Visualización del análisis léxico para enseñanza de Compiladores Multilenguaje

Soncco Pacco Ivan Ulises, Quispe Huancayo Joselyn Lizeth

isoncco@unsa.edu.pe
jquispe@unsa.edu.pe

***Abstract*—**

I. Estado del arte

Los autores Swetchana Dutta et. al.(2024) abordan en su trabajo *"Compiler Design for Recognizing Different Programming Languages",* las fases clave del diseño de compiladores —análisis léxico, parsing sintáctico, validación semántica, generación de código intermedio, optimización y generación final de código— destacando el **análisis léxico**, donde el código fuente se descompone en componentes como palabras clave, identificadores, literales y operadores. Aprovechando la biblioteca **Python-Lex-Yacc (PLY)**, el estudio se centra en la implementación de un compilador capaz de **reconocer múltiples lenguajes de programación**, incluyendo C++, Java y Python simultáneamente, dada la naturaleza heterogénea de las aplicaciones modernas. El método propuesto configura reglas gramaticales adaptadas a cada lenguaje, generando automáticamente tokens mediante PLY en una fase lingüística altamente específica para cada sintaxis. Los resultados reportados demuestran que el compilador diseñado cumple eficazmente con el reconocimiento concurrente de varios lenguajes, cumpliendo con los objetivos iniciales de robustez y flexibilidad {since performance metrics are not given in the abstract}.

Los autores Muthukumar et. al. (2024) proponen un sistema que analiza archivos de código en cualquiera de estos lenguajes, clasifica los tokens, los almacena en una tabla de símbolos y muestra dicha tabla. También detecta y reporta cualquier error de sintaxis encontrado.

Farhanaaz, Sanju, V. (2016) proponen una exploración y discusión sobre el análisis léxico como la primera fase crucial en el proceso de compilación. El trabajo se centra en el funcionamiento de los analizadores léxicos, su capacidad para transformar el código fuente en tokens significativos, y su rol en la gestión de errores y tablas de símbolos. El método que proponen para la construcción de analizadores léxicos implica la definición de reglas del dominio del problema (como palabras reservadas, identificadores válidos) y la traducción de este modelo léxico a Autómatas Finitos Deterministas (DFA) o No Deterministas (NFA). La técnica principal empleada es el uso de Expresiones Regulares para la especificación de tokens y Autómatas Finitos (DFA y NFA) para su reconocimiento e implementación. Los resultados implícitos de la propuesta son una comprensión clara de cómo se puede diseñar e implementar un analizador léxico , incluyendo la gestión de tokens, la identificación de patrones y la integración con las siguientes fases del compilador.

Upadhyaya (2011) propone una exploración exhaustiva sobre el análisis léxico, enfocándose en su papel como la primera fase crítica de la compilación, donde el código fuente se convierte en una secuencia de tokens. El método que proponen para la construcción de analizadores léxicos implica definir y describir las reglas del dominio del problema (como palabras reservadas e identificadores) y luego traducir este modelo léxico a Autómatas Finitos Deterministas (DFA) o No Deterministas (NFA). Una técnica clave que proponen y detallan es el uso de Expresiones Regulares para la especificación de tokens y los Autómatas Finitos (DFA y NFA) para su reconocimiento eficiente, incluyendo el proceso de conversión de NFA a DFA mediante la construcción de subconjuntos. Los resultados de esta exploración no son resultados experimentales de una implementación específica, sino una clarificación y propuesta de los principios y mecanismos fundamentales del análisis léxico.

Barve et. al.(2014) el desarrollo de una estrategia para el análisis léxico paralelo, aprovechando las capacidades de las máquinas multinúcleo. Su objetivo es mejorar la eficiencia de la fase de análisis léxico al permitir el procesamiento simultáneo de múltiples archivos fuente. El método que proponen para lograr esto es a través de un algoritmo de análisis léxico paralelo diseñado específicamente para operar en entornos multinúcleo. La técnica principal subyacente es la de explotar el paralelismo inherente de la arquitectura de múltiples núcleos para realizar análisis léxico de más de un archivo fuente de forma concurrente. Como resultados, el trabajo busca demostrar una mejora en el rendimiento del análisis léxico, lo que se traduce en una mayor velocidad de procesamiento de los archivos fuente y, en consecuencia, en una reducción del tiempo total de compilación en sistemas con múltiples núcleos.

REFERENCIAS

[1] C. R. A. N. Sharma, B. M. Reddy, D. Swetchana and N. Panda, "Compiler Design for recognizing different Programming Languages," 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10724115.

[2] S. S. Muthukumar, B. Gandham, T. Vijayekkumaran, N. S. Krishna and N. Panda, "Implementing a Multilingual Lexical Analyser for Java, C and C++," 2024 5th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2024, pp. 1-9, doi: 10.1109/GCAT62922.2024.10923959.

[3] Farhanaaz, Sanju, V. (2016). An Exploration on Lexical Analysis. Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), 253-258. DOI: https://doi.org/10.1109/ICEEOT.2016.7755127.

[4] Upadhyaya, M. (2011, April). Simple calculator compiler using Lex and YACC. In 2011 3rd International Conference on Electronics Computer Technology (Vol. 6, pp. 182-187). IEEE.

[5] Barve, A., & Joshi, B. K. (2014). Parallel lexical analysis of multiple files on multi-core machines. International Journal of Computer Applications, 96(16).