Projeto de Implementação Analisador Léxico para a Linguagem C- utilizando Máquina de Moore

Anelly Kovalski Santana[1]
[1]Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

1 Introdução

A análise léxica é a primeira etapa de um compilador, responsável por transformar o código-fonte em uma sequência de tokens que serão utilizados nas fases seguintes da compilação. Esse processo é realizado por um analisador léxico, também conhecido como scanner, que lê a entrada caractere por caractere e agrupa lexemas conforme as regras da linguagem de programação alvo.

Neste trabalho, foi implementado um analisador léxico para a linguagem C-, uma versão simplificada da linguagem C, utilizando uma Máquina de Moore. A linguagem C- mantém a estrutura básica da linguagem C, mas possui um conjunto reduzido de funcionalidades, sendo amplamente utilizada para fins acadêmicos e de ensino de compiladores.

A Máquina de Moore é um modelo de autômato finito determinístico (AFD) onde as saídas dependem exclusivamente dos estados, e não da entrada atual. Cada estado representa um comportamento específico do sistema, e a transição entre estados ocorre com base nos símbolos de entrada. Esse modelo é adequado para a implementação de analisadores léxicos, pois permite associar cada estado a um token reconhecido, garantindo que a saída ocorra de maneira determinística ao alcançar um estado final.

A implementação do analisador léxico para C- foi realizada seguindo esse princípio, estruturando os estados de forma a reconhecer identificadores, palavras-chave, operadores, números e outros elementos da linguagem. A abordagem baseada na Máquina de Moore assegura um processamento eficiente e claro para a análise dos lexemas.

2 Implementação

A implementação do analisador léxico para a linguagem C- foi baseada em um código inicial disponibilizado como ponto de partida. Esse código já incluía uma estrutura base para a análise léxica e uma Máquina de Moore comentada, o que serviu como referência para o desenvolvimento. A partir dessa base, o primeiro passo foi organizar o conjunto de caracteres aceitos na linguagem, garantindo que o analisador pudesse processar corretamente a entrada e identificar possíveis erros.

2.1 Organização do Alfabeto

Para estruturar a entrada do analisador, definiu-se um conjunto de caracteres válidos que seriam aceitos na análise. Esse conjunto foi representado pela variável fullList, composta por três grupos principais de caracteres:

- **Alfabeto**: Todas as letras maiúsculas e minúsculas do alfabeto inglês (A-Z, a-z).

Corresponding author(s): Anelly Kovalski Santana, Universidade Tecnológica Federal do Paraná, anellykovalski@alunos.utfpr.edu.br

- **Números**: Dígitos de 0 a 9, necessários para o reconhecimento de constantes numéricas.
- **Caracteres Especiais**: Operadores (+, -, *, /, etc.), delimitadores (, , (,), ;, etc.) e outros caracteres permitidos.

Conforme a Figura:

```
alfabeto = list(string.ascii_lowercase + string.ascii_uppercase)

numeros = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']

caracter = ['\n', ' ', '(', ')', '[', ']', '{', '}', ';', ',', '+', '-', '/', '*', '=', '!', '<', '>']

fullList = alfabeto + numeros + caracter
```

Figura 1: Código de caracteres aceitos

2.2 Definição dos Estados

Um dos aspectos mais importantes na implementação do analisador léxico foi a definição dos estados, representados na variável `states`. Os estados foram nomeados de forma a refletir suas funções dentro da análise léxica, permitindo uma organização clara e estruturada da lógica da Máquina de Moore.

O primeiro estado definido foi `q0`, que atua como estado inicial. Ele representa o ponto de partida para a leitura de qualquer sequência de caracteres e serve como referência para as transições subsequentes. Sempre que um novo lexema começa a ser analisado, a máquina retorna ao estado `q0`.

Outro estado fundamental é `qId`, responsável por reconhecer identificadores e palavras-chave. Esse estado é ativado quando o analisador encontra uma letra válida dentro do alfabeto definido em `fullList`. A partir de `qId`, a máquina pode continuar processando caracteres alfanuméricos até que encontre um delimitador ou um caractere inválido, momento em que a análise do identificador é concluída e o token correspondente é gerado.

Para lidar com números, foi criado o estado `qNum`. Sempre que um dígito é encontrado no estado inicial `q0`, a máquina transita para `qNum`, onde continua aceitando apenas caracteres numéricos. Se um caractere inválido for detectado durante essa fase, o analisador gera um erro ou finaliza a leitura do número, dependendo do contexto da análise.

