Universidade de São Paulo - USP

Escola de Engenharia de São Carlos - EESC

Instituto de Ciências Matemáticas e de Computação - ICMC

ENGENHARIA DE COMPUTAÇÃO

SCC0605 - Teoria da Computação e Compiladores (2025)

Trabalho 1 - Análise Léxica da Linguagem PL/0

Prof. Thiago A. S. Pardo

Daniel Dias Silva Filho - Nº USP: 13677114
João Victor de Almeida - Nº USP: 13695424
Lucas Corlete Alves de Melo - Nº USP: 13676461
Manoel Thomaz Gama da Silva Neto- Nº USP: 13676392

29 de Abril de 2025

São Carlos - SP

1 - Introdução

O presente trabalho tem como objetivo o desenvolvimento de um analisador léxico para a linguagem PL/0. Esta etapa corresponde à primeira fase da construção de um compilador, sendo responsável por transformar um programa-fonte em uma sequência estruturada de tokens. Para isso, foram projetados autômatos finitos que reconhecem os principais elementos da linguagem, e o sistema foi implementado na linguagem de programação C. O trabalho é composto por este relatório, os códigos-fonte do analisador e um conjunto de imagens ilustrativas dos autômatos utilizados.

2 - Decisões do projeto

O desenvolvimento do projeto foi guiado por decisões que priorizam a organização e eficiência. Para estruturar o reconhecimento dos tokens da linguagem PL/0, optou-se por dividir o processo em cinco sub-autômatos independentes: comentários, identificadores, números inteiros, operadores relacionais e símbolos unitários. Cada autômato foi modelado individualmente, decisão tomada após sequência de erros de lógica e código.

Na implementação em C, foi adotado o uso de uma tabela de transições explícita para representar os autômatos. Essa abordagem permitiu localizar rapidamente o próximo estado a partir do estado atual e do caractere de entrada. Além disso, uma tabela hash foi utilizada para armazenar palavras reservadas e operadores, o que resultou em buscas rápidas durante a classificação dos tokens. Durante o processamento do código-fonte, a estratégia de "lookahead" foi incorporada para lidar com ambiguidades léxicas, possibilitando a leitura de caracteres adicionais sem comprometer a integridade dos tokens já reconhecidos. Também foi implementado um buffer para a construção de cada token à medida que os caracteres são lidos.

Outro ponto importante foi o tratamento de erros: o analisador é capaz de identificar caracteres inválidos, ultrapassagem do tamanho máximo de token e finalização inesperada de comentários. Sempre que ocorre um erro léxico, o sistema gera uma mensagem para o usuário. Por fim, optou-se por utilizar um bloco switch-case para gerenciar as ações dos autômatos, evitando o uso de estruturas mais complexas como ponteiros de função, que, embora possíveis, adicionam complexidade desnecessária para os objetivos do projeto.

3 - Autômatos

3.1 - Autômato de símbolos unitários

Na linguagem PL/0, alguns caracteres são tratados isoladamente como símbolos unitários, desempenhando papeis em expressões, operações e estruturas de controle. Esses símbolos incluem operadores aritméticos, delimitadores e o marcador de fim de programa. Para reconhecê-los corretamente, foi projetado um autômato específico para essa categoria.

O autômato de símbolos unitários é simples e direto: cada símbolo aceito representa uma transição imediata do estado inicial para um estado final correspondente. Entre os símbolos reconhecidos estão:

- Delimitadores: , (vírgula), ; (ponto e vírgula), . (ponto final).
- Operadores Aritméticos: + (adição), - (subtração), * (multiplicação), / (divisão).

Ao identificar um desses caracteres, o autômato imediatamente encerra a captura do token e classifica-o de acordo com sua função. Caso seja encontrado um caractere diferente dos símbolos unitários esperados, o autômato o rejeita, sinalizando a necessidade de outro módulo assumir a análise ou reportando um erro.

