UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CAMPUS CHAPECÓ

CURSO DE CIÊNCIA DA COMPUTAÇÃO

COMPONENTE CURRICULAR DE CONSTRUÇÃO DE COMPILADORES

Prof. Dr. BRAULIO ADRIANO DE MELLO

RIAN BORGES BARBOSA

ANALISADOR LÉXICO

CHAPECÓ

2025

Sumário

1 Resumo

O objetivo deste trabalho foi construir um analisador léxico para uma linguagem de programação simples, capaz de identificar tokens como variáveis, palavras-chave, operadores e números. A implementação utiliza um autômato finito determinístico (AFD) para determinar a sequência válida de tokens e classificá-los de acordo com as regras pré-definidas.

O analisador gera uma tabela de símbolos conforme o analisador lê o código. Ele também sinaliza quaisquer erros léxicos que descobre durante o processo, que são essencialmente erros de digitação ou formatação de código.

2 Introdução

A análise léxica é a primeira etapa do processo de compilação, responsável por converter a sequência de caracteres de um programa em tokens significativos para fases subsequentes, e reconhecedores léxicos são componentes fundamentais na construção de compiladores.

O presente trabalho tem como objetivo implementar um reconhecedor léxico baseado em um AFD. O sistema divide o código-fonte em tokens individuais, classifica o token de acordo com o seu tipo, registra os tokens em uma tabela de símbolos, verifica a validade da sequência de tokens usando um AFD e identifica e reporta erros.

3 Referencial Teórico

A análise léxica é uma etapa importante na etapa de um compilador, pois ela transforma código-fonte em tokens. Estes tokens são elementos básicos que o compilador reconhece, como variáveis, operadores e números.

Os tokens combinam o lexema (sequência de caracteres) com sua classificação. Por exemplo, "count = 5", temos três tokens distintos: "count"(uma variável), "="(um operador de atribuição) e "5"(um valor numérico).

Para realizar o reconhecimento e classificação, geralmente é usado autômatos finitos. Dentre estes, os Autômatos Finitos Determinísticos (AFDs) destacam-se pela sua

eficiência computacional. O AFD realiza o processamento de caracteres de entrada sequencialmente, transitando entre estados conforme encontra diferentes símbolos, permitindo determinar com precisão onde cada token começa e termina. Segundo [1], o AFD é um tuplo (Q, Σ, δ, q_I, F) onde:

- O estado de controle Q é um conjunto finito.
- O alfabeto de entrada Σ é um conjunto finito de símbolos.
- A transição $\delta : Q \times \Sigma \to Q$ é uma função.
- O estado inicial q_I pertence a Q.
- O conjunto de estados finais F é um subconjunto de Q.

4 Implementação e resultado

A construção do AFD no projeto foi feita manualmente no arquivo *afd.py*.

4.1 Tokens

Os tokens definidos para a linguagem foram **count**, **=**, **0**, **while**, **:**, **<**, **5**, **+**, **1**, sendo:

- Variável: *count*
- Palavra-chave: *while*, *=*, *:*
- Operador: *<*, *+*
- Número: *0*, *1*, *5*
- E qualquer palavra não reconhecida é inválida e gera um erro.

O AFD contém as seguintes transições para cada estado:

- De q0 a q5: reconhece a palavra `count`
- De q5 a q7: reconhece a atribuição `count = 0`
- De q7 a q12: reconhece a palavra `while`

- De q12 a q17: reconhece a segunda ocorrência de `count`
- De q17 a q19: reconhece a condição `count < 5`
- De q19 a q20: reconhece o delimitador `:`
- De q20 a q25: reconhece a terceira ocorrência de `count`
- De q25 a q28: reconhece o incremento `count += 1`

Os estados finais {q5, q7, q12, q17, q19, q20, q25, q28} representam pontos significativos da expressão reconhecidos pelo autômato.

4.2 Implementação

Foi usada a linguagem Python. Foi determinado um dicionário para o alfabeto, transições e estados finais. O programa realiza as seguintes ações:

- Lê um arquivo *input.in*, contendo as palavras com quebras de linha.
- Tokeniza o código de entrada, separando operadores, variáveis e palavras-chave.
- Classifica cada token em categorias (VARIABLE, KEYWORD, OPERATOR, NUMBER).
- Realiza transições entre estados do AFD para cada caractere lido.
- Gera uma tabela de símbolos (contendo linha, identificador e rótulo), e reporta erros encontrados durante a análise.

4.3 Resultados Finais

Por meio de diversos testes, o analisador mostrou-se capaz de reconhecer corretamente todos os tokens válidos presentes no código-fonte. Quando encontra um token inválido, ele emite uma mensagem de erro indicando o lexema problemático e a respectiva linha, o que facilita a localização e correção de inconsistências no programa.

5 Conclusão

Este trabalho apresentou a implementação de um analisador léxico baseado em AFD, capaz de identificar e classificar tokens de forma precisa, além de construir uma tabela de símbolos e reportar erros léxicos com indicação de linha.

Uma das principais dificuldades enfrentadas foi gerenciar a iteração de transição para cada caractere de entrada, exigindo atenção a detalhes de estados e condições de parada.

Apesar desse desafio, o analisador mostrou-se robusto em diferentes cenários de teste. Como trabalho futuro, sugere-se estender o suporte a expressões regulares mais complexas e integrar um parser sintático para compor um compilador completo.

Referências

[1] Faculdade de Ciências da Universidade de Évora. *Autómatos Finitos Deterministas*. Acessado em: 27 de abril de 2025. URL: https://home.uevora.pt/~fc/alp/02-automatos_finitos/02.01-afd.html (acesso em 27/04/2025).