Compilación
COOL Compiler

Karl Lewis Sosa Justiz
Juan David Menéndez del Cueto
C-412

November, 2020

Uso del compilador

Para usar el compilador es necesario tener instalado Python 3.7 o superior. Todos los requerimientos pueden ser instalados ejecutando `pip install -r requeriments.txt` desde la raíz del proyecto.

El módulo que contiene toda la lógica del compilador es `exec_cool.py`. Para utilizarlo solo es necesario pasarle como argumento un fichero con código fuente de COOL, por ejemplo, ejecutar `python exec_cool.py <path>`. Un archivo en el mismo path del código fuente será creado, con el mismo nombre, pero con extensión .mips. Este fichero contendrá código MIPS con pseudo instrucciones y se puede probar en cualquier implementación del simulador SPIM.

Arquitectura del compilador

La estructura del compilador se puede separar en módulos, que pertenecen a las fases del desarrollo del mismo. A continuación se explicarán las fases antes mencionadas y las ideas principales seguidas durante su desarrollo.

Análisis lexicográfico y sintáctico

El análisi lexicográfico y sintáctico trata de la comprobación del programa fuente hasta su representación en un árbol de derivación, incluye desde la definición de la gramática hasta la construcción del lexer y el parser.

En estas fases se emplearon las herramientas de construcción de compiladores lex y yacc a través del paquete de *Python* ply.

El lexer y el parser del proyecto se encuentran implementados en los módulos lexer.py y Parser.py respectivamente, además AstNodes.py ofrece la jerarquía del AST de COOL propuesta.

Análisi semántico

El análisis semántico consiste en la revisión de las sentencias aceptadas en la fase anterior mediante la validación de que los predicados semánticos se cumplan.

El árbol de derivación es una estructura conveniente para ser explorada. Por lo tanto, el procedimiento para validar las reglas semánticas es recorrer cada nodo. El patrón visitor fue el utilizado para esta tarea.

En el primer recorrido se recogen todos los tipos definidos; en el segundo se construye el contexto de métodos y atributos; y en la tercera pasada se procede a la construcción de los escopes recolectando las variables definidas en el código, teniendo en cuenta su visibilidad. Por último, se verifica la consistencia de tipos en los nodos del AST.

Generación de código

Esta fase consiste en la generación de código de máquina a partir del código ya procesado y validado por las fases anteriores. Debido a la dificultad de llevar

directamente el código de COOL a MIPS es demasiado alta, se genera un código intermedio para facilitar esta tarea.

CIL

Como lenguaje intermedio para este proceso, se utilizó el lenguaje CIL, un lenguaje parecido al dado en clases pero al que le fueron añadidas algunas instrucciones para facilitar el proceso de generación de código MIPS.

Para la generación de código CIL se usó el patrón visitor, generando para cada nodo del AST de COOL el conjunto de instrucciones equivalentes en este lenguaje. Además, a cada variable, atributo y función se le asigna un identificador único para facilitar la generación de código MIPS.

MIPS

Se recorren los nodos creados de CIL usando, nuevamente, el patrón visitor, creando las operaciones de MIPS equivalentes para cada instrucción.

Problemas técnicos

Análisis lexicográfico y sintáctico

En esta fase el mayor problema enfrentado fue la tokenización de strings y comentarios de varias líneas. El mismo fue resuelto mediante el uso de estados que brinda `ply`.

Análisis semántico

El mayor problema enfrentado fue el trabajo con los scopes, principalmente en las instrucciones `case` y `letIn`.

Generación de código

En esta sección, los problemas más interesantes fueron: la necesidad de hacer `Boxing` cuando era necesario castear los tipos `String`, `Bool` o `Int` a `Object`; y la necesidad del valor `void`, el cual fue manejado como una dirección de memoria.