Mini Analisador Léxico

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Mini analisador léxico

Considere uma linguagem de programação cujos os átomos são os seguintes:

- **Identificador**: começado por uma letra minúscula e seguido letras minúscula ou dígito.
- **Número**: um ou mais dígitos separados por ponto;

Assim podemos dizer que o alfabeto do mini analisador léxico é formado por **letras minúscula**, **dígitos**, **ponto** e **caracteres delimitadores** (**espaços em branco**, **quebra de linhas**, **tabulação** e **retorno de carro**), caso o analisador encontre um caractere estranho, que não faça parte do alfabeto do analisador léxico, o analisador deve parar com **erro léxico**.

O analisador é **sensível ao caso**, ou seja, o lexema **`var1`** é um identificador, mas o lexema **`Var1`** é considerado **erro léxico**, pois o mini analisador não aceita letra em maiúsculo.

Definições regulares

Para facilitar a escrita das expressões regulares dos átomos para o mini analisador léxico vamos dar nome as expressões regulares (**definições regulares**), e assim a partir das definições regulares mais simples podemos definir outras definições mais complexas.

Para o mini analisador léxico teríamos as seguintes **definições regulares simples**.

LETRA_MINUSCULA → a|b|...|z
LETRA_MAIUSCULA → A|B|...|Z
DIGITO → 0|1|...|9

Definições que representam os átomos (mais complexas)

IDENTIFICADOR → **LETRA_MINUSCULA**(**LETRA_MINUSCULA**|**DIGITO**)*
NUMERO → **DIGITO**+.**DIGITO**+

Autômatos finitos determinísticos (AFD)

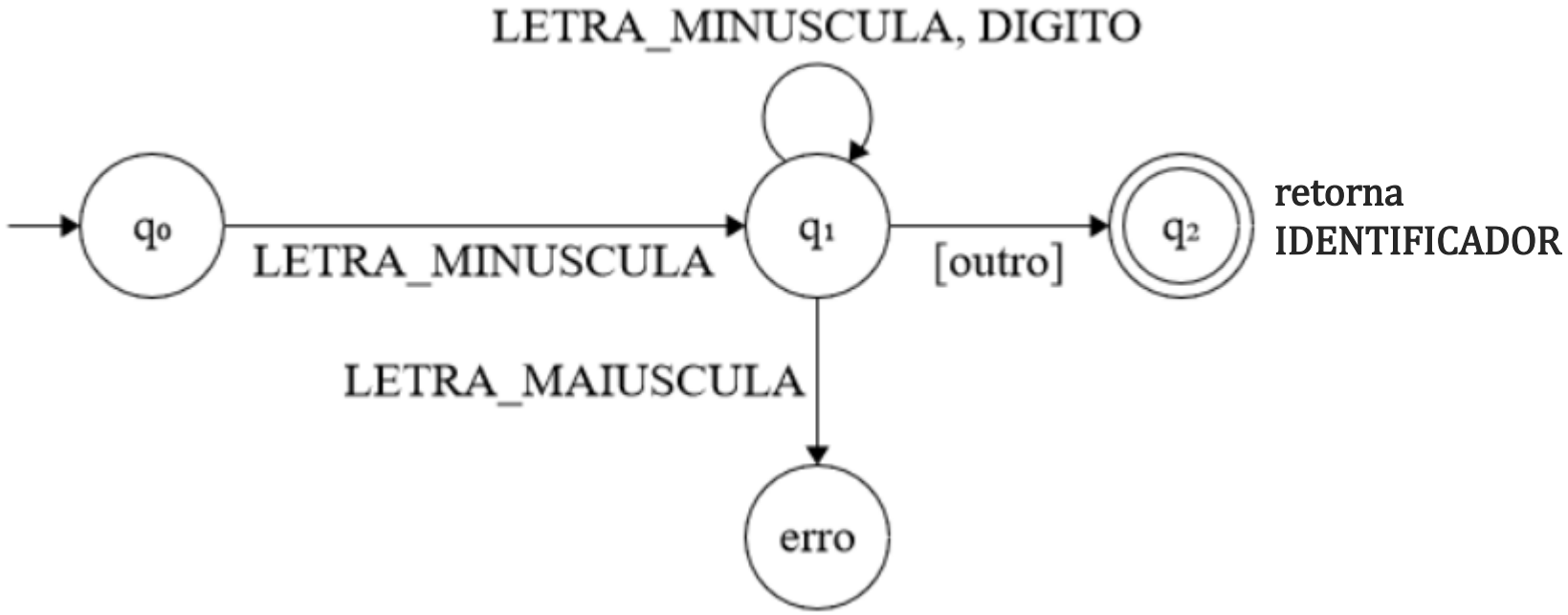
Um AFD representa os algoritmos que aceitam cadeias de caracteres que casam com padrões definidos por uma expressão regular, mas para nosso analisador léxico usaremos um AFD, não no sentido tradicional, pois um AFD precisa ter uma transição para cada estado e caractere, e isso acaba não refletindo o comportamento das funções que reconhecem os átomos de uma linguagem de programação.

