Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Compiladores e intérpretes

Tarea III: Generación de código

Profesor: Óscar Víquez

Estudiantes:
Kenneth Sancho, carnet 200941125
Luis Alonso Vega, carnet 201042592

Viernes, 22 de junio 2012

Soluciones e implementación

Para esta etapa final del proyecto se necesitaba realizar la generación de código a partir de un código fuente, ya revisado y analizado por las etapas de análisis sintáctico y contextual.

Como primer paso se analizaron y diseñaron las plantillas para generar el código intermedio bytecode. Al terminar con este paso tuvimos las plantillas siguientes:

If Statement

```
evaluar[expresión]
si falso, ir a else
evaluar[statements para verdadero]
ir a final
else:
evaluar[statements para falso]
final:
```

While Statement

```
check:
evaluar[expresión]
si falso, ir a final
evaluar[statements]
ir a check
final:
```

Assign Statement

```
evaluar[expresión]
almacenar[variable]
```

Print Statement

```
evaluar[expresión]
llamar[imprimir]
```

Message Send Expression

```
evaluar[expresiones enviadas]
evaluar[expresión de instancia]
evaluar[nombre de función llamada]
llamar [función]
```

Method Declaration

```
escribir[cabecera de método]
escribir[límites de stack y locales]
evaluar[parámetros] // Contar parámetros
evaluar[declaraciones de variables] // Contar declaraciones
evaluar[statements] // Contar uso de stack
evaluar[expresión de retorno]
```

```
sustituir[límites de stack y locales]
```

Binary Expressions

(Suma, Resta, Multiplicación, División, Menor que, Mayor que)

```
evaluar[expresión 1]
evaluar[expresión 2]
llamar[operador]
```

Unary Expressions

(Not)

```
evaluar[expresión]
si falso, ir a false
escribir[verdadero]
ir a salida
false:
escribir[falso]
salida:
```

Una vez que tuvimos las plantillas procedimos a realizar un visitor que permitiera hacer el recorrido del árbol.

Contando el total de variables locales y el stack se sustituyen una vez revisado el método. Este tipo de funciones se realizaron con valores globales para el visitor que se restauraban en cada método. Por el resto de la generación, se necesitó investigar un poco más sobre el uso del bytecode. Finalmente al programa se le hicieron unas adaptaciones para cumplir con los requisitos extra: mostrar los archivos generados en nuevas ventanas una vez que se compila el código fuente. También se añadió la opción de correr el código (compilar y correr), el cual genera un proceso que llama al método principal de las clases creadas. Cabe recalcar que para cada clase del archivo de código fuente se debe crear un nuevo archivo. Estos archivos se registran y se generan utilizando jasmin al realizar las tres fases principales.

Si el programa se compila correctamente y se corre, el resultado se muestra en el panel de salida del editor. Se puede modificar el código fuente y generar nuevamente el código sin problemas.

Resultados obtenidos

El proyecto está generando el código correctamente para las funciones que manipulan varios parámetros enteros, además esta funcionando con parámetros de tipo boolean. Funciones tales como las de fibonacci y factorial son algunas de las funciones que hemos utilizado como prueba y las cuales funcionan de manera correcta.

En estas funciones se puede evaluar el funcionamiento de generación de código para “printStatement”, “messageSend”, “ifStatement”, “whileStatement”, “AssignStatement”, así como la generación de código para a los “expression” los cuales son parte indispensables de los programas y que están funcionando exitosamente.

También se lleva el conteo de las veces que se inserta en la pila, así como las veces que se utiliza una variables para conocer los límites de “stacks” y “locals” son muy necesarios cuando se crean los métodos en *jasmin*.

Además de la generación del código intermedio “.j” el programa es capaz de correr mediante un comando en consola (un proceso) para generar los archivos ensamblados “.class” y ya una vez que tenemos esto podemos ejecutar nuestro programa con java. El editor que nosotros hemos implementado se puede compilar el código o ejecutarlo. Junto con esto se nos muestra, en ventanas diferentes, el código que generamos a partir de un archivo o código fuente.

Conclusiones

La finalización de este proyecto fue de suma importancia ya logramos concluir la última etapa de cómo crear un compilador, todas las etapas que se realizaron fueron de suma importancia, pues si fallábamos en uno fallábamos en las otras etapas, tanto el análisis sintáctico como el contextual son de suma importancia para una generación de código correcta.

Ampliamos nuestro conocimiento ya que trabajamos con un generador de código, en este caso jasmín, que nos permitió el desarrollo del proyecto.

El trabajo en equipo es parte importante a la hora de realizar un proyecto de este tipo, ya que es una aplicación más amplia y elaborada, que necesita más trabajo. En grupo el peso de la aplicación se disminuye para cada miembro, haciéndolo más llevadero.

Manual de pruebas

Requisitos

Para ejecutar el programa correctamente se necesita tener en el mismo directorio el *jar* de *Jasmin*.

Archivos de pruebas

Se adjuntan tres archivos de prueba:

- Factorial.java: En este se utilizan llamadas recursivas.
- Fibonacci.java: Llamadas recursivas dobles.
- Sumatoria.java: Ciclo while.
- Template.java: El programa busca este archivo al iniciar. Es una plantilla con contenido básico (una clase principal que imprime cero, otra clase con un método que no recibe parámetros y retorna cero).

Abrir archivo utilizando el menú (ó Ctrl + O)
Run -> Build ó Run para compilar (y correr) el código fuente.

* Posiblemente sea necesario ejecutar el programa llamándolo desde una línea de comandos.

Bibliografía

- Ceng, (sf). JVM and jasmin Tutoria. Visitado el 19 de Junio del 2012 en: http://www.ceng.metu.edu.tr/courses/ceng444/link/f3jasmintutorial.html
- Developerworks, (sf). Java bytecode. Visitado el 19 de Junio del 2012 en: http://www.ibm.com/developerworks/ibm/library/it-haggar_bytecode/
- Blogspot, (2 de enero del 2011). Java bytecode fundamentals. Visitado el 19 de Junio del 2012 en: http://arhipov.blogspot.com/2011/01/java-bytecode-fundamentals.html
- Java World, (9 de enero de 1996). Bytecode basics. Visitado el 19 de Junio del 2012 en: http://www.javaworld.com/jw-09-1996/jw-09-bytecodes.html?page=9