Análise Léxica

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Análise Léxica

A **análise léxica** lida com técnicas para especificar e implementar um **analisador léxico**.

Um programa-fonte é informado para um **analisador léxico** como sendo uma única cadeia de caracteres. O analisador léxico coleta caracteres e os agrupa logicamente de acordo com um **padrão**, atribuindo um **código interno** aos agrupamentos de acordo com sua estrutura.

Os **agrupamentos lógicos** podem ser gerados por sequência de caracteres (**lexemas**) **diferentes**.

Os **códigos internos** desses agrupamentos são chamados de **tokens** (**átomos**).

Átomo

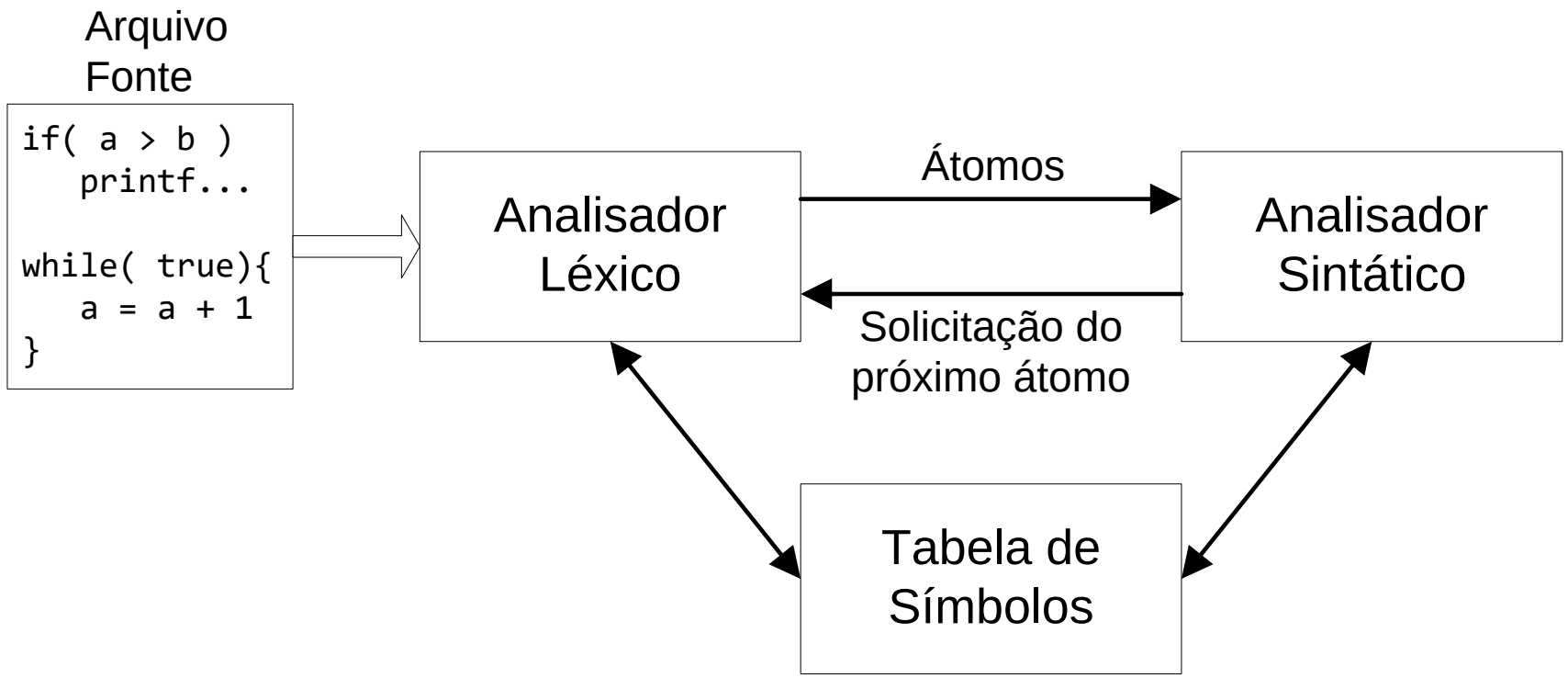
Na maioria das linguagens de programação, os seguintes **agrupamentos lógicos** são considerados **átomos**:

> **palavras-chaves (int, float...), operadores, identificadores, constantes, cadeias de caracteres (strings), e símbolos de pontuação como vírgula, ponto e vírgula, dois pontos, e ponto, etc.**

Diferentes lexemas (sequência de caracteres) podem gerar um **mesmo token**, ou seja, um conjunto de **diferentes caracteres** podem determinar o mesmo **átomo**.