Por exemplo um AFD tradicional não especifica o que ocorre quando temos um **erro no reconhecimento da cadeia**.

Por conta disso os AFD para **definições regulares** terão algumas modificações em suas transições e estados de **erros**.

Átomo `IDENTIFICADOR`

Para a definição regular do átomo **`IDENTIFICADOR`** temos o seguinte autômato.
`IDENTIFICADOR` $\rightarrow$ **`LETRA_MINUSCULA(LETRA_MINUSCULA|DIGITO)`***



A transição **`[outro]`**, com colchetes, indica que **caractere identificado** deve ser considerado mas não consumindo na transição, o caractere identificado finaliza o reconhecimento do átomo identificador.

Átomo `IDENTIFICADOR`

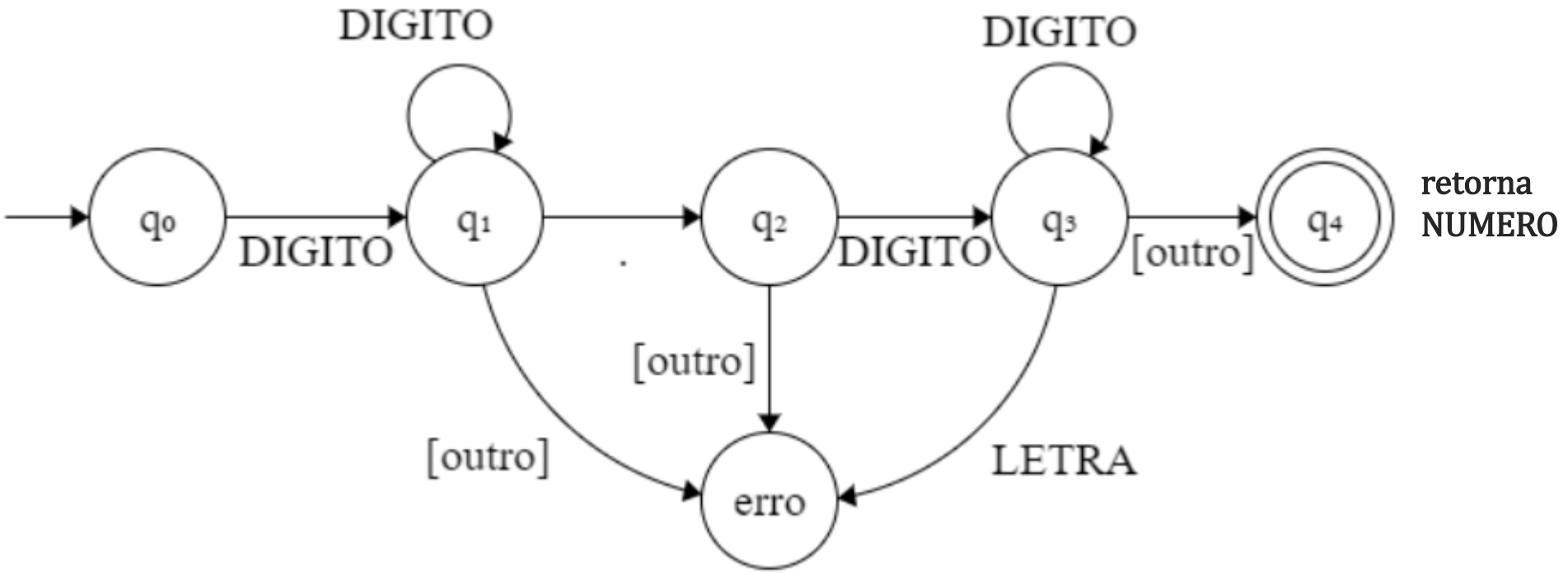
No autômato para reconhecer o átomo **`IDENTIFICADOR`** o caractere **`[outro]`** é qualquer caractere diferente de **`LETRA_MAIUSCULA`** ou **`DIGITO`** e presente do alfabeto do mini analisador léxico.

