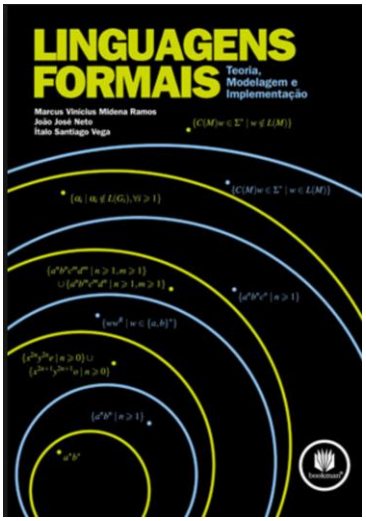
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
Linguagens Formais e Autômatos – Aula 08 – 1º SEMESTRE/2016
Prof. Luciano Silva

TEORIA: IMPLEMENTAÇÃO DE AUTÔMATOS FINITOS DETERMINÍSTICOS

Nossos **objetivos**nesta aula são:

- conhecer alternativas de implementação de autômatos finitos determinísticos
- praticar com implementação autômatos determinísticos



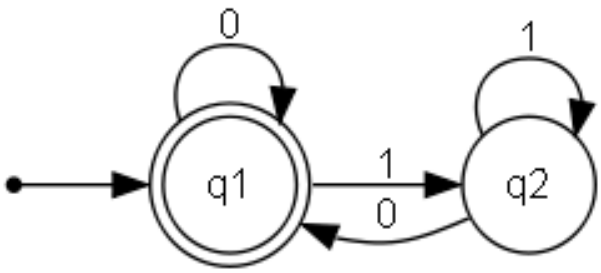
Para esta semana, usamos como referência a **Seção3.12 (Modelagem e Implementação)** do nosso livro da referência básica:

RAMOS, M.V.M., JOSÉ NETO, J., VEJA, I.S. **Linguagens Formais: Teoria, Modelagem e Implementação**. Porto Alegre: Bookman, 2009.

Não deixem de ler esta seção depois desta aula!

TEORIA: IMPLEMENTAÇÃO DE AUTÔMATOS FINITOS DETERMINÍSTICOS

- Existem diversas formas de se implementar um autômato finito determinístico (afd). Em linguagens que suportam rótulos e goto, a implementação fica muito simplificada. Por exemplo:



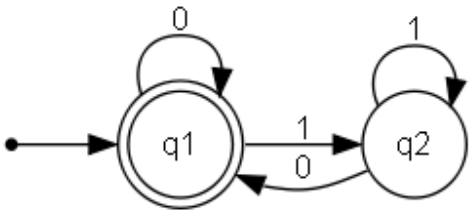
IMPLEMENTAÇÃO EM C	IMPLEMENTAÇÃO EM JAVA

void scanner(){
char c;
q1: c=getchar();
if (c=='0') **goto** q1;
else if (c=='1') **goto** q2;
return;
q2:c=getchar();
if (c=='0') **goto** q1;
else if (c=='1') **goto** q2;
return;
}
	void scanner(){
	char c;
	q1: c=(char) System.in.read();
	if (c=='0') **break** q1;
	else if (c=='1') **break** q2;
	return;
	q2:c=(char) System.in.read();
	if (c=='0') **break** q1;
	else if (c=='1') **break** q2;
	return;
	}

- Observe que, na implementação acima, não diferenciamos entre estados finais e não-finais. Uma maneira de se fazer isto é alterando-se o retorno de scanner() para que indique em qual estado o afd parou no reconhecimento.