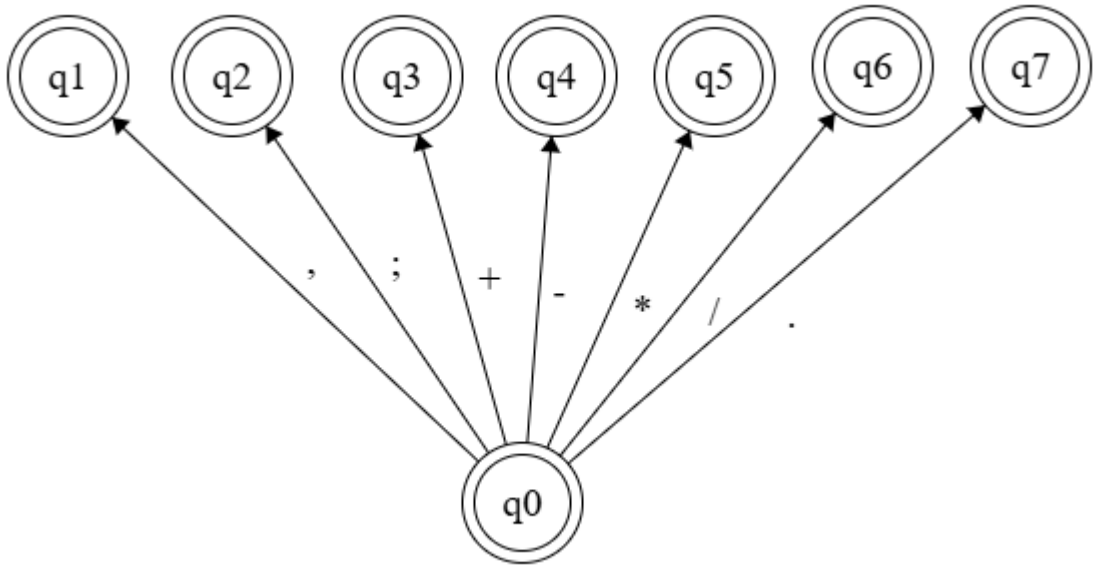

Imagem 01: autômato de símbolos unitários feito usando Finite State Machine Designer

3.2 - Autômato de operadores relacionais

Os operadores relacionais são usados na construção de expressões condicionais e comandos de decisão na linguagem PL/0. Para reconhecer essas estruturas, foi projetado um autômato dedicado, capaz de identificar tanto operadores simples quanto compostos.
Os operadores reconhecidos são:

- Simples: <, >, =
- Compostos: <=, >=, <>, :=

O autômato inicia a leitura a partir de um caractere potencialmente pertencente a um operador (<, >, =, :). Dependendo do símbolo lido, pode haver duas possibilidades:

- O operador é simples e a captura se encerra imediatamente (por exemplo, =).
- O operador pode formar um símbolo composto, exigindo a leitura de um próximo caractere (por exemplo, < seguido de = formando <= ou seguido de > formando <>).

Para tratar essas possibilidades, o autômato implementa transições condicionais baseadas no caractere subsequente, utilizando a técnica de lookahead. Caso o próximo caractere valide a formação de um operador composto, ambos os caracteres são consumidos como um único token. Caso contrário, apenas o primeiro símbolo é considerado, e o segundo caractere é preservado para análise posterior.