Na implementação do mini analisador léxico, o estado de **`erro`** é acessado quando após uma sequência de **`LETRAS MINÚSCULAS`**, vier uma **`LETRA MAIÚSCULA`**, nesse caso o mini analisador léxico interrompe a execução e imprime uma **mensagem de erro**, por exemplo, **`numero da linha # erro léxico`**.

Caso o analisador chegue ao **estado final**, o analisador **retorna uma constante numérica** para representar o reconhecimento do átomo.

Átomo NUMERO

Para a definição regular do átomo **NUMERO** temos o seguinte autômato.
NUMERO → **DIGITO⁺.DIGITO⁺**



O estado **erro** é acessado quando após uma sequência de **DIGITOS** vier uma **letra maiúscula ou minúscula**, ou seja, o caractere **[outro]** é qualquer caractere diferente do erro e presente alfabeto do mini analisador léxico.

Implementando o mini analisador léxico

Para codificar a constantes numéricas dos átomos, definiremos algumas **constantes inteiras na linguagem C** a partir das definições regulares, e também, constantes para estado de erro e fim de buffer:

```
#define ERRO 0
#define IDENTIFICADOR 1
#define NUMERO 2
#define EOS 3 // fim de buffer
```

Implementando o mini analisador léxico

Podemos também definir as **constantes inteiras** como um tipo enumerado.

```
typedef enum{
  ERRO,
  IDENTIFICADOR,
  NUMERO,
  EOS
}TAtomo;
```

Rotina principal do mini analisador léxico

A rotina **`obter_atomo()`** do mini analisador léxico retorna para cada **átomo** reconhecido uma **codificação inteira** (constante) para representar o valor do átomo e o seu **atributo**, caso se faça necessário para o átomo.

Essas informações serão utilizados pelo **analisador sintático** avaliar se a sequência de átomos está correta sintaticamente. As informações retornadas pela rotina **`obter_atomo()`** seriam:

- número da linha onde foi identificado o átomo
- Os seguintes átomos possuem **atributos**:
  - **`IDENTIFICADOR:`** o lexema do átomo, suponha que cada lexema terá no **máximo 15 caracteres**, mais que 15 é erro léxico.
  - **`NUMERO:`** o valor do numérico (**`float`**) da constante reconhecida.

Estrutura para armazenar o átomo

Para armazenar todas as informações que serão retornadas pela rotina **`obter_atomo()`**, vamos definir um novo tipo utilizando uma estrutura de registros (`struct`) com um conjunto de campos de tipos diferentes para armazenar cada umas das informações referente ao átomo reconhecido:

```
typedef struct{
  TAtomo atomo;
  int linha;
  float atributo_numero;
  char atributo_ID[16];
}TInfoAtomo;
```

Implementação do mini analisador léxico

- Agora podemos desenvolver o código para o mini analisador léxico. O programa deve ler um o arquivo texto e identificar os átomos (tokens) da linguagem.

- A rotina **`obter_atomo()`**, faz a chamada das sub-rotinas que implementam os autômatos vistos, e retorna as informações dos átomos reconhecidos. A declaração da rotina seria:

```
TInfoAtomo obter_atomo(void);
```

- Para simular o analisador sintático você deve fazer uma função com um laço que ficará chamando a função **`obter_atomo()`** até que chegue ao final do buffer, nesse caso a função **`obter_atomo()`** retornará o átomo **`EOS`**.