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Uma maneira alternativa de se implementar um afd é construindo-se uma tabela de transições. Vamos considerar, novamente, o afd apresentado na parte teórica anterior:



Vamos, inicialmente, construir dois vetores: um para estados e outro para o alfabeto.

```
int estado[2]={1,2} ;          // estados q1 e q2
char alfabeto[2]={'0','1'};   // símbolos 0, 1
```

A partir destes dois vetores, montamos uma matriz para indicar as transições:

```
int trans[2][2] = {{0,1},
                   {0,1}};
```

Nesta representação, a posição trans[0][1]=1 indica que estamos no estado 0 (q1), lemos a letra 1 ('1') e vamos para o novo estado q (q2). Como base nesta estratégia, implemente novamente a função scanner():

PACOTE JAVA.UTIL.REGEX

- Várias linguagens disponibilizam pacotes especiais para geração de autômatos finitos a partir de expressões regulares. Por exemplo, a linguagem Java possui o pacote java.util.regex para geração e busca de padrões em strings:

```
import java.util.regex.Matcher;
        import java.util.regex.Pattern;

public class Automato {
        public static void main(String[] args) {
                Pattern pattern = Pattern.compile("es");// O padrão será "es"
        Matcher matcher = pattern.matcher("busca simples");
        while (matcher.find()) {
                System.out.println(matcher.start() + " - " + matcher.group());
        }
        }
}
```

O resultado deste código será: **11-es**

Pode-se construir expressões regulares complexas através dos códigos abaixo:

- **\d** representa números.
- **\s** representa um espaço em branco.
- **\w** representa letras, números ou o "_" (underscore).
- **.** representa qualquer dígito.
- **[]** representa uma cadeia de valores. Ex: [a-c] buscaria a ou b ou c.
- **?** representa zero ou uma ocorrência.
- ***** representa zero ou mais ocorrências.
- **+** representa uma ou mais ocorrências.
- **^** representa negação.
- **()** agrupa os padrões, usado com os metacaracteres acima.

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Construa um programa em Java que utilize o pacote java.util.regex para encontrar todas as posições de um string que contenham números inteiros. O número identificado deve ser exibido.

- Além dos pacotes de algumas linguagens, existem programas especiais como o lex (C/C++), flex(C/C++), JavaLex(Java), JavaCC(Java) e SableCC(Java) que, a partir de especificações de expressões regulares, geram implementações de autômatos finitos determinísticos (analisadores léxicos). Abaixo, tem-se um exemplo de especificação em lex:

```
%{
#include <iostream>
%}
%%
[ \t] ;
[0-9]+\.[0-9]+ { cout <<"Ponto Flutuante:"<< yytext << endl; }
[0-9]+        { cout <<"Inteiro:" << yytext << endl; }
 [a-zA-Z]+  { cout << "Nome:" << yytext << endl; }
%%
main() {
            yylex();
}
```

arquivo automato.lex

- Para se gerar o código C++ que implemente um autômato finito determinístico minimizado para estas expressões regulares, chamamos o seguinte comando:

lex automato.lex

- O resultado deste comando gera um arquivo chamado lex.yy.c, que contém a implementação do autômato.

- O programa lex é nativo de UNIX e LINUX, mas existem também versões para WINDOWS e MacOS.

EXERCÍCIO COM DISCUSSÃO PAREADA (III)

Construa uma especificação lex que seja capaz de identificar os **tokens** de um comando de atribuição com a seguinte sintaxe:

variável = número inteiro com ou sem sinal ;

EXERCÍCIOS EXTRA-CLASSE

1. Considere uma lista de contatos formada por nomes, endereços e números de telefones (sem DDD/DDI), conforme mostrado no exemplo abaixo:

   Carlos Alberto de Morais
   Rua das Grumixamas, 223
   234-4567
   Maria dos Anjos Celes
   Alameda dos Apiacás, 2345 – Apto 78
   2342935

   Observe que, por algum motivo, alguns números de telefone tem (-) e outros não. Construa um programa em Java que utilize o pacote java.util.regex para exibir todos os números de telefone que constam da agenda. Considere que os números de telefone tenham 7 dígitos, descontando-se o sinal de (-).

2. Construa uma especificação em lex que permita contar quantas if's aparecem em um programa C.