CENTRO REGIONAL UNIVERSITARIO CÓRDOBA
IUA

“DESARROLLO DE UN COMPILADOR DE C CON ANTLR4”

DESARROLLO DE HERRAMIENTAS DE SOFTWARE

PROYECTO

PROFESOR

Eschoyez, Maximiliano Andrés

INTEGRANTES

Carrizo, Matías

Llamosas, Simon

15 de diciembre de 2023

Índice General

1. Introducción

En este informe, se explorará el proceso de construcción de un compilador para el lenguaje de programación C utilizando ANTLR4, una herramienta de generación de analizadores léxicos y sintácticos, junto con el lenguaje de programación Python.

Se presentará una visión global del proceso de construcción del compilador, abarcando desde la definición de la gramática hasta la implementación práctica en Python utilizando ANTLR4. Se explorarán aspectos clave como la estructura de la gramática y su definición, el uso del analizador léxico, la confección de la tabla de símbolos, entre otras cuestiones de importancia.

2. Desarrollo

2.1 Las etapas de la compilación

El compilador atraviesa seis etapas antes de generar el código objeto, las cuales son las siguientes:

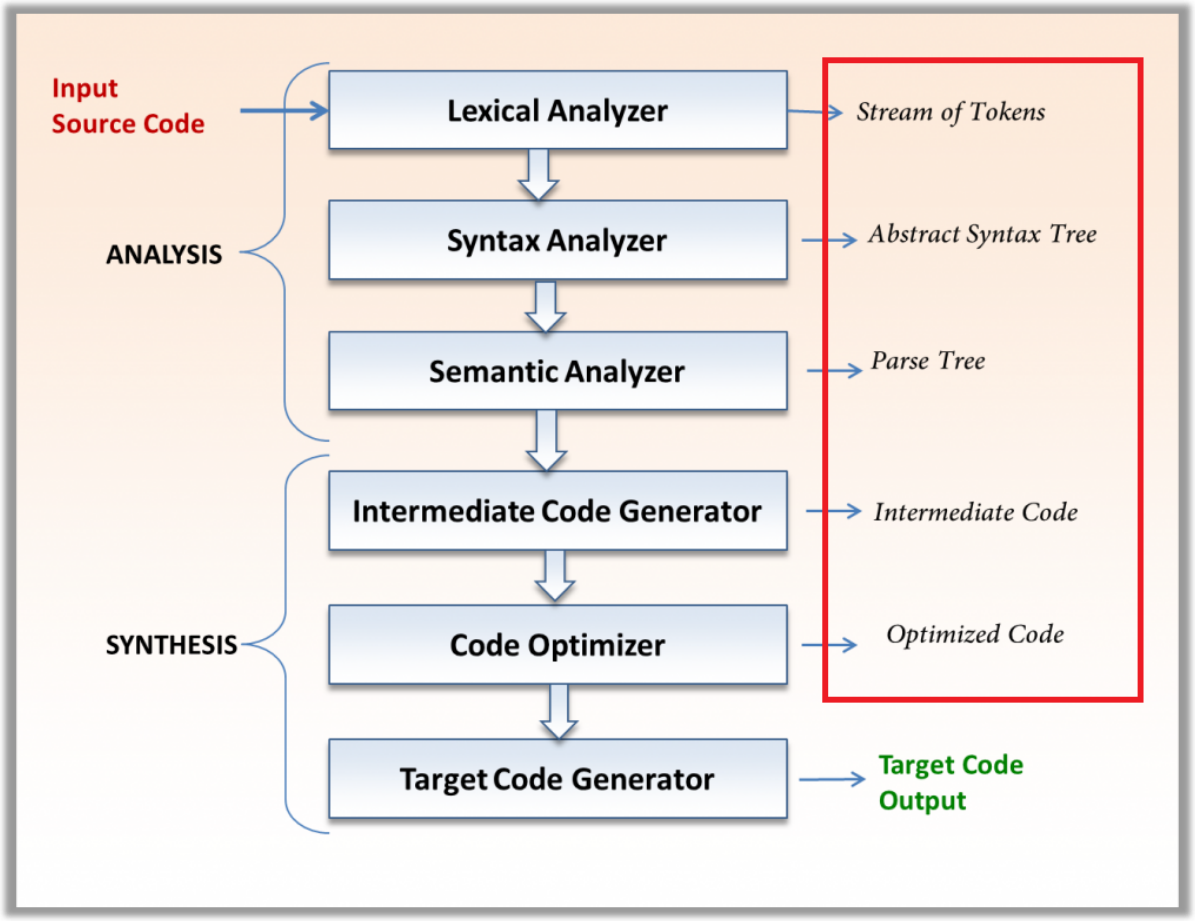

Figura 1.1: Las etapas del proceso de compilación

El compilador diseñado se centra específicamente hasta la etapa de la generación de código intermedio optimizado, La última etapa escapa de los alcances del presente proyecto, pero podría ser considerada para futuras implementaciones.