Implementação do mini analisador léxico

- Além disso, o analisador faz um **controle das linhas** do programa fonte, e também, a eliminação (ignora) dos **delimitadores** (espaços em branco, tabulação, nova linha e retorno de carro).

- Para cada átomo reconhecido devem ser apresentados (**na tela**) as seguintes informações:

  `Número da Linha do Átomo # Átomo | Atributo`

- Por exemplo:

  `11# IDENTIFICADOR | var1`

- Caso ocorra um erro no reconhecimento de um dos átomos do programa fonte analisado, o seu programa deve parar a execução com uma mensagem informando a linha onde ocorreu o **erro.**

Exercícios – mini analex

1) Modifique o mini analisador léxico para que agora ele reconheça os átomos operador adição (“+”), atribuição (“=“) operador de igualdade (“==“). Para cada um desses átomos o analisador deve retornar uma constante numérica para representar o átomo. **Importante:** esses átomos não possuem atributos.

2) Considere que agora o mini analisador léxico deverá reconhecer palavras reservadas, por exemplo o lexema **`while`** gera o **átomo `WHILE`** e não mais o átomo **`IDENTIFICADOR`**. Modifique o mini analisador léxico para reconhecer a palavra reservada **`WHILE`**.

Exercícios – mini analex

3) Considere que o mini analisador léxico deve reconhecer todas as palavras reservadas da linguagem C. Para cada palavra reservada encontrada o analisador retorna um átomo equivalente. **Importante:** os átomos das palavras reservadas não possuem atributos.

   Abaixo a lista de palavras reservadas da linguagem C.

   http://linguagemc.com.br/lista-de-palavras-reservadas-em-c/

4) Escreva um programa na linguagem C que tenha como entrada um arquivo fonte de um programa escrito na linguagem C e tem como saída um arquivo fonte modificado com todas as palavras reservadas no arquivo de entrada em MAIÚSCULO.

Exercícios – mini analex

5) Considere a expressão regular para o átomo **IDENTIFICADOR**:

`[a-z][A-Za-z0-9]*_([0-9]⁺|ε)`

onde:

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

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

São exemplos de palavras geradas pelo expressão regular:

`var1_2`, `var2_`, `a_`, `abcDEF012_32`, ...

E exemplos de erros no analisador léxico: `A0`, `0a`, `aAZ0`, `Abc_10`, ...

Modifique o mini analisador léxico para que agora os átomos **IDENTIFICADOR** sejam reconhecidos a partir da expressão regular acima.

Exercícios – mini analex

6) Considere a expressão regular para o átomo **NUMERO**:

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

onde **n** é uma sequência de um ou mais dígitos;

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

São exemplos de palavras geradas pelo expressão regular:

`+1E3`, `12.3E-1`, `100`, `10.1`, `...`

E exemplos de erros no analisador léxico: `.23`, `1.3+1`, `1.2-`, `100.`, ...

Modifique o mini analisador léxico para que agora os átomos **NUMERO** sejam reconhecidos a partir da expressão regular acima.

Exercícios – Questão Discursiva 70 - ENADE 2005

7) Considere a tabela abaixo que apresenta os símbolos numéricos para números na notação romana e os dígitos equivalente na notação arábica.

símbolos romanos	símbolos arábicos
I	1
II	2
III	3
IV	4
V	5
VI	6
VII	7
VIII	8
IX	9
X	10
.....	
L	50

Modifique o mini analisador léxico para que agora o átomo **NUMERO** seja na notação romana, além disso no atributo **atributo_numero** deve ser guardado o número em notação decimal.

Considere que valor máximo reconhecido pelo seu autômato seja **50=L**

Exercícios – DESAFIO

7) Escreva um programa na linguagem C que tenha como entrada um arquivo fonte de um programa escrito na linguagem C, o seu programa deve verificar se as chaves do programa fonte estão aninhados de forma correta.

Dúvidas ??

boa sorte !