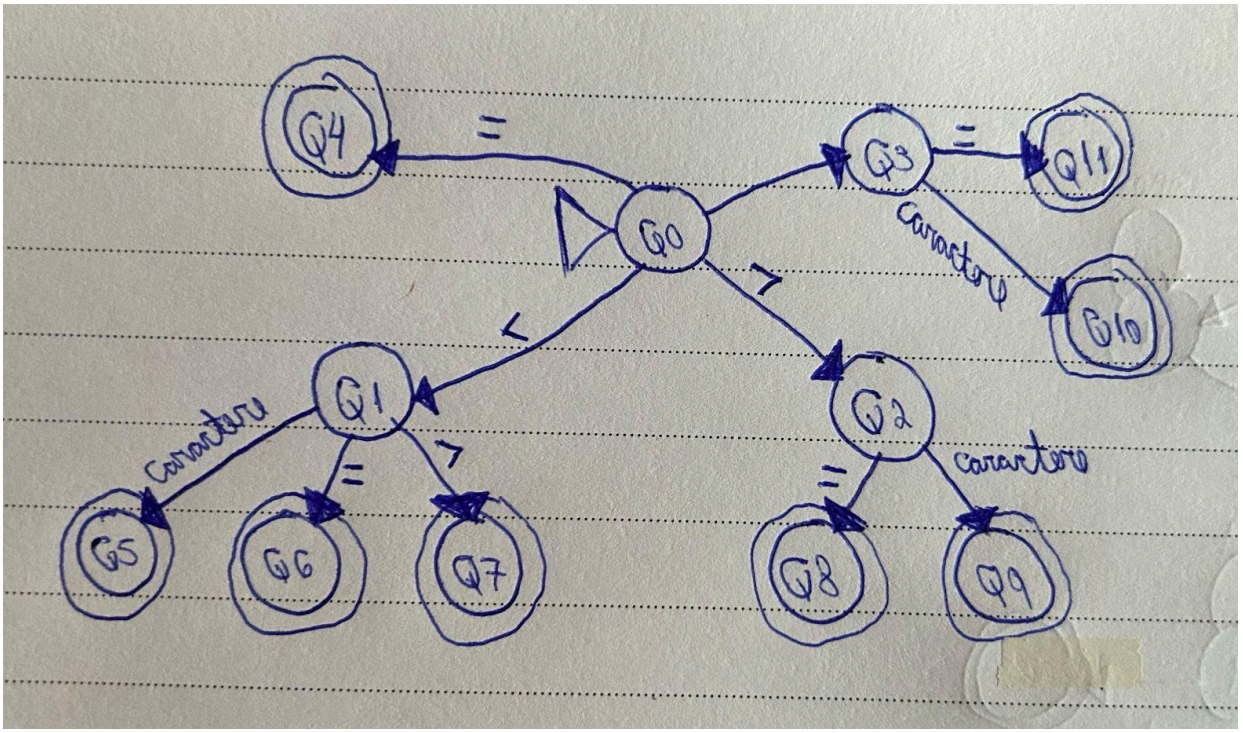

Imagem 02: autômato de operadores relacionais feito a mão

3.3 - Autômato de números inteiros

Na linguagem PL/0, os números são exclusivamente inteiros e seguem regras específicas de formação: devem ser compostos apenas por dígitos e não podem apresentar zeros à esquerda, exceto no caso do número zero propriamente dito. Para reconhecer essas estruturas, foi projetado um autômato específico para números inteiros.
O funcionamento do autômato é dividido em dois casos:

- Quando o primeiro dígito lido é 0, o número é reconhecido imediatamente como zero e o token é concluído, sem permitir dígitos subsequentes.
- Quando o primeiro dígito está no intervalo de 1 a 9, o autômato continua aceitando dígitos subsequentes (0 a 9), permitindo a formação de números maiores, como 23, 4589, etc.

Assim, o autômato evita a formação de números inválidos como 012 ou 0007, que não são aceitos pela gramática de PL/0.

Caso, durante a leitura de dígitos, um caractere diferente de número seja encontrado (como um espaço, símbolo ou letra), o autômato encerra a captura do número e deixa o caractere lido para o processamento do próximo token.

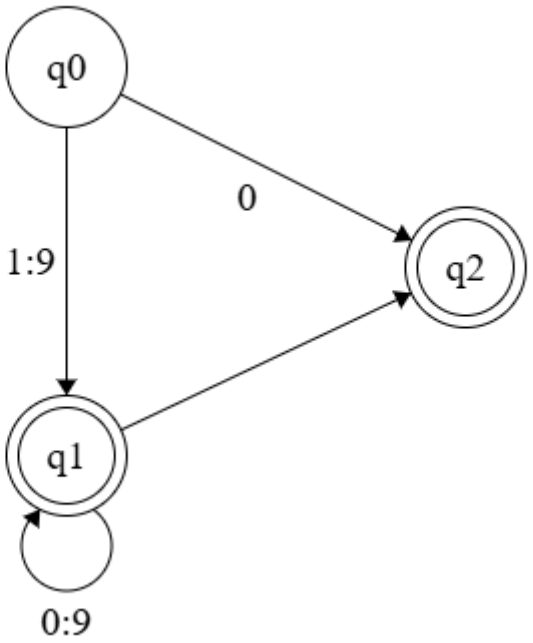

Imagem 03: Autômato de números inteiros feito usando Finite State Machine Designer

3.4 - Autômato de identificadores

Os identificadores são elementos na linguagem PL/0 utilizados para nomear variáveis, procedimentos e constantes. Para reconhecer esses componentes, foi projetado um autômato que segue as regras estabelecidas pela gramática da linguagem.
A formação de um identificador obedece aos seguintes critérios:

- O primeiro caractere deve obrigatoriamente ser uma letra (maiúscula ou minúscula).
- Após o primeiro caractere, podem ser aceitos tanto letras quanto dígitos.
- A linguagem é case-sensitive, ou seja, diferencia maiúsculas de minúsculas (a ≠ A).

O autômato inicia sua operação ao encontrar uma letra. Em seguida, enquanto forem lidos caracteres válidos (letras ou dígitos), o identificador é expandido. A leitura é

encerrada assim que um caractere inválido para a composição de identificadores — denominado "outro" — é encontrado. Nesse momento, o autômato retrocede uma posição para garantir que o caractere não consumido seja corretamente analisado no contexto do próximo token.

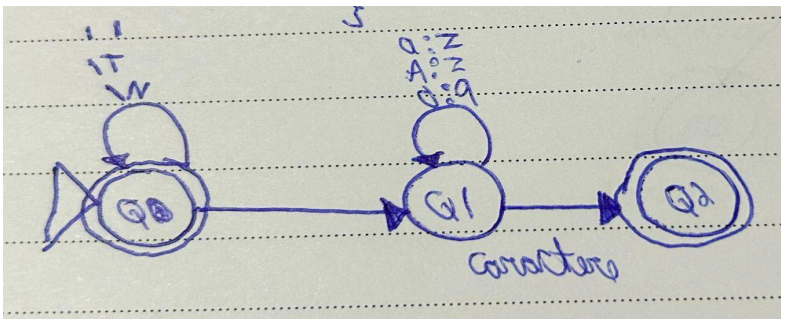

Imagem 04: autômato de identificadores feito a mão

3.5 - Autômato de comentários

Na linguagem PL/0, comentários são utilizados para incluir anotações no código-fonte que devem ser ignoradas pelo compilador. Eles são delimitados pelos símbolos { e }. Para gerenciar corretamente esse comportamento, foi projetado um autômato específico para a detecção e descarte de comentários.

