Pablo Adrián Fuentes González C-511

Uso del Compilador:

El compilador está implementado en Python, por lo que para usarlo debe ejecutarse el script llamado main.py pasándole como parámetro el fichero de COOL (.cl) para ser compilado. Tras dicha ejecución se generará un fichero con extensión .mips con el código MIPS equivalente al programa de entrada. En caso de ocurrir errores durante el proceso de compilación se reportará el error por la salida estándar y culminará el proceso con código de salida exit(1).

Arquitectura del Compilador:

El proceso de compilación se divide en 4 etapas:

1- Análisis Lexicográfico:

Durante esta fase se realiza la conversión de la secuencia inicial de caracteres proveniente del código fuente a una secuencia de tokens. En esta fase se detectan errores relacionados con la escritura incorrecta de símbolos.

2- Análisis Sintáctico:

En esta fase se determina la estructura sintáctica del programa a partir de los tokens y se obtiene el árbol de sintaxis abstracta del programa. La herramienta utilizada para los análisis léxicos y sintácticos es PLY la cual es una implementación de las herramientas lex y yacc de C. Esta hace uso de LALR-parsing. La gramática que se propone posee varios conflictos shift/reduce los cuales se resuelven al definir la precedencia de símbolos en PLY y definir un orden de definición de las producciones.

3- Análisis Semántico:

En esta fase se realiza el chequeo semántico y de tipos. Fue necesario implementar ciertas estructuras de datos que permitieran encapsular toda la información semántica de un programa. Debido a que en COOL, es posible usar una clase que aún no ha sido definida; o sea, que la definición de dicha clase se encuentre después de la línea de código donde se usa; entonces es necesario comprobar primero que dicha clase y sus miembros realmente existen, para después conformar la jerarquía de tipos y por último hacer el chequeo semántico. Esta fase se divide en varias subetapas.

a- Se crean los tipos existentes. Se verifica que no se redefina ningún tipo básico, que solo exista una única implementación de un tipo A, y que ningún tipo herede de los tipos básicos: String, Int o Bool.
b- Una vez creados los tipos, se pasa a verificar que no exista problemas con la herencia. Que no existan ciclos de herencia. Que una clase no defina la misma propiedad que uno de sus ancestros. Que si alguna clase redefine un método de su clase padre lo haga correctamente.
c- Se pasa a crear el scope de las variables, y posteriormente al análisis semántico y de tipos.

Para recorrer cada nodo del AST se utiliza el patrón visitor.

4- Generación de Código:

Una vez llegados a esta fase ya han sido chequeados todos los errores lexicográficos y semánticos y ya ha sido conformado el árbol de sintaxis abstracta que encierra la semántica del programa que se está compilando. Por lo que el próximo paso a dar es la construcción de un AST para CIL, un lenguaje intermedio que suavizará el salto desde COOL hasta MIPS y que resolverá problemas como los Dispatch estáticos y dinámicos (la tabla de métodos virtuales). Usando el patrón visitor se recorre nuevamente el AST y se crea el AST para CIL, y luego se pasa a la escritura de código MIPS. En esta fase se resuelven e implementan además los métodos de los tipos básicos de COOL: Object, IO, Int, String y Bool.

Problemas Técnicos:

El chequeo de tipos en Runtime se resuelve utilizando una tabla generada en MIPS almacenada en la memoria estática, que guarda en sus primeros 4 bytes la referencia a la posición en la tabla donde se encuentra el padre, en los 4 siguientes el nombre del tipo y posteriormente referencias al punto de entrada de cada método. En caso de Object, la referencia a su padre es 0.
En caso de detectarse algún error en Runtime finaliza el programa sin mostrar el tipo de error.

Un problema interesante fue a la hora de transformar tipos por valor a tipos por referencia a través del tipo Object. Para ello fueron necesarios controles a la hora de generación de MIPS para detectar tipos por valor y reservarles espacio en memoria para almacenarlos como tipos por referencia.

En cuanto, al momento del unboxing en la instrucción case hubo que detectar si lo que se tenía en la variable requería de hacerle unboxing para hacer la comparación de tipos en las ramas de la instrucción.