Curso: Engenharia de Computação

Linguagens Formais e Compiladores

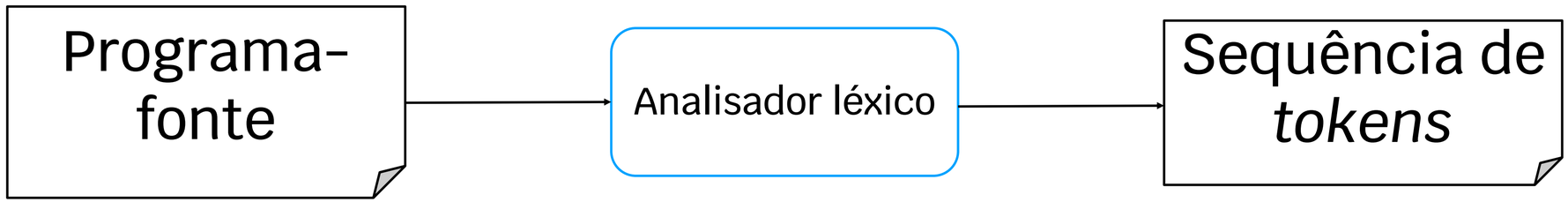
Prof. Clayton J A Silva, MSc
clayton.silva@professores.ibmec.edu.br

ibmec

Varredura ou análise léxica

Varredura ou análise léxica

- É a fase de um compilador que **lê o arquivo de um programa-fonte como um arquivo de caracteres** e **o converte em uma sequência de *tokens***.



Varredura ou análise léxica

- Genericamente, os *tokens* podem ser **palavras reservadas ou palavras-chave**, **identificadores**, **símbolos especiais** e **literais ou constantes**
- Os *tokens* classificam os *lexemas*; existem *tokens* que podem classificar somente um lexema; existem tokens que podem classificar infinitos lexemas
- Qualquer **valor** associado a um *token* recebe o nome de um **atributo** do *token*
- O sistema de varredura poderá definir vários atributos a cada *token* quanto necessários para prosseguir o processamento

Varredura ou análise léxica

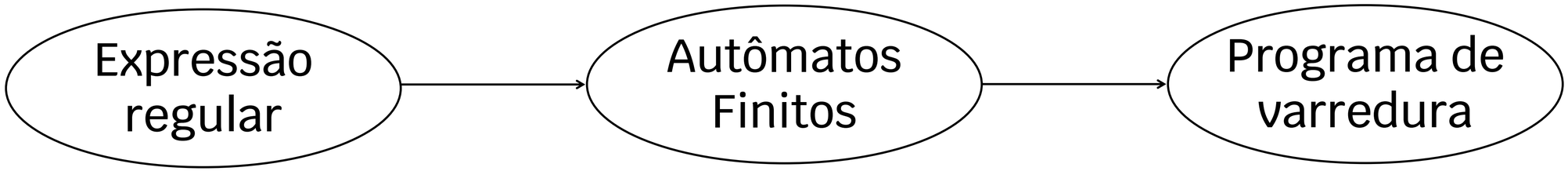
- O **conjunto de atributos** de um token pode ser atribuído a uma estrutura de dados chamada **registro** do token

```
struct
  { TokenTipo tokenval;
    tipo atributo1token;
    ...
  }
```

- O analisador sintático – que controla a análise léxica - normalmente chama o *token* por uma **função, que retorna o próximo *token* identificado no código**

Varredura ou análise léxica

- O processo para desenvolver o programa de varredura (análise léxica) geralmente consiste em aplicar **métodos de especificação e de reconhecimento de padrões**: **expressões regulares** e **autômatos finitos**.