O funcionamento do autômato é simples: ao identificar o caractere {, o analisador entra em um estado de consumo contínuo de caracteres, ignorando seu conteúdo. O autômato permanece nesse estado até encontrar o caractere de fechamento }. Todo o texto entre as chaves é considerado parte do comentário e não gera tokens para as próximas etapas da análise. Importante destacar que, diferentemente dos autômatos de outros tokens, o autômato de comentários não possui um estado final associado à emissão de token. Seu propósito é apenas consumir os caracteres do comentário, sem produzir nenhuma saída léxica. Caso o final do arquivo seja alcançado antes de encontrar o fechamento }, o sistema sinaliza um erro léxico, indicando que o comentário foi encerrado incorretamente.

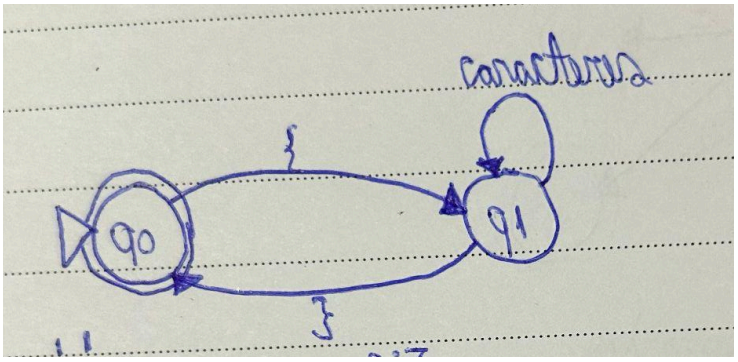

Imagem 05: autômato de comentários feito a mão

4 - Código e compilação

Para testar o funcionamento do analisador léxico, foram fornecidos dois casos teste, um correto e um com erros , abaixo estão expostos os resultados :

4.1 - Caso sem erros

```
CONST
  five = 5;

VAR
  x;

BEGIN
  x := five + 3;
END.
```

Imagem 06: Código teste sem erros.

O código acima não apresenta erros, desse modo o analisador léxico deve lê-lo do início até o fim separando todas as palavras e símbolos reservadas e identificadores corretamente.

```
CONST , CONST
five , identifier
= , symbol_equals
5 , integer
VAR , VAR
x , identifier
; , symbol_semicolon
BEGIN , BEGIN
x , identifier
:= , symbol_assign
five , identifier
+ , symbol_plus
3 , integer
END , END
. , symbol_dot
```

Imagem 07: Output do analisador léxico.

O output final se parece com o desejado , com a análise léxica feita corretamente.

4.2 - Caso com erros

```
VAR
  x$, y;

BEGIN
  x := 10;
  y := x + 5;
{ Comentário nunca fechado...

END.
```

Imagem 08: Código teste com erros .

Vemos que o código teste apresenta dois erros , um sendo a declaração de variável com caractere proibido , neste caso o caractere “$” , e além disso , temos um comentário que não está fechado .

```
VAR , VAR
x , identifier
$ , error_unknown_character
, , symbol_comma
y , identifier
; , symbol_semicolon
BEGIN , BEGIN
x , identifier
:= , symbol_assign
10 , integer
; , symbol_semicolon
y , identifier
:= , symbol_assign
x , identifier
+ , symbol_plus
5 , integer
ERROR: Unexpected end of file encountered.
```

Imagem 09: Output do analisador léxico .

O output do analisador léxico se parece com o desejado , identificando o símbolo proibido e o erro no fim do output se refere ao fato de que todas as linhas abaixo do comentário aberto estão comentadas , sendo assim , o programa não reconhece a linha “END.” e chega ao fim do arquivo inesperadamente .

5 - Conclusão

O desenvolvimento deste projeto proporcionou uma visão prática sobre a importância da análise léxica no processo de compilação. A implementação dos autômatos finitos permitiu compreender, de forma aplicada, como é feita a identificação e a separação dos componentes básicos de uma linguagem de programação. A estrutura modular adotada no código, associada às decisões de projeto voltadas à eficiência e à clareza, resultou em um analisador léxico capaz de reconhecer corretamente os tokens da linguagem PL/0 e de lidar adequadamente com erros léxicos.