico debemos modificar el fichero Tiny.cup estableciendo la sintaxis de este lenguaje. Como analizador léxico, podemos utilizar el que desarrollamos en la sesión anterior. Debemos prestar atención a utilizar los mismos nombres para las unidades léxicas en los analizadores léxico y sintáctico: los nombres de los no terminales del fichero CUP deben ser los mismos que las unidades léxicas que definimos en ALexOperations.