Entrega 1: FrontEnd para el Proyecto LogoTec

Castro Moreno Henry Andrés
hencastro@estudiantec.cr
Carné: 2022026502

Rivera Mora José Ignacio
j.rivera@estudiantec.cr
Carné: 2022227827

1 de Octubre de 2025

Resumen

Este documento presenta el primer avance del proyecto de desarrollo de un lenguaje de programación tipo Logo con su propio IDE y compilador. Se detalla la arquitectura propuesta, problemas técnicos enfrentados, soluciones implementadas y el estado actual del desarrollo. El proyecto incluye un analizador léxico, sintáctico, generación de AST y un entorno integrado de desarrollo.

1. Introducción

Este proyecto tiene como objetivo diseñar e implementar un lenguaje de programación educativo inspirado en Logo, junto con su entorno de desarrollo integrado (IDE) y compilador. Logo es conocido por su sintaxis simple y su enfoque en la enseñanza de programación mediante gráficos de tortuga. En este avance, se presenta el diseño arquitectónico y los componentes fundamentales del sistema.

2. Diagrama de Arquitectura de la Solución

2.1. Descripción General

Segun indica [1], la arquitectura del sistema LogoTec se organiza en capas funcionales que reflejan el flujo lógico de procesamiento desde la entrada del programa hasta su ejecución gráfica. Aunque sigue el modelo clásico de compiladores, se adapta para incluir validaciones semánticas específicas, gestión de símbolos y evaluación de expresiones. El diseño modular busca facilitar la extensibilidad, la depuración y la integración de nuevas funcionalidades.

2.2. Componentes Principales

- **FrontEnd (Entrada y Validación)**: Punto de entrada del programa, que verifica la estructura inicial del código. Incluye validaciones como la presencia de un comentario en la primera línea y al menos una variable declarada.
- **Gestión de Símbolos y Variables**: Se encarga de validar identificadores, declarar o asignar variables y mantener una tabla de símbolos que almacena el estado de cada variable.
- **Procesamiento de Sentencias**: La regla actúa como núcleo de ejecución, conectando instrucciones como declaraciones de procedimientos, llamadas, bloques de ejecución y repeticiones.
- **Control de Flujo**: Incluye estructuras condicionales y de repetición como `SI`, `MIENTRAS`, `HASTA`, `Haz.Mientras` y `Haz.HASTA`, todas basadas en la evaluación de expresiones lógicas.
- **Evaluación de Expresiones**: Procesa expresiones aritméticas, lógicas y funciones personalizadas. Estas expresiones alimentan decisiones, cálculos y comandos gráficos.
- **BackEnd**: Interpreta comandos gráficos como `AVANZA`, `GIRADERECHA`, `PONPOS`, etc. Gestiona la posición y dirección de la tortuga, mientras que se representan visualmente los resultados.
- **Editor de Código**: Interfaz gráfica que permite al usuario escribir programas en el lenguaje LogoTec, integrando el flujo completo desde la edición hasta la ejecución visual.

2.3. Diagrama de arquitectura

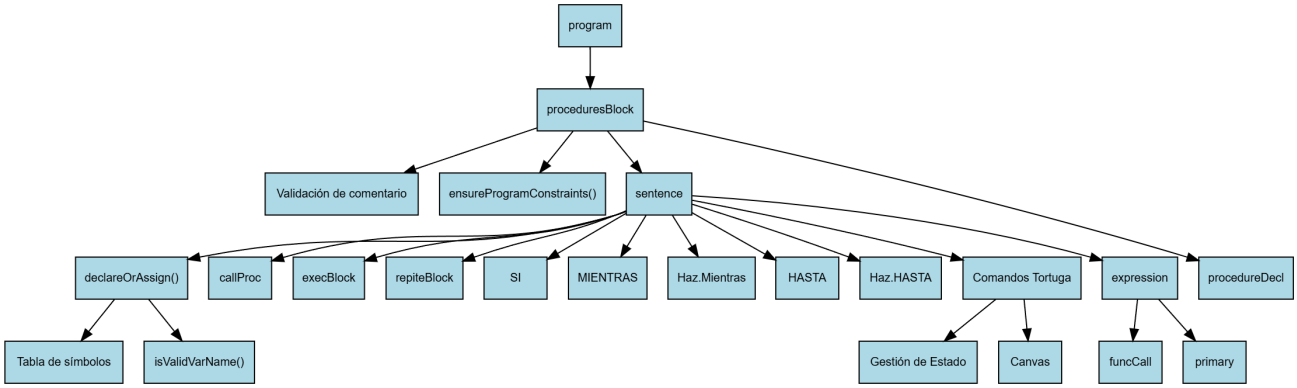

Figura 1: Diagrama de arquitectura del sistema

Nota: El diagrama de la figura 1 esta hecho basandose en la herramienta Graphviz[2] .

3. Problemas Conocidos

3.1. Limitaciones Actuales

1. **Error de versiones**: El error `ANTLR Tool version 1.1 used for code generation does not match the current runtime version 4.5.1. ANTLR Tool version 1.4 used for code` no ha generado obstáculos pero se presenta en cada ejecución.
2. **Comandos anidados**: El parser tiene dificultades con estructuras profundamente anidadas.
3. **Interfaz gráfica**: La interfaz todavía no se ha implementado con el sistema de Antlr.

4. Problemas Encontrados

4.1. Problema 1: Ciclo en archivo de funciones.

4.1.1. Descripción Detallada

Todos tenian el mismo formato, en el archivo del parser se indicaba que eso estaba mal.

4.1.2. Intentos de Solución

- Se trató de cambiar la estructura del ciclo, se valoró usar recursión o un ciclo while, ambos intentos no resultaban en nada útil. Problema de tipo de variables.

4.1.3. Solución Encontrada

Recorrer la lista y conforme lo hace se va añadiendo el contenido de la lista al objeto, antes para ejecutar las operaciones y asi que no tenga problemas en el parser.

4.1.4. Recomendaciones

- Validar frecuentemente con casos de prueba
- **Separación clara de responsabilidades en la gramática**: Mantener una distinción explícita entre reglas de control de flujo, comandos gráficos, expresiones y procedimientos. Esto facilita la extensibilidad y reduce ambigüedades en el parser.

4.1.5. Bibliografía Consultada

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools.

5. Conclusiones y Recomendaciones del Proyecto

5.1. Conclusiones

- El diseño de un lenguaje de programación requiere balance entre simplicidad y expresividad.
- La arquitectura modular facilita el desarrollo incremental y las pruebas.
- La generación automática de componentes (lexer, parser) acelera el desarrollo pero requiere comprensión profunda.

5.2. Recomendaciones

- **Para futuros desarrollos**: Implementar análisis semántico temprano en el proceso
- **Para equipos**: Establecer convenciones de código desde el inicio
- **Para gestión**: Planificar iteraciones cortas con objetivos claros
- **Para documentación**: Mantener documentación actualizada paralelamente al desarrollo

6. Bibliografía Consultada

Referencias

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Addison-Wesley Professional.

[2] Ellson, J., Gansner, E., Koutsofios, L., North, S. C., & Woodhull, G. (2002). *Graphviz—open source graph drawing tools*. Graph Drawing, 483-484.