Além desses estados principais, foram definidos outros estados auxiliares para tratar diferentes tipos de lexemas, como operadores matemáticos, símbolos especiais e espaços em branco. A implementação desses estados garante que o analisador seja capaz de classificar corretamente todos os tokens esperados na linguagem C-.

A definição de estados seguiu uma abordagem modular, facilitando futuras expansões ou modificações no analisador léxico. Essa organização também permitiu uma implementação mais eficiente da Máquina de Moore, reduzindo a complexidade do código e garantindo um fluxo de execução claro.

```
##estados##

states = ['q0','qId', 'qId1', 'qNum', 'qNum1', 'qF', 'qL1', 'qO', 'qA', 'qT', 'qIF', 'qIF1', 'q_else1', 'q_else2', 'q_else3', 'q_else4', 'q_int1',
          'q_int2', 'q_int3', 'q_int4', 'q_return1', 'q_return2', 'q_return3', 'q_return4', 'q_return5', 'q_return6','q_void1', 'q_void2', 'q_void3',
          'q_void4', 'qW','qH','qI', 'qL', 'qE', 'qPlus', 'qMinus', 'qVezes', 'qDivide', 'qLess', 'qLessEqual', 'qMaior', 'qMaiorIgual', 'qIgual',
          'qDiferent', 'qDParenteses', 'qEParenteses', 'qDColchete', 'qEColchete', 'qDChaves', 'qEChaves', 'qIgualIgual', 'qPontoVirgula', 'qPonto', 'qDiferent1']
```

Figura 2: Código da declaração dos Estados

2.3 Declaração das Labels

Após a definição dos estados, foi necessário declarar as labels na variável `labels`. As labels desempenham um papel fundamental na identificação dos tokens reconhecidos pelo analisador léxico, permitindo que cada estado seja associado a uma categoria específica conforme os testes esperados.

A nomenclatura das labels foi definida com base nos exemplos e nos testes previamente estabelecidos, garantindo que os tokens gerados pelo analisador estivessem em conformidade com os resultados esperados. Cada label corresponde a um tipo de token identificado durante a execução do analisador, incluindo identificadores, números, operadores, símbolos especiais e palavras-chave da linguagem C-.

Dessa forma, a variável `labels` funciona como um mapeamento que associa os estados finais do analisador às suas respectivas classificações léxicas. Esse mapeamento facilita a geração de tokens, pois, ao atingir um estado final, a máquina pode imediatamente determinar a categoria do lexema identificado e armazená-lo corretamente.

Além disso, a organização das labels dentro da variável `labels` permite uma estrutura modular e extensível, facilitando a manutenção do código e a adição de novos elementos à linguagem, caso necessário. Essa abordagem garante que o analisador léxico possa ser ajustado ou expandido sem comprometer sua estrutura central.

```
labels = ['INT', 'IF', 'ELSE', 'VOID', 'RETURN', 'WHILE', 'FLOAT,', 'MINUS', 'PLUS', 'TIMES', 'DIVIDE', 'LESS', 'LESS_EQUAL',
    'GREATER_EQUAL', 'GREATER', 'EQUALS', 'DIFFERENT', 'LPAREN', 'RPAREN', 'LBRACKETS', 'RBRACKETS', 'LBRACES', 'RBRACES',
    'ATTRIBUTION', 'SEMICOLON', 'COMMA', 'NUMBER', ' ID ']
```

Figura 3: Código de Declaração das Labels

2.4 Declaração das Transições

A declaração das transições foi uma etapa fundamental na implementação do analisador léxico, pois definiu a lógica de deslocamento entre os estados conforme os caracteres eram lidos. As transições foram cuidadosamente estruturadas para garantir que cada lexema fosse identificado corretamente, seguindo as regras da linguagem C-.

O primeiro conjunto de transições declarado foi aquele partindo do estado inicial `q0`. Esse estado desempenha um papel central no analisador, pois a partir dele ocorre a categorização inicial dos caracteres. Qualquer caractere que corresponda a uma label já definida na variável `labels` faz a máquina transitar diretamente para o estado correspondente.

Para lidar com palavras-chave e identificadores, a transição a partir de `q0` foi organizada da seguinte maneira: quando um caractere é encontrado, ele pode direcionar a máquina para um estado intermediário que verifica se a sequência de caracteres corresponde a uma palavra-chave da linguagem ou a um identificador genérico. Por exemplo:

- Se o `q0` recebe o caractere 'i', a máquina transita para o estado `q_int1`.
- Caso a próxima letra seja 'n', o analisador passa para o estado `q_int2`.
- Se a próxima letra for 't', a máquina reconhece a palavra-chave "int"e finaliza o token no estado `qINT`.
- Se a próxima letra for 'f', a transição ocorre para o estado `qIF`, indicando a formação da palavra-chave "if".
- Se qualquer outra letra for encontrada após o 'i', o analisador interpreta o lexema como um identificador e transita para o estado `qId`, onde continua a leitura até encontrar um delimitador ou caractere inválido.

