Reporte Escrito

Compilador para Go

Juan David Lara Camacho
jlara@unal.edu.co

Santiago Jimenez Salazar
sajimenezs@unal.edu.co

2 de diciembre del 2022

.....

1. Introducción

El lenguaje de programación `Go` (también llamado `Golang`) fue creado en 2009 por `Google`. Es un lenguaje concurrente y compilado con tipado estático inspirado en la sintaxis de `C`, pero con seguridad de memoria y recolección de basura. En el siguiente trabajo se diseña un (mini) compilador para Go, partiendo por una gramática libre de contexto(GLC) que genera su lenguaje y las definiciones del mismo tomadas de [4, 3]. Luego, mediante el uso de dicha gramática para construir un parser para dicho lenguaje.

2. Materiales y métodos

Un compilador para Golang implementado con Python Lex y YACC (PLY) [2] que maneja las construcciones de interruptor y bucle de Golang. Para esto usamos PLY el cual es una implementación de herramientas de análisis sintáctico lex y yacc para Python. PLY proporciona la mayoría de las características estándar de lex/yacc, incluyendo soporte para producciones vacías, reglas de precedencia, recuperación de errores y soporte para gramáticas ambiguas [1, 2].

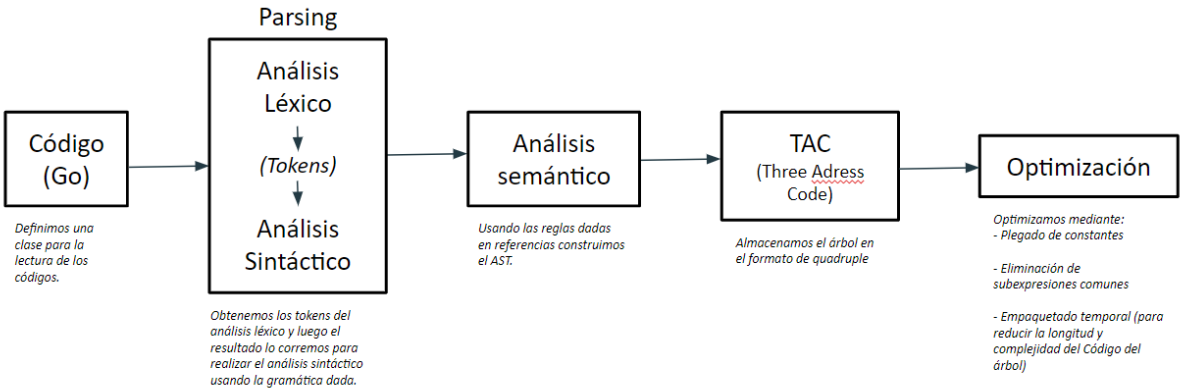

Figura 2.1: Diagrama de solución

En la Figura 2.1 encontramos el diagrama de desarrollo, este corresponde a las partes de un

compilador [1] dada la intención del proyecto, en particular en cada uno de los pasos (o cajas del diagrama) realizamos un archivo en Python, primero para tratar el código definimos unas clases las cuales nos permitan tratar la cadena de texto o los códigos escritos en el lenguaje Go. Luego, para el parsing usamos la gramática definida en [4] y generamos una tabla mediante la herramienta PLY ya mencionada. Finalmente trabajamos emparejand y encontrando coincidencias, tratamos los datos para construir el árbol dado un código inicial y este lo almacenamos en el formato *quadruple*, el cual nos es útil para la optimización. Para la optimización usamos plegado de constantes, eliminación de subexpresiones comunes y empaquetado temporal el cual nos sirve para reducir la longitud y complejidad del código del árbol.

3. Resultados

Se observa que el front-end del compilador genera una representación intermedia correcta y optimizada del programa de entrada de alto nivel escrito en Golang. El código de tres direcciones(Three Adress Code) se genera en formato cuádruple y las entradas de la tabla de símbolos se muestran correctamente.

El front-end del compilador fue diseñado completamente con Python Lex y Yacc, además funciona en entornos Windows y Linux, con un mínimo número de dependencias de paquetes y con un rendimiento razonable para programas de entrada de tamaño mediano.

Es muy probable que el código optimizado final sea menos eficiente que el generado por la implementación de referencia real del compilador Golang. Además, no se implementaron algunas características únicas del modelo de programación Go, como los canales y el modelo de concurrencia que involucra programación avanzada de bajo nivel.

4. Discusión y conclusiones

En el presente trabajo se realizó un análisis detallado de un compilador para Golang, desde la gramática, la cual fue diseñada para ser LALR(1), hasta sus tokens y su tabla de parsing. Para este trabajo, se utilizaron herramientas de Python como Lex y Yacc que nos permiten capturar la lógica de las estructuras de control (if, switch) y de iteración de Golang.

Mediante el parser construido, se puede verificar, para cualquier expresión dada, si su sintaxis es correcta en el lenguaje y en caso afirmativo, se logró construir un árbol de derivación para dicha expresión. Usando este árbol de derivación, podemos ejecutar en Python la instrucción dada, en caso de que sea de carácter aritmético realizando un cálculo, o aplicando cierta lógica en caso de que sea una estructura de tipo for, if, else o switch.

Este proyecto nos permitió estudiar a fondo una aplicación práctica de los temas vistos en el curso. Se usaron lenguajes regulares para construir los tokens del lenguaje, tal como se vió en el Flex que se diseñó para la primera entrega del curso. También se construyó también una tabla de parser de manera similar a como se vió en el curso. Además, se hizo un uso intensivo de temáticas de cursos anteriores como Programación orientada a objetos y Estructuras de datos para construir el árbol de derivación de una oración dada en el lenguaje y los métodos que permiten manipularla.

Respecto a los posible proyectos que pueden realizarse con base en este encontramos:

- ◇ Implementación de reglas sintácticas y semánticas para las sentencias de bifurcación condicional e incondicional (if, break, goto).

- ◇ Más técnicas de optimización basadas en la eliminación de código muerto (implementación de análisis de variables vivas y adición de información de uso siguiente a la tabla de símbolos).

5. Referencias

[1] Alfred V. Aho and Alfred V. Aho, editors. *Compilers: principles, techniques, & tools*. Pearson/Addison Wesley, Boston, 2nd ed edition, 2007. OCLC: ocm70775643.

[2] dabeaz. PLY (Python lex-yacc) — ply 4.0 documentation.

[3] Google. Codewalk: first-class functions in go - the go programming language.

[4] Google. The go programming language specification - the go programming language.