Compilador para el lenguaje Hulk

John García Muñoz - Grupo: 311
Victor Hugo Pacheco Fonseca - Grupo: 311
Jose Agustín del Toro González - Grupo: 312

June 22, 2025

Contents

1 Introducción

1.1 Sobre el Proyecto

El presente informe describe el desarrollo e implementación de un compilador completo para el lenguaje de programación Hulk, diseñado como un lenguaje tipado con soporte para programación orientada a objetos y herencia simple. El proyecto se encuentra alojado en GitHub en la siguiente dirección: github
Este compilador ha sido construido desde cero, abordando cada una de las fases fundamentales del proceso de compilación:

- Análisis léxico (Lexer): Se implementó un analizador léxico personalizado que convierte el código fuente en una secuencia de tokens, reconociendo literales, identificadores, palabras clave y símbolos del lenguaje
- Análisis sintáctico (Parser): Se diseñó un parser predictivo descendente LL(1) completamente manual, capaz de generar un arbol de parser a partir de la gramática del lenguaje Hulk.
- Análisis semántico: Se implementaron varias capas de verificación, destinadas a diferentes fases de la semántica.
- Generación de código intermedio: Finalmente, se implementó la generación de código utilizando la infraestructura de LLVM IR, permitiendo traducir los programas escritos en Hulk a una representación de bajo nivel.

2 Requerimientos

A continuación los requerimientos de software necesarios para la ejecución del proyecto.

2.1 Requerimientos de Software.

- CMake (versión 3.10 o superior):Utilizado como sistema de construcción para organizar, compilar y enlazar los módulos del compilador.
- Compilador de C++ moderno (GCC o Clang):Se requiere soporte completo para C++17 o superior
- LLVM (versión 14 o superior):Hulk genera código intermedio LLVM IR, por lo que se necesita:llvm-config para obtener flags de compilación;Bibliotecas de LLVM (como LLVMCore, LLVMIR, etc.)

3 Características del Lenguaje Hulk

El lenguaje Hulk es un lenguaje de programación de alto nivel diseñado como un proyecto educativo para explorar la implementación de compiladores modernos.Para un informe mas detallado sobre el lengajue visitar el enlace: matcom/hulk .A continuación se detallan sus principales características implementadas en este proyecto:

3.1 Programación Orientada a Objetos (POO)

Hulk soporta un sistema de tipos definido por el usuario mediante la palabra clave type, que permite definir clases con atributos, métodos y constructores. Cada tipo puede encapsular datos y comportamiento.Donde todos los atributos son privados y todos los metodos son virtuales.

3.2 Herencia Simple

El lenguaje permite herencia simple, lo que significa que una clase puede heredar de una sola clase base. Esta herencia facilita la reutilización del código, permitiendo que una clase hija acceda a los métodos de su clase padre.

3.3 Estructuras de Control

- Ciclos while:Incluye soporte para ciclos while, permitiendo repetir bloques de código mientras una condición booleana se mantenga verdadera:
- If-else:Permite controlar el flujo del programa mediante estructuras condicionales que soportan múltiples ramas usando if, elif y else

3.4 Declaraciones de Variables: let-in

El lenguaje soporta expresiones locales con let-in, las cuales permiten declarar variables limitadas a un bloque de código.

4 Fase 1: Análisis Léxico (Lexer)

La primera etapa del compilador Hulk es el análisis léxico, cuya responsabilidad es transformar el código fuente en una secuencia de tokens. Estos tokens representan las unidades léxicas fundamentales del lenguaje, como identificadores, palabras clave (let, type, if, etc.), operadores (+, -, =, etc.), literales (números, cadenas) y símbolos de puntuación (, , ;, etc.).

4.1 Implementación

El proceso incluye:

- Motor de expresiones regulares: Permite definir una expresión regular para cada tipo de token
- Cada expresión regular se convierte a un NFA (Autómata Finito No Determinista).
- Todos los NFAs individuales se combinan en un solo NFA global
- Este NFA global se convierte en un DFA (Autómata Finito Determinista) mediante el algoritmo de subconjutos.
- Finalmente, se implementa un scanner eficiente que recorre el texto de entrada usando el DFA, produciendo una secuencia ordenada de tokens con su tipo y posición en el código.

