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 esta práctica vamos a continuar desarrollando nuestro compilador para el lenguaje sencillo, creando un analizador sintáctico que lo detecte. Para ello vamos a utilizar la herramienta CUP.

Como léxico utilizaremos el mismo lenguaje que en la pasada práctica. El fichero .l es prácticamente igual, tan solo hemos cambiado las cabeceras para indicar que queremos trabajar de forma integrada con CUP. La otra diferencia fundamental con respecto al análisis léxico que desarrollamos en la práctica anterior es que el fichero `ClaseLexica.java` lo vamos a generar de forma automática con CUP, no debemos crearlo manualmente. La carpeta `alex` contiene el analizador léxico `AnalizadorLexicoTiny.java` y `AlexOperations.java`, en caso de querer cambiar nuestra sintaxis deberemos modificarlos.

1. En primer lugar, vamos a mirar el archivo `Tiny.cup`. En él se define la gramática incontextual de nuestro lenguaje. Podemos 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:

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

   Esto generará los archivos `AnalizadorSintacticoTiny.java` y `ClaseLexica.java`, que debemos mover a la carpeta `asint`.

2. Desde la carpeta `AnalizadorSintacticoCUP`, compilamos todo el código Java usando el comando:

   ```
   En SO LINUX: $ javac -cp "../cup.jar" */*.java
   ```

   y ya podemos usar el analizador generado.

3. Vamos a probarlo sobre el ejemplo siguiente que está en el fichero `input.txt` de la sesión anterior:

   ```
   # Este es un ejemplo #

   ent x;
   ent y;
   bool b;

   x = 13;
   y = 2;

   si (x == y)[
      b = falso;
   ] sino [
      b = x == y + 2;
   ]
   ```

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

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

No mostrará nada porque la entrada forma parte del lenguaje. Si modificamos la entrada introduciendo algún error sintáctico y volvemos a ejecutarlo vemos como detecta el error.

Posibles cambios a realizar

Para practicar, podemos añadir los cambios que hicimos para extender el análisis léxico a este analizador sintáctico. De cara a añadir los cambios debemos:

1. Modificar los ficheros `mini_lexico.l` y `AlexOperations.java` añadiendo las nuevas unidades léxicas a reconocer y sus constructores.
2. Modificar el fichero `Tiny.cup` añadiendo las nuevas unidades léxicas como terminales. ES IMPORTANTE UTILIZAR LOS MISMOS NOMBRES QUE EN `AlexOperations.java`.
3. Modificar el fichero `Tiny.cup` añadiendo los nuevos no terminales necesarios y las nuevas expresiones a añadir en la gramática.

Algunos cambios que podéis realizar son los siguientes (ordenados de más sencillos a más complejos) son los siguientes (son los mismos en los que trabajamos en la práctica anterior):

- Añadir el operador para la multiplicación
- Añadir otros operadores relacionales y las operaciones entre booleanos
- Añadir instrucciones para permitir bucles
- Añadir números reales y el tipo de número real
- ...

Recuerda que de cara al proyecto final será necesario extender el lenguaje para que incluya las instrucciones básicas presentes en la mayoría de los lenguajes de programación, así como las incluidas en los requisitos vía carta.