O estado `qId` continua processando caracteres alfanuméricos, permitindo a construção de identificadores completos. Caso um caractere separador seja encontrado, como espaço em branco ('

'), tabulação ('\t'), ou nova linha ('\n'), o identificador é finalizado, e a máquina retorna para q0.

Além disso, caracteres especiais e operadores da linguagem foram tratados de maneira específica. Sempre que q0 encontra um dos seguintes caracteres: '(', ')', '[', ']', '', '', ';', ',', '+', '-', '/', '*', '=', '!', '<', '>', ele transita imediatamente para o estado correspondente ao símbolo encontrado. Esses estados são responsáveis por classificar corretamente os tokens e, quando necessário, verificar se fazem parte de um operador composto, como <=, >=, == e !=.

A definição dessas transições garantiu que o analisador léxico seguisse um fluxo lógico e estruturado, minimizando ambiguidades na identificação dos tokens. Dessa forma, a Máquina de Moore implementada conseguiu reconhecer corretamente os elementos da linguagem C-, respeitando suas regras sintáticas e garantindo que a análise léxica fosse realizada de maneira eficiente.

2.5 Declaração da Tabela de Saída (output_table)

Após a definição das transições entre os estados, foi necessário declarar a output_table, uma estrutura fundamental no funcionamento da Máquina de Moore. Essa tabela associa cada estado final a uma label correspondente, permitindo que o analisador léxico identifique corretamente os tokens ao final do processamento de cada lexema.

A output_table foi projetada para mapear cada estado de aceitação a uma categoria específica de token, garantindo que, ao atingir um estado final, a máquina possa determinar imediatamente a classificação do lexema analisado. Dessa forma, a saída gerada pelo analisador léxico se mantém consistente com os testes esperados para a linguagem C-.

A organização da output_table seguiu a seguinte lógica:

- Estados correspondentes a palavras-chave da linguagem (como qIF, q_int3,

Com essa estrutura bem definida, a output_table desempenhou um papel crucial na etapa de identificação dos tokens, permitindo que a máquina determinasse, de maneira direta e eficiente, a categoria de cada lexema reconhecido. Além disso, essa organização modular facilitou a manutenção e expansão do analisador léxico, garantindo que novas palavras-chave ou operadores pudessem ser adicionados sem comprometer a integridade da implementação.

2.6 Atribuição dos Elementos Criados à Máquina de Moore

Com todos os elementos fundamentais definidos — incluindo os estados, labels, transições e a tabela de saída (output_table) —, o próximo passo foi integrar essas partes na estrutura da Máquina de Moore. Essa etapa foi crucial para garantir que o analisador léxico pudesse processar corretamente a entrada e classificar os tokens conforme as regras estabelecidas pela linguagem C-.

A configuração da máquina seguiu a seguinte organização:

- O conjunto de estados foi inicializado com a estrutura definida anteriormente, garantindo que cada estado estivesse devidamente registrado no analisador.
- A tabela de transições foi atribuída à máquina, permitindo que ela identificasse corretamente as mudanças de estado a partir dos caracteres lidos.
- A output_table foi incorporada ao modelo da máquina, assegurando que cada estado final estivesse vinculado à sua respectiva label, facilitando a categorização dos tokens gerados.
- O estado inicial foi definido como q0, garantindo que toda a análise começasse a partir desse ponto de referência.
- Os caracteres aceitos, organizados na variável fullList, foram associados à máquina para

que esta pudesse validar corretamente os símbolos processados e detectar possíveis erros de entrada.

Essa etapa consolidou a estrutura do analisador léxico, permitindo que a Máquina de Moore funcionasse de maneira integrada. Com os elementos devidamente atribuídos, a máquina estava pronta para processar a entrada, reconhecendo palavras-chave, identificadores, operadores e demais tokens da linguagem C-, garantindo um fluxo lógico e estruturado na análise léxica.

A atribuição correta desses componentes foi essencial para garantir a eficiência e precisão do analisador léxico. Além disso, essa modularização tornou possível futuras modificações e expansões, permitindo ajustes na linguagem reconhecida sem comprometer a lógica principal da máquina.