5 Fase 2: Análisis Sintáctico (Parser)

En el compilador del lenguaje Hulk se implementó un parser LL(1) desde cero. Este tipo de parser realiza un análisis sintáctico predictivo de arriba hacia abajo utilizando una tabla de predicciones, lo cual permite una implementación eficiente y sin ambigüedades para gramáticas adecuadas.
La gramática del lenguaje Hulk está definida de forma declarativa en un archivo llamado gramatica.txt ubicado en la raíz del proyecto. Este archivo contiene las reglas de producción de la gramática libre de contexto que describe la sintaxis válida del lenguaje.A partir de esta gramática se construyen automáticamente:

- El conjunto FIRST y FOLLOW de cada no terminal.
- La tabla de análisis LL(1) utilizada para el proceso de parsing.

5.1 Generación del Árbol de Sintaxis

Durante el proceso de análisis, el parser genera inicialmente un árbol de sintaxis concreto (CST, Concrete Syntax Tree) que representa fielmente la estructura de la entrada, incluyendo símbolos intermedios y tokens.
Para la construcción del árbol de sintaxis abstracta (AST) se implementaron funciones específicas que simulan una gramática atributiva, aplicando acciones semánticas al momento de reducir producciones

6 Fase 3: Análisis Semántico

La fase de análisis semántico en el compilador se encarga de validar que el programa fuente, además de tener una estructura sintáctica válida, cumpla con las reglas del lenguaje a nivel lógico y de tipos. Esta etapa se divide en tres subfases principales, ejecutadas en orden para lograr una verificación completa:

6.1 Recolección de Definiciones (Primer Pasada)

En esta primera pasada, se recorre todo el programa para recolectar las definiciones de funciones, independientemente del orden en que aparecen en el código. Esto es necesario porque en Hulk, al igual que en muchos lenguajes modernos, las funciones pueden ser utilizadas antes de ser declaradas.

6.2 Resolución de Símbolos e Inferencia (Segunda Pasada)

Una vez conocidas todas las definiciones, se realiza un recorrido para:

- Resolver todos los identificadores, asegurando que cada símbolo utilizado haya sido previamente declarado.
- Aplicar un sistema de inferencia de tipos básica, especialmente en casos como expresiones sin anotación explícita o el uso de let sin tipo declarado.

6.3 Chequeo Semántico (Tercera Pasada)

En la última pasada se realizan las verificaciones profundas, tales como:

- Compatibilidad de tipos en operaciones aritméticas y lógicas.
- Tipos de retorno en funciones.
- Validez de llamadas a funciones y métodos (aridad y tipos de argumentos).

7 Generación de Código Intermedio (LLVM IR)

La última etapa del compilador de Hulk consiste en la traducción del AST a una representación intermedia (IR) utilizando la infraestructura de LLVM, lo que permite obtener un código portable a múltiples arquitecturas.

Se usa LLVM para generar el código. Cada parte del lenguaje (funciones, clases, expresiones, etc.) se traduce a instrucciones LLVM. El resultado es un archivo .ll que contiene el código LLVM, el cual puede compilarse y ejecutarse con herramientas como make compile y make execute.

8 Conclusiones

El desarrollo del compilador para el lenguaje Hulk ha sido un proyecto integral que abarcó desde la implementación del análisis léxico, pasando por el análisis sintáctico y semántico, hasta la generación de código LLVM listo para ejecución.

Se logró construir un compilador funcional desde cero, diseñando un lexer basado en expresiones regulares, un parser LL(1) que construye árboles sintácticos abstractos, y un sistema de chequeo semántico que verifica la validez del código y permite inferencia de tipos básica.

La generación de código a través de LLVM permitió traducir las estructuras del lenguaje a un código eficiente y portable, incluyendo soporte para características avanzadas como la herencia simple y la programación orientada a objetos.

Este proyecto demuestra la viabilidad de construir un lenguaje con conceptos modernos usando herramientas de código abierto, y sienta las bases para futuras ampliaciones como optimizaciones, manejo avanzado de tipos y características de lenguaje más complejas.

9 Bibliografias

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley. — Conocido como "El Libro del Dragón", es una referencia fundamental en el campo de compiladores.

- LLVM Project. (2025). LLVM Language Reference Manual. Disponible en: llvm.org/docs — Documentación oficial y recurso principal para el uso de LLVM en generación de código.