Análise Léxica

Normalmente o **analisador léxico** é implementado como uma **subrotina do analisador sintático**.



Funções do Analisador Léxico

- Extração e classificação de átomos
- Eliminação de delimitadores e comentários
- Conversão numérica
- Identificação de palavras reservadas
- Tratamento de identificadores
- Recuperação de erros
- Interação com o sistema de arquivos
- Controle da numeração de linhas do programa fonte

Atributos para átomos

Quando mais de uma sequência de caracteres (lexema) casa com um padrão para um átomo, o analisador léxico deve retornar informações adicionais sobre o átomo identificado.

> **Exemplo: 10** e **20** podem geram o átomo **NUMERO**, mas é essencial para o gerador de código saber qual cadeia foi de fato reconhecida.

Como descrever a sequência de caracteres (**padrão**) que podem gerar um determinado **átomo** ?

Expressão regular

O **analisador léxico** entende o programa fonte como uma **linguagem regular**.

Uma **expressão regular** é regra de formação para representar todas as combinações de caracteres que estão associadas a determinado **átomo**.

Considere a expressão regular que descreve o padrão (regra de formação) para números inteiros em uma linguagem de programação.

$(+|-|\varepsilon)(0|1|2|3|4|5|6|7|8|9)^{+}$

ou em outra notação

$(+|-)?(0-9)^{+}$

Analisador Léxico

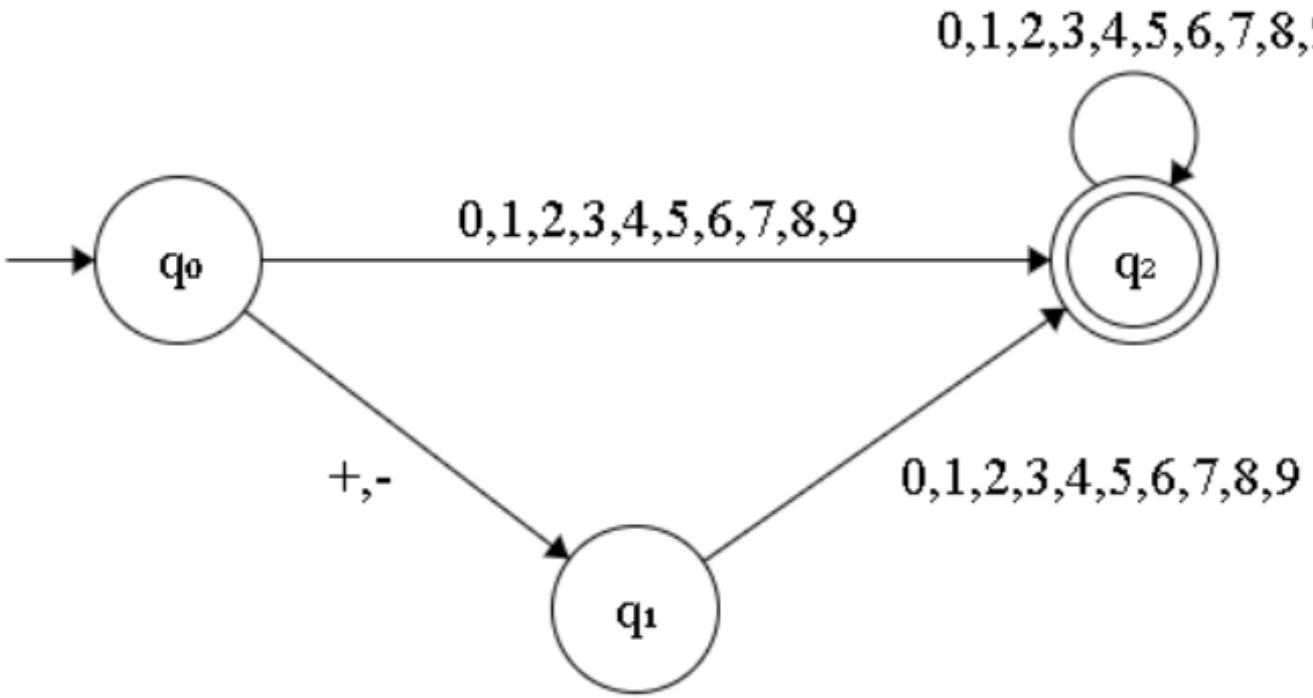
Um **analisador léxico** reconhece os **agrupamentos lógicos** (**lexemas**) de uma arquivo fonte lido de um arquivo texto e produz os átomos para o analisador sintático, a implementação do analisador léxico é baseada em um **autômato finito**.