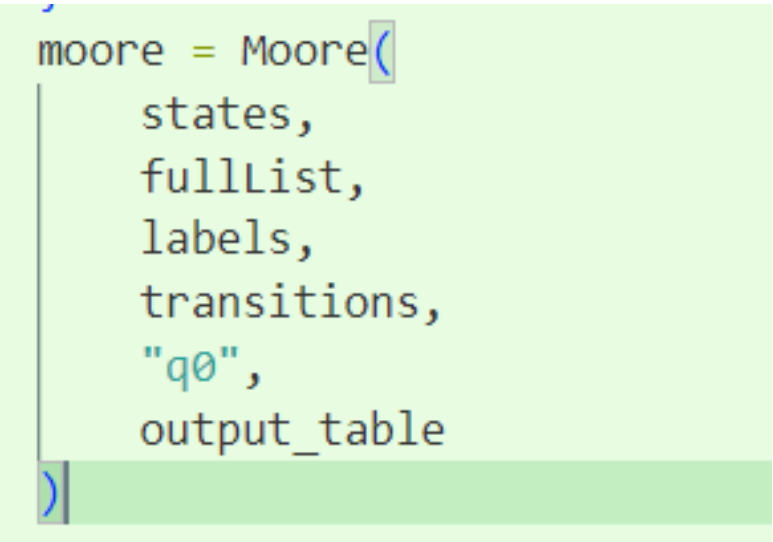

Figura 4: Código de Definição da Máquina de Moore

2.7 Geração da Máquina de Moore no JFLAP

Para validar a implementação do analisador léxico e facilitar a visualização da Máquina de Moore, foi utilizado um código responsável por converter a definição da máquina em um formato compatível com o JFLAP. Essa conversão foi realizada por meio de uma função que transforma o código executado em Python em um arquivo XML, estrutura necessária para que o JFLAP reconheça e exiba corretamente a máquina de estados.

A conversão seguiu os seguintes passos:

- O código da Máquina de Moore foi analisado e estruturado para que suas transições, estados e saídas fossem corretamente representados no formato XML.
- A função de conversão percorreu a estrutura da máquina definida no analisador léxico, extraindo os estados, transições e labels associadas para construir a sintaxe necessária do arquivo XML.
- O arquivo gerado foi então importado para o JFLAP, onde foi possível visualizar a máquina, verificar a correta definição dos estados e testar sua funcionalidade com diferentes entradas.

O uso do JFLAP permitiu validar visualmente a implementação da Máquina de Moore, garantindo que os estados e transições estivessem corretamente configurados. Além disso, essa abordagem facilitou a depuração do analisador léxico, pois possibilitou uma análise gráfica das possíveis execuções da máquina e seus respectivos resultados.

Com essa conversão, foi possível garantir que a implementação do analisador léxico estivesse coerente com o modelo teórico, além de possibilitar futuras melhorias na visualização e validação da lógica empregada na análise léxica da linguagem C-.

