Instituto Tecnológico de Costa Rica
Sede Regional San Carlos

Escuela de Computación
Compiladores e intérpretes

Proyecto 3

Responsables:

Crisly González Sánchez
Kevin Zamora Arias

16 de Noviembre del 2016
Santa Clara, San Carlos

Introducción

La generación de código es considerada la etapa final de la creación de un compilador, en ella podemos aplicar procesos de optimización del código. Dicha etapa consiste en tomar instrucciones escritas en alto nivel y traducirlas a código máquina modelando así un código objeto, entendiendo así que el código objeto el medio entre el lenguaje de alto nivel y máquina.

Cabe destacar que para llegar a dicha etapa se requiere un proceso previo de Análisis contextual, por lo tanto ya estaríamos generando código objeto sin errores, por medio de la herramienta ASM del lenguaje Java se logró hacer la traducción correcta para hacer comunicación con lenguaje máquina. Como resultado se observará la creación de código de alto nivel escrito de acuerdo a las etiquetas bytecode que fueron estudiadas previamente.

Se logró comprender el proceso de desensamblaje del código de alto nivel y a su vez conocer las distintas funcionalidades de las etiquetas que a través de línea de comandos fueron expresadas en el proceso de desensamblaje a través de “javap”.

Análisis del problema

En la etapa de generación de código es requerido la creación de un archivo de lectura con las reglas de la gramática ejemplificadas, ya que se deberá tomar lectura de él e iniciar la etapa de desensamblaje para lograr código objeto.

Para esta parte del proyecto es recomendable analizar las instrucciones que por medio de ASM se visualiza un código de alto nivel hasta convertirse en bytecode, a través de la línea de comandos el equipo de desarrollo deberá crear un archivo de prueba con métodos que contengan cada una de las reglas de gramática mini Java, es así como por medio del desensamblaje visualizamos los cambios que sufre el código en su transformación a código objeto.

Algunas de las instrucciones más comunes que se utilizarán en el proyecto son: ICONST, BIPUSH, NewArray , ASTORE, CASTORE, ILOAD, ALOAD, etc. Tener presente los tipos de datos manipulados en el compilador que se ha venido creando facilitará conocer cuál de las anteriores instrucciones podrá utilizar, ejemplo: Cuando deseamos almacenar una variable en pila y es su valor es mayor a 5 utilizamos la instrucción BIPUSH. La librería ASM cuenta con un set de instrucciones ampliamente documentadas que facilitan la investigación de sus usos.

Para el manejo de las posiciones de los datos cargados en pila, se propone crear una lista de objetos variables, que contienen nombre, tipo. Para obtener la posición de ese objeto lo realizaremos mediante su índice en la lista.

El manejo de la pila en la programación de instrucciones bytecode es fundamental, en el lenguaje de programación Java, ese manejo queda a responsabilidad del programador, por lo tanto deberán realizarse los cálculos necesarios para incluir estos valores antes de cerrar la escritura del método.

Se deberá crear modularidad en las clases del proyecto cada sección creada , Generación de código, Análisis Contextual, Análisis Sintáctico y Interfaz de usuario deberán estar en paquetes distintos, esto facilitará la comprensión / interpretación del código y sus etapas desarrolladas.

Se recomienda verificar los diferentes tipos de lectura que tiene el lenguaje de programación java, para la implementación de la regla gramatical read.

Solución del problema

Se crea un archivo de prueba con las instrucciones de cada una de las reglas de la gramática mini java, este archivo es leído desde el Main y comienza así la etapa de generar instrucciones bytecode.

Se requirió la creación de un método **construirClase** para crear el String del atributo que debe asignársele a un método cuando es creado para indicar quien es su clase, se recibió el nombre y a partir de este se construye un String nombreClase.java. Además se creó el **método tipoDe** el cual se encarga de buscar el objeto variable en la lista variablesDeMetodos y retorna el tipo de variable, esto con el objetivo de que algunas de las instrucciones bytecode de ASM se requirió analizar el tipo de datos y así crear en pila la instrucción más indicada, un ejemplo simple es subir un tipo de arreglo a la pila ya que si es int se gira la instrucción ASTORE, si es char CASTORE y en caso boolean sería BASTORE; aspectos como los anteriores forzaron la creación de un método como Tipo de variable. Otro método es el indiceDe que su funcionalidad es buscar un objeto en la lista y devuelve el índice.

Como se ha mencionado anteriormente, se creó una **lista de variables** que son almacenadas en la etapa de ejecución cuando hay una declaración de parámetro o variable en la clase. Esto facilitó el manejo de los datos en la pila y la ejecución de las instrucciones.

Se implementó el tipo de lectura System. Console(). readLine() que es utilizado en java para tomar datos de consola, es así como se implementó la regla gramatical read, por lo tanto cuando sea llamada desde el archivo prueba tiene como resultado la descompilación de la misma en código de alto nivel.

Análisis de Resultados

Tarea	Estado	Observaciones
Generación de código de la gramática MiniJava	100%	
Creación de documentación del proyecto	100%	
Implementación de las instrucciones bytecode ASM	100%	
Manejo de la pila	100%	
Creación de clase con código objeto.	100%	
Creación de Archivo de prueba para lectura	100%	

Conclusión

Se logró el análisis completo de la etapa generación de código desde el Análisis Contextual, Sintáctico y la creación de código objeto. Además la interpretación e investigación de reglas del api ASM permitió la creación de dicha fase.

ASM es una librería presente en el lenguaje de programación Java , mediante sus instrucciones podemos crear una etapa de generación de código de acuerdo al compilador, su extensa documentación permite entender el funcionamiento y sus componentes en cada instrucción.

La creación de las etapas anteriores con la menor cantidad de errores ayudó en el desarrollo completo de la tercera entrega, ya que el equipo de desarrollo se enfocó en generar el código y no se invirtió tiempo resolviendo problemas previos a inicio del proyecto.

Las tres etapas de un compilador están fuertemente relacionadas, debemos analizar cada una de ellas para lograr un correcto entendimiento del funcionamiento del compilador creado, al ser un desarrollo en cascada puede dar como resultado negativo, que problemas anteriores a la iniciación de la etapa afecten nuestra generación de código.

Recomendaciones

1. Se debe tener conocimiento previo de las instrucciones de la librería ASM ya que la etapa de generación de código está basada en las instrucciones que se puede generar de dicha librería.

2. Es recomendable analizar el proceso de desensamblaje que realiza el compilador de java, además de entender la funcionalidad de cada una de las instrucciones en pila.

3. Es fundamental realizar un manejo detallado de la pila, con sus variables de ingreso y el crecimiento y decrecimiento de la misma, esto facilita el trabajo que se debe realizar con las instrucciones de generación de código.

4. La traducción de lenguaje de alto nivel a código objeto debe ser cuidada minuciosamente ya que una instrucción mal escrita puede aplazar el funcionamiento de la generación del código.

5. Se debe realizar un análisis previo del desarrollo de la etapa de Análisis Sintáctico y Análisis Contextual, con el fin de verificar que no existan errores que impidan la etapa de Generación de Código.

Bibliografía

SR. (SR). Compilador Diseño - Generación de Código. 13 de Noviembre del 2016, de SR Sitio web: https://www.tutorialspoint.com/es/compiler_design/compiler_design_code_generation.htm