Um **autômato finito** é uma máquina de estados finitos que reconhece as palavras geradas por uma **expressão regular**.

Implementação do AFD

- Considere a expressão regular vista anteriormente:
  `(+|-|ε)(0|1|2|3|4|5|6|7|8|9)⁺`

- O autômato finito que reconhece as palavras geradas pela expressão regular acima é:



Como implementar um AFD

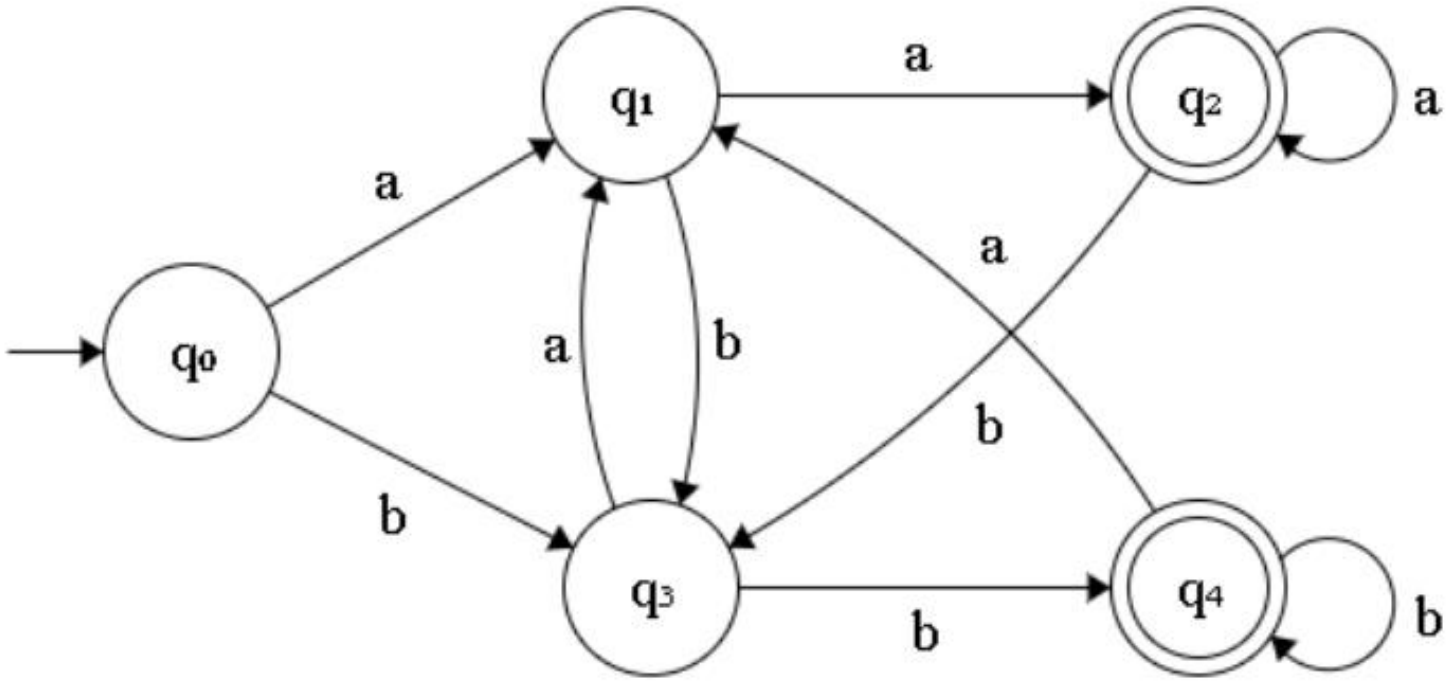
Um **autômato finito** pode ser convertido em **uma função** que toma como entrada cadeias (palavras) e respondem “sim=1” se a cadeia fizer parte da linguagem reconhecida pelo autômato e “não=0” caso contrário.

