ñe’ẽasaha - Compilador Guaraní

[a] Melisa Agostina Lezcano Airaldi; [b] Manuela Marder

[c] Osvaldo Agustín Lago; [d] Juan Agustín Acosta

[a] Facultad de Ciencias Exactas y Naturales y Agrimensura,

Teoría de la Computación, melisalezcanoairaldi@gmail.com

[b] Facultad de Ciencias Exactas y Naturales y Agrimensura,

Teoría de la Computación, manumarder@gmail.com

[c] Facultad de Ciencias Exactas y Naturales y Agrimensura,

Teoría de la Computación, agustinlago164@gmail.com

[d] Facultad de Ciencias Exactas y Naturales y Agrimensura,

Teoría de la Computación, juanagustinacost@gmail.com

Resumen

Este trabajo presenta el diseño e implementación de un analizador sintáctico para un lenguaje ficticio educativo basado en palabras simples del idioma guaraní. El objetivo principal fue aplicar los conocimientos teóricos de gramáticas libres de contexto mediante la construcción de un parser **descendente recursivo** que reconozca estructuras básicas como declaraciones, condicionales y repeticiones. A partir de una gramática definida en notación **BNF**, se desarrolló un sistema que incluye análisis léxico, sintáctico y generación de un árbol sintáctico abstracto (AST). Además, se implementaron etapas posteriores como la generación de código intermedio y optimizado, con el fin de representar la estructura lógica del programa. El lenguaje fue diseñado para ser intuitivo y didáctico, promoviendo el uso del guaraní en entornos computacionales. Finalmente, el sistema fue probado con múltiples casos válidos y con errores, verificando su comportamiento. Este proyecto no solo refuerza conceptos clave de la teoría de lenguajes formales, sino que también fomenta la creatividad, el trabajo en equipo y la documentación técnica en un entorno educativo.

1. Introducción

El presente trabajo presenta el desarrollo de un compilador educativo denominado *Compilador Guaraní*, cuyo principal objetivo es enseñar los conceptos fundamentales de compiladores mediante un lenguaje de programación con sintaxis en idioma guaraní. La motivación detrás de este proyecto surge del interés por incorporar elementos culturales y lingüísticos propios de la región de Corrientes, Argentina, en herramientas tecnológicas y pedagógicas.

Si bien el lenguaje no busca ser más eficiente ni competir con lenguajes existentes, su valor radica en la familiaridad lingüística y en el interés que puede despertar en estudiantes locales. Consideramos que este enfoque puede facilitar el aprendizaje inicial de estructuras propias de la programación, a la vez que fortalece vínculos identitarios con el entorno cultural del estudiante.

2. Desarrollo

Para la construcción del compilador "Guaraní", se comenzó con una investigación teórica sobre los analizadores léxicos y

sintácticos, así como su rol dentro del proceso de compilación. En base a esta revisión, se elaboró un esquema de trabajo utilizando herramientas de gestión como *Trello*, con el objetivo de organizar las tareas, establecer prioridades y asignar funciones específicas a cada componente del sistema.

El desarrollo inicial incluyó la implementación del analizador léxico (*lexer*), encargado de transformar el código fuente escrito en guaraní en una secuencia de tokens. Posteriormente, se construyó el analizador sintáctico (*parser*), diseñado para interpretar la estructura gramatical de los tokens producidos por el lexer, y generar un Árbol de Sintaxis Abstracta (AST) que representa jerárquicamente las construcciones del lenguaje.

Durante las primeras pruebas, el sistema mostró un comportamiento correcto en el análisis léxico y sintáctico. El árbol AST generado permitió representar expresiones, declaraciones, estructuras de control y otras construcciones semánticas esenciales.

Una vez consolidado el analizador, se procedió con la implementación de un intérprete, cuya función es recorrer el AST y ejecutar el código representado. En esta etapa, se detectaron errores vinculados al manejo de tipos de datos. Por ejemplo, todas las variables eran interpretadas como cadenas de texto, lo que provocaba resultados incorrectos al realizar operaciones aritméticas (e.g., la suma `7 + 9` devolvía `"79"` en lugar de `16`).

Otro problema importante surgió a partir del orden de definición de los tokens en las expresiones regulares del lexer. Operadores compuestos como `<=` eran reconocidos erróneamente como dos tokens separados (`<` y `=`), debido a que los tokens individuales estaban definidos antes que los compuestos. Este tipo de conflicto fue resuelto reordenando las expresiones regulares para priorizar los tokens de mayor longitud, siguiendo principios comunes en la construcción de analizadores léxicos.

Finalmente, se desarrolló una interfaz gráfica en Next.js que permite escribir código en guaraní, visualizar los tokens generados, examinar el árbol AST y observar la salida producida por el intérprete. Esto convierte al compilador en una herramienta educativa interactiva que permite comprender el proceso de interpretación de un lenguaje desde su definición gramatical hasta su ejecución

3. Conclusión

El desarrollo de este compilador fue una experiencia formativa y enriquecedora. A lo largo del trabajo se evidenció cómo aspectos que podrían parecer menores, como el orden en que se definen los patrones del lexer, pueden afectar de forma crítica el comportamiento del sistema.

Además, se observó que la correcta definición de cada módulo —lexer, parser, AST e intérprete— es fundamental para garantizar el funcionamiento global del compilador. El proceso resultó desafiante, ya que implicó prestar atención a numerosos detalles técnicos y conceptuales, pero también fue muy interesante al revelar la complejidad detrás de un sistema que, a simple vista, podría parecer sencillo.

Este proyecto permitió comprender en profundidad la naturaleza meticulosa de los compiladores, y cómo cada etapa debe funcionar con precisión para que el todo opere de forma coherente y confiable.

Bibliografía

[1] A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman, *Compiladores: Principios, técnicas y herramientas*, 2.ª ed., Pearson, 2008.

[2] D. Crockford, *JavaScript: The Good Parts*, 1.ª ed., O'Reilly Media, 2008.

[3] Next.js, "The React Framework for the Web," [En línea]. Disponible en: https://nextjs.org. [Último acceso: 14-jun-2025].

[4] E. Gamma, R. Helm, R. Johnson y J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[5] TypeScript, "TypeScript: Typed JavaScript at Any Scale," [En línea]. Disponible en: https://www.typescriptlang.org. [Último acceso: 06-2025].

[6] Mozilla Developer Network (MDN), "Regular Expressions," [En línea]. Disponible en: https://developer.mozilla.org/en-

US/docs/Web/JavaScript/Guide/Regular_Expressions. [Último acceso: 06-2025].

[7] Y. Bengio, I. Goodfellow y A. Courville, *Deep Learning*, MIT Press, 2016. [Disponible en línea]: https://www.deeplearningbook.org. [Último acceso: 06-2025].

[8] OpenAI, *ChatGPT: language model for conversation and reasoning*, [En línea]. Disponible en: https://chat.openai.com. [Último acceso: 06-2025].