En la siguiente sección, presentaremos las implementaciones de cada etapa utilizando ANTLR4 y Python.

2.2 Implementación de las etapas

2.2.1 Análisis Léxico

En el contexto del análisis léxico, ANTLR4 toma la gramática definida y genera un analizador capaz de dividir el código fuente en tokens, los cuales representan unidades básicas como identificadores, palabras clave, números, operadores, entre otros elementos fundamentales del lenguaje. Estos tokens son utilizados en etapas posteriores del proceso de compilación para construir la estructura del árbol de análisis sintáctico.

ANTLR4 utiliza archivos con extensión ".g4" los cuales son, en esencia, archivos de gramática que describen las reglas y estructuras de un lenguaje en particular. Estos archivos, también conocidos como archivos de gramática ANTLR4, son la base para la generación de analizadores léxicos y sintácticos.

Tras el armado de las reglas gramáticas, y una vez ejecutado el archivo ".g4", ANTLR4 se encargará de definir y crear los tokens correspondientes para cada regla, creando los archivos correspondientes para su uso, como lo son *compiladoresLexer.py* (que contiene la implementación del analizador léxico en Python) y *compiladoresLexer.tokens* (el cual enumera y asigna un identificador numérico a cada token reconocido por el analizador léxico).

2.2.2 Análisis Sintáctico y Semántico

Desde ANTLR4 podremos ver gráficamente el árbol sintáctico generado. A fines prácticos, no lo incluimos por ser muy grande, pero puede verse en el repositorio correspondiente.

Para llevar a cabo el proceso de análisis sintáctico y semantico, realizamos la implementación de un Listener y un Visitor, de la mano de ANTLR.

- El *myListener* abarca un conjunto de lógicas destinadas a examinar las diversas reglas gramaticales implicadas en el código fuente para la generación de la tabla de símbolos. Durante esta etapa, se efectúa un análisis de la estructura gramatical del código, permitiendo la construcción precisa y ordenada de la tabla de símbolos, la cual será empleada en las fases posteriores del análisis.
- Por otro lado, *myVisitor* encapsula el conjunto de algoritmos y procedimientos diseñados para verificar la corrección sintáctica y semántica del código fuente. Durante esta etapa, se verifica la exactitud y coherencia de las instrucciones, garantizando su adecuación a las reglas del lenguaje y su razonabilidad lógica. Además, este proceso facilita la generación del código intermedio en forma de código de tres direcciones, una representación intermedia que simplifica la lógica del código fuente original para facilitar su posterior traducción o interpretación.

Algunos ejemplos de la implementación del código de tres direcciones se ven a continuación.

Código de 3 direcciones

INSTRUCCIÓN	DESCRIPCIÓN
Asignación	x = y **op** z donde **op** = + , **-** , < , > , <= , >= , !=, etc
Etiqueta	label l**n** donde $n = \mathbb{N}$
Salto	jmp l**n** donde $n = \mathbb{N}$
Salto	jmp l**n** donde $n = \mathbb{N}$
Temporal	t_n donde $n = \mathbb{N}$
Condicional	if not **a** jmp **b** donde **a** = condición y **b** = ln

2.2.3 Optimización del código intermedio

Durante esta etapa, se llevó a cabo la implementación de estrategias de optimización destinadas al código intermedio generado. Se adoptaron técnicas como la sustitución de

temporales, entre otras optimizaciones, con el objetivo de mejorar la eficiencia y la legibilidad del código producido por el compilador.

Estas optimizaciones están destinadas a aplicar cambios que mejoren la eficiencia del programa resultante, contribuyendo así a la optimización y la calidad del código intermedio producido por el compilador.

3. Conclusión

En conclusión, los resultados obtenidos en el desarrollo del proyecto del compilador de C, utilizando ANTLR y Python, han cumplido con las consignas solicitadas y con nuestras expectativas.

El trabajo realizado nos ha permitido adentrarnos en el complejo mundo de la construcción de compiladores, comprendiendo detalladamente cada etapa del proceso, desde el análisis léxico hasta la generación de código intermedio. La utilización de ANTLR junto con Python facilitaron la implementación de las distintas fases del compilador, ofreciendo una mayor comprensión y control sobre la gramática del lenguaje, las reglas de traducción y la gestión de la semántica del código, permitiéndonos manejar de una manera más sencilla la complejidad que conlleva confeccionar un compilador.

Además de lograr resultados funcionales en la generación de código intermedio a partir del código fuente en C, este proyecto nos ha brindado una perspectiva más profunda sobre cómo funciona un compilador, fortaleciendo nuestro entendimiento teórico y práctico sobre la construcción de lenguajes de programación, el análisis gramatical y las estrategias de traducción de código.

4. Anexo

Link del repositorio (GitHub): https://github.com/simonll4/C-Compiler.git