As cadeias são vetores de caracteres onde sua última posição tem o finalizador de string (‘\0’==0) da linguagem C. Dentro da função usaremos somente rótulos e goto para os rótulos, isso torna a implementação bastante simples.

Vamos implementar a função para o autômato que representa a expressão regular $(+|-|\varepsilon)(0|1|3|4|5|6|7|8|9)^{+}$

Exercícios – Implementação AFD usando goto

1) Baseado na implementação vista em aula, agora você deve implementar uma função para o AFD abaixo que reconhece a linguagem L={W ∈ ∑* | o sufixo de W é aa ou bb } onde ∑={a,b},



Exercícios

2) Construa um autômato e, a partir do autômato, implemente uma função para reconhecer as palavras sobre o alfabeto $\sum=\{a,b\}$ para palavras geradas pela Expressão Regular `(a|ε)(b|ba)*`. Qual é a linguagem gerada pela expressão regular ?

3) Considere a expressão regular (em notação estendida) para IDENTIFICADOR em uma linguagem de programação:

`[A-Za-z][A-Za-z0-9]*_([A-Za-z0-9]⁺|ε)`

Construa o autômato e, baseado no autômato, implemente uma função para reconhecer IDENTIFICADORES dessa linguagem de programação, onde:

`[A-Za-z]` representa todas as letras do alfabeto; e

`[0-9]` representa todos os dígitos de `0` até `9`.

Exercícios

4) Considere a expressão regular para constantes numéricas de uma linguagem algorítmica com o seguinte formato abaixo:

`(+|-)?n(,n)?(E(+|-)?n)?`

Onde:

`n` é uma sequência de um ou mais dígitos, ou seja, `(0|1|2|3|4|5|6|7|8|9)`$^{+}$;

`x?` significa que **`x`** é opcional, ou seja `(x|ε)`.

Construa um autômato para a expressão regular acima, e baseado no autômato, implemente uma função que reconheça as constantes numéricas representadas pela expressão regular.

Exercícios - Implementação

5) Reescreva as implementações, trocando as funções que utilizam goto para funções que utilizam `while` e `switch .. case`, ou seja, sem o uso de `goto`.

6) Escreva um programa que leia um arquivo fonte na linguagem C e converte todo o texto que está dentro de um comentário de uma linha (`//`) para maiúsculo. A saída do programa pode ser a tela do computador.

7) Escreva um programa em C que remova os comentários uma linha (`//`) de um programa fonte escrito na linguagem C padrão, considere que o arquivo está léxica e sintaticamente correto.

   O programa fonte original deve ser lido de um arquivo, o seu programa terá como saída um novo arquivo fonte "limpo“, sem os comentários (`//`) e tudo que estava dentro dos comentários, e é claro, o programa deverá continuar compilando sem erros como antes.

Exercícios - Implementação

8) Escreva um programa em C que remova os comentários várias linhas (`/* */`) de um programa fonte escrito na linguagem C, considere que o arquivo está léxica e sintaticamente correto.

   O programa fonte original deve ser lido de um arquivo, o seu programa terá como saída um novo arquivo fonte "limpo“, sem os comentários (`/* */`) e tudo que estava dentro dos comentários, e é claro, o programa deverá continuar compilando sem erros como antes.

Grato, alguma dúvida ?