Informe de Entrega CoolCompiler 2020

Jorge Daniel Valle Días JORGE.VALLE@ESTUDIANTES.MATCOM.UH.CU
Grupo C412

Leonel Alejandro García López L.GARCIA3@ESTUDIANTES.MATCOM.UH.CU
Grupo C412

Roberto Marti Cedeño R.MARTI@ESTUDIANTES.MATCOM.UH.CU
Grupo C412

Tutor(es):
Msc. Alejandro Piad Morffis, *Facultad de Matemática y Computación, Universidad de La Habana*

Tema: Compilación, Cool Language.

1. Introducción

El siguiente trabajo representa el informe sobre la confección del compilador. Para la confección del mismo se empleó el lenguaje python de programación. El proyecto se dividió en varias etapas: Análisis lexicográfico, sintáctico, semántico, código intermedio y generación de código de máquina. Es importante destacar que se reutilizaron en la medida de lo posible los archivos de clase práctica y los proyectos realizados en el curso previo de la asignatura.

2. Lexer

En la fase de análisis lexicográfico se empleó la biblioteca ply, en especial su módulo lex. Salvo en el caso de las cadenas de caracteres y los comentarios multilínea, el resto de los tokens fueron procesados mediante expresiones regulares. Para el caso de las cadenas de caracteres y los comentarios multi-líneas se emplearon estados especiales exclusivos.

3. Parser

La fase de análisis sintáctico ha sido una de las fases mas controversiales a la hora de la realización del proyecto. Se empleó en una primera fase el parser LR1 tomado de clase práctica y los proyectos previos de la asignatura. La gramática que se definió durante la primera entrega del proyecto contenía ambigüedades, las cuales fueron detectada comprobando las pruebas correspondientes a la parte de semántica.

Dado el tiempo restante que se disponía para la entrega, el equipo decidió tratar de adelantar todas las funcionalidades posibles para minimizar los cambios después de la fecha de entrega.

Actualmente se esta valorando la posibilidad de emplear el módulo yacc de ply para definir la gramática y evitar las ambigüedades que se desprenden de nuestra implementación inicial.

4. Análisis Semántico

La fase de análisis semántico se compuso por 3 recorridos del árbol de sintaxis abstracta derivado de la fase de análisis sintáctico que siguen el patrón visitor.

Recolector de tipos: Primer recorrido del ast, en el cual se conforman los tipos nativos y los definidos en el archivo de código a procesar. En este mismo recorrido también se detectan los problemas relacionados con la herencia cíclica.

Constructor de tipos: Segundo recorrido del ast, en el cual se construyen los tipos, se definen sus métodos y atributos.

Verificador de tipos: Tercer y ultimo recorrido del ast, en el cual se verifica la estructura de cada uno de los nodos del ast, este recorrido es el que mas abarca las reglas semánticas del lenguaje cool.

5. Código intermedio y código de máquina

El equipo decidió realizar una representación intermedia del lenguaje cool antes de pasar a la generación de código de máquina. Para ello se definieron 2 nuevos recorridos a árboles de sintaxis.

5.1 Código intermedio

Cuarto recorrido del ast, en el cual se conforma la representación del árbol de código intermedio de cool. En el mismo nos dimos a la tarea de simplificar la complejidad de cool para llevarlo a mips sin mucha dificultad. Entre las transformaciones no evidente las siguientes:

- **Relacionado a la herencia** mantenemos como tipos de cil como el conjunto de atributos de los respectivos en cool pero con los de sus antecesores en la jerarquía de tipos(los métodos y la tabla virtual queda para resolverse en mips)

- **Para las estructuras *CaseOf*** de cool realizamos un ordenamiento (dicho ordenamiento fue realizado de acuerdo a la profundidad de los tipos en la jerarquías, desde el más específico, hasta el mas cercano a *Object*) de los *case actions* para realizar un *matching* no iterativo y a partir del primer tipo presente en la jerarquía del objeto resultante de la expresión del case. Luego esta lista que llamamos **CaseActionExpressions** se extende de manera ordenada para añadir los tipos mas específicos de los que se encontraban originalmente que a su vez como expresión a realizar sería la mismo que el ancestro del que extendio, quedando una lista con bloques de case donde antes solo una opción (conservando el orden anterior pero solo por bloques), de forma tal que en tiempo de ejecución el primer *matching* fuese el correcto a realizar.

5.2 Código de máquina

Último recorrido, en este caso del árbol de sintaxis de representación intermedia. Mediante el cual se genera el código a ejecutar en el microprocesador con arquitectura MIPS.