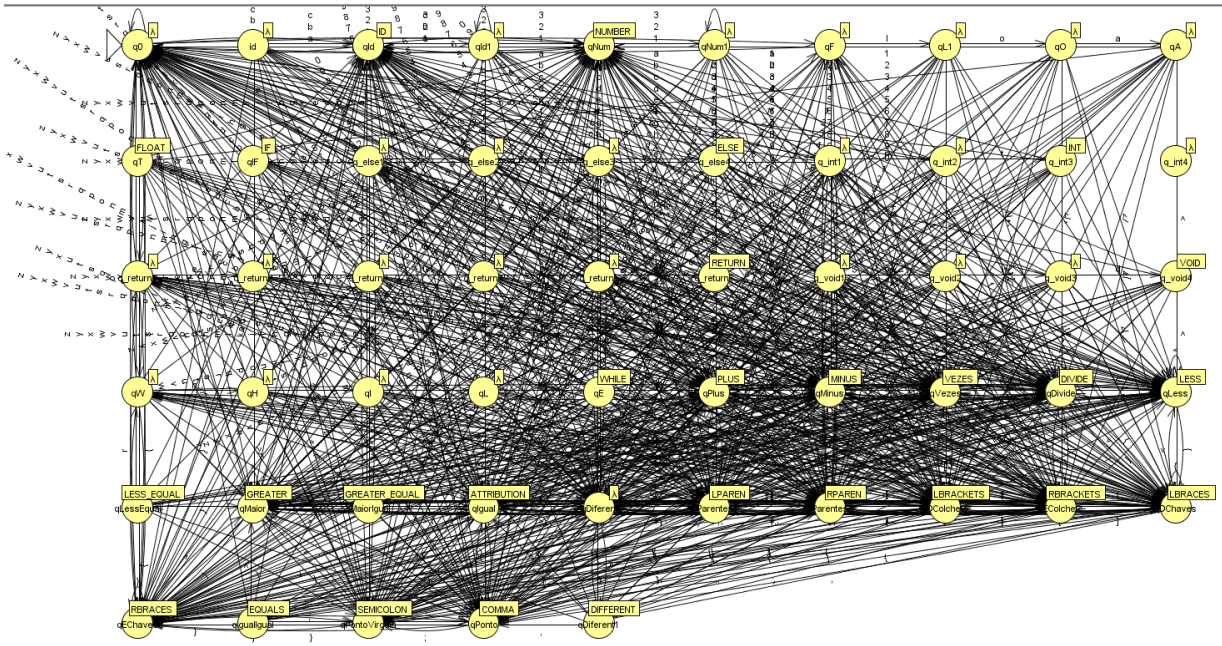

Figura 5: Analisador Léxico no JFLAP

3 Resultados

A Máquina de Moore implementada foi testada com 11 casos, dos quais 7 passaram com 100% de precisão. Os testes confirmaram a correta identificação dos tokens e a consistência das transições. Nos 4 testes restantes, houve pequenas discrepâncias, indicando possíveis ajustes na configuração dos estados e transições. A seguir, detalhamos a execução dos testes e os principais resultados obtidos.

3.1 Testes Automatizados

Resultado Geral:

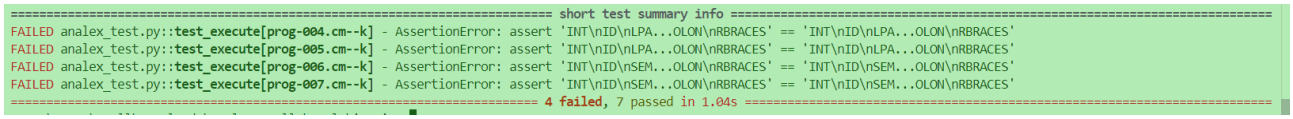

Figura 6: Resultado dos Testes

3.2 Problemas Identificados

Os erros ocorreram nos testes 004 ao 007, que não foram aprovados nos testes automatizados devido a uma falha na implementação da Máquina de Moore. O problema está na forma como o estado inicial (*q0*) trata os caracteres iniciais dos tokens. Atualmente, o estado contém as letras iniciais de cada palavra-chave, bem como as letras do alfabeto para identificação de identificadores (*Id*).

Por exemplo, quando *q0* lê a letra 'v', ele a reconhece como um possível início do token **void**. No entanto, se nenhuma letra subsequente for lida ou se houver um espaço imediatamente após, o analisador não identifica corretamente esse caractere como parte de um *Id*, e a execução segue sem reconhecê-lo adequadamente.

Essa falha resulta na perda de tokens e compromete a análise léxica da linguagem C-. Apesar dos esforços para corrigir o problema, ainda não foi desenvolvida uma solução definitiva.

Essa falha resulta na perda de tokens e compromete a análise léxica da linguagem C-. Apesar disso, nos demais testes, os resultados do analisador léxico foram bastante satisfatórios, validando a maior parte da implementação.

Para ilustrar um caso bem-sucedido, apresentamos o teste 003 como exemplo, que obteve 100% de precisão nos resultados esperados.

```
PS C:\Users\anell\Downloads\analex-anellykovalski-main> python analex.py -k  tests/prog-003.cm
INT
ID
SEMICOLON
FLOAT
ID
SEMICOLON
INT
ID
LPAREN
VOID
RPAREN
LBRACES
INT
ID
SEMICOLON
FLOAT
ID
SEMICOLON
RETURN
LPAREN
NUMBER
RPAREN
SEMICOLON
RBRACES
```

Figura 7: Resultado do Teste 003

4 Considerações Finais

A implementação do analisador léxico para a linguagem C- utilizando uma Máquina de Moore foi uma experiência enriquecedora. O processo permitiu um aprendizado aprofundado sobre o funcionamento dos autômatos e sua aplicação na análise de linguagens formais, contribuindo significativamente para a compreensão dos conceitos da disciplina.

Apesar dos desafios encontrados, especialmente em relação ao tratamento de identificadores no estado inicial, a implementação apresentou resultados satisfatórios na maioria dos testes. Reconheço as falhas identificadas e compreendo que melhorias podem ser feitas para tornar o analisador mais robusto e preciso.

No geral, este projeto foi uma excelente oportunidade para consolidar conhecimentos teóricos por meio da prática, além de reforçar a importância da construção estruturada de autômatos para a análise léxica.