Compilador de Cool

Dayrene Fundora González— C411

Wendy Díaz Ramírez — C412

Complementos de Compilación.

4to año, Ciencia de la Computación.

MATCOM, UH.

1.Arquitectura

Arquitectura del Compilador

El punto de entrada es el código de COOL, y la entrada de cada etapa es la salida de la etapa anterior, excepto en la primera que es el propio código COOL.

i. **Analizador Léxico**: El código COOL es procesado por un código generado automático por ANTLR4 a partir de la gramática de COOL definida.
ii. **Analizador Sintáctico**: A partir del árbol devuelto por el leer se construye el AST y se construye la tabla de símbolos.
iii. **Analizador semántico:** Se hacen 2 recorridos sobre el AST para realizar el chequeo semántico.
iv. **Generación de Código Intermedio:** A partir del AST ya chequeado, se genera un árbol que contiene nodos de CIL como lenguaje intermedio.
v. **Generación de Código MIPS:** A partir de cada nodo del árbol de CIL generar su correspondiente código MIPS.

2. Analizador Léxico y Analizador Sintáctico

La implementación del *tokenizer* y del *parser* se realizaron con ANTLR v4.7.1. Se especificaron las reglas del *lexer* y del *parser* tomando como base la gramática 'Cool.g4', esta gramática fue diseñada bajo la ya existente en el MIT. Una vez la herramienta nos proporciona la información necesaria se procede a la construcción del AST. Cabe destacar que ANTLR4 trata la recursión izquierda por nosotros eliminando la ambigüedad de esta y establece el orden de las operaciones acorde del orden en que definimos las producciones para cada no terminal.

3. Análisis Semántico

Después de tener nuestro AST armado vendría el chequeo semántico donde recorremos nuestro AST 2 veces, antes ordenamos las clases por orden topológico y revisamos los casos de herencia cíclica o clases repetidas.

La primera pasada recorremos cada clase capturando la definición de los tipos, las propiedades y los métodos para cada clase, además de ir añadiendo al scope la herencia, para después saber los métodos del padre para cada clase. En la segunda pasada analizamos las expresiones definidas en el cuerpo de los métodos y las inicializaciones de las propiedades, analizando el tipo estático de cada método y atributo.

Se implementó *IScope* para atrapar los 'tipos', los 'métodos' y 'propiedades' por cada uno, los 'parámetros' para cada uno de los métodos además de tener funcionalidades que nos serán de gran ayuda como saber si un tipo o método está definido entre otras cosas.

3. Generación de Código Intermedio

Se hacen dos pasadas, primero para almacenar las clases básicas y las posibles nuevas definidas por el programador y definir los métodos que tiene cada clase en orden topológico de forma que cada vez que se define una clase nueva esta también almacena los métodos de la clase que hereda. La segunda pasada es para generar el código de las '*direcciones*' de las funciones.

4. Generando de Código de MIPS

El programa en MIPS se divide en 2 secciones fundamentales:

.*data* En la cual van las definiciones de tipos, excepciones, strings, y la herencia

.*text* Que contiene todo el código del programa, empezando por los constructores de los tipos básicos y sus funciones, después un *main* que es donde comienza el código a generar.

5. Como ejecutar

La ejecución del compilador será sencilla, de la siguiente manera se efectúa:

$ cd src

Caso de Linux $ ./cool.sh <file.cl>

Caso de Windows $ cool.sh <file.cl>