

# Trabalho 1: Gerador de Analisadores Léxicos

Linguagens Formais e Compiladores  
Prof<sup>a</sup>. Jerusa Marchi

## 1. Objetivo do Trabalho:

O objetivo deste trabalho é o desenvolvimento de um arcabouço (framework) para gerar analisadores léxicos. O arcabouço consiste na implementação dos algoritmos necessários a criação de um Autômato Finito para atender as definições regulares expressas no arquivo de entrada. A saída deve ser a lista de tokens gerada, associando ao lexema, seu padrão.

Para a construção de um gerador de analisador léxico são necessários os seguintes algoritmos:

- (a) Conversão de Expressão Regular para Autômato Finito Determinístico;
- (b) União de Autômatos via  $\epsilon$ -transição;
- (c) Determinização de Autômatos;
- (d) Construção da TS (incluir palavras reservadas, para associar corretamente ao padrão - PR - na construção do token.

O fluxo de execução desta etapa é o seguinte:

- A **interface de projeto**, ou arquivo de entrada, **do analisador léxico** deve permitir a inclusão de, ou incluir, expressões regulares para todos os padrões de tokens, usando definições regulares;
- Para cada ER deve ser gerado o AFD corresponde;
- Os AFD devem ser unidos com  $\epsilon$ -transições;
- O AFND resultante deve ser determinizado gerando a tabela de análise léxica (representação implícita).
- A **interface de execução**, ou arquivo de teste, do analisador léxico deve permitir a entrada de um texto fonte;
- O texto fonte será analisado, utilizando a tabela de análise léxica gerada na parte de projeto, e deve gerar um arquivo de saída com a lista de todos os tokens encontrados na forma  $\langle \text{lexema}, \text{padrão} \rangle$  ou reportar erro *lexema*, erro! caso a entrada não seja válida.

Observações:

- As notações devem ser as utilizadas em sala (para notacionar  $\epsilon$  usar o  $\&$ );
- A tabela de análise léxica deve poder ser "visualizada"(arquivo e tela - na forma de tabela);

- Os Autômatos Finitos gerados pela Conversão das ERs também devem poder ser visualizados (arquivo ou tela - na forma de tabela);

## 2. Realização:

O trabalho deverá ser realizado em grupos de no mínimo 2 e no máximo 3 integrantes. Os grupos deverão, ao final do semestre apresentar o trabalho em dia e horário agendado (juntamente com o analisador sintático), onde serão executados testes das partes em separado (por esta razão é muito importante que as partes possam ser testadas de forma independente). Além dos fontes, executáveis e de um HowTo, o grupo deve prover exemplos variados de testes, incluindo diferentes definições léxicas para que o trabalho possa ser melhor avaliado.

## 3. Avaliação:

O trabalho será avaliado pela robustez e corretude dos algoritmos. Além disso, quesitos como legibilidade do código e organização dos fontes também serão considerados.

# 1 Anexo I - Formato de entrada de ER

Os arquivos com ER devem seguir o padrão:

```
def-reg1: ER1
def-reg2: ER2
...
def-regn: ERn
```

As ER devem aceitar grupos como [a-zA-Z] e [0-9] e os operadores usuais de \* (fecho), + (fecho positivo), ? (0 ou 1) e | (ou).

- Exemplo1:

```
id: [a-zA-Z]([a-zA-Z] | [0-9])*
num: [1-9]([0-9])* | 0
```

- Exemplo 2:

```
er1: a?(a | b)+
er2: b?(a | b)+
```

# 2 Anexo II - Formato de saída dos AF

Os arquivos com Af devem seguir o padrão:

```
número de estados
estado inicial
```

estados finais  
alfabeto  
transições (uma por linha)

- Exemplo afd:

5  
0  
1,2  
a,b  
0,a,1  
0,b,2  
1,a,1  
1,b,3  
2,a,4  
2,b,2  
3,a,1  
3,b,3  
4,a,4  
4,b,2

### **3 Anexo III - Exemplo de entrada do arquivo de teste e possível saída**

Os arquivos de teste dependem das definições regulares introduzidas, assim considerando o exemplo 1 e 2 do anexo ??, tem-se

- Exemplo entrada para ER que reconhece identificadores e numeros:

a1  
0  
teste2  
21  
alpha123  
3444  
a43teste

que irá gerar a seguinte lista de tokens:

<a1,id>  
<0, num>  
<teste2, id>

<21, num>  
<alpha123, id>  
<3444, num>  
<a43teste, id>

- Exemplo entrada para ER  $a^?(a \mid b)^+$

aa  
bbbba  
ababab  
bbbbbb

que irá gerar a seguinte lista de tokens:

<aa , er1>  
<bbbba, er2>  
<ababab, er1>  
<bbbbbb, er2>