Varredura ou análise léxica

- As expressões regulares constituem uma **notação padrão para representar padrões em cadeias de caracteres** que formam a estrutura léxica de uma LP

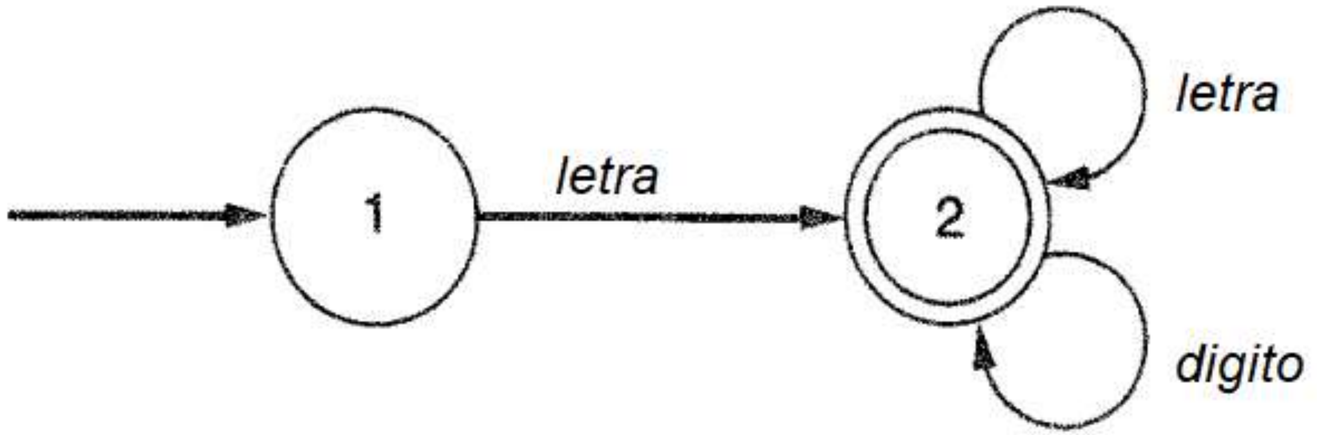
- As máquinas de estados finitos, ou autômatos finitos, são **algoritmos para reconhecimento de padrões** em cadeias de caracteres dadas por expressões regulares.

Autômatos Finitos

- Forma matemática de descrever tipos particulares de algoritmos ou máquinas
- Podem ser utilizados para descrever o **processo de reconhecimento de padrões** em cadeias de entrada, logo para construir **sistemas de varredura**

Autômatos Finitos

- Exemplo: Seja a expressão regular dada por
  letra = [a–z]
  digito = [0–9]
  identificador = letra(letra | digito)*



Autômatos Finitos

- Utilizam o conceito de **estados** e **transições**
- **Estados** = posições do processo de reconhecimento que indicam o quanto o padrão já foi visto
  - Estado inicial
  - Estado de aceitação
- **Transições** = Alterações de um estado para outro com base no casamento de padrões que o **rotulam**

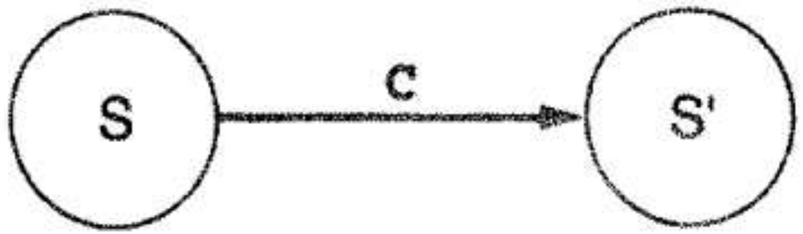
Autômatos Finitos

- **Autômatos Finitos Determinísticos (DFA)** – estado seguinte é unicamente determinado pelo estado atual e caractere de entrada

- Matematicamente, um DFA é composto por
  - Alfabeto, Σ
  - Conjunto de estados, *S*
  - Função de transição *T: Si* $\rightarrow$ *Sj*
  - Um estado inicial s_0
  - Um conjunto de estados de aceitação $A \in S$

DFA

- Dado um estado *s*, a transição *s* para um estado *s'* é condicionada por um rótulo ***c***, representado graficamente por



- A aceitação é caracterizada por uma sequência de estados a partir de s_0, de tal forma que $s_1=T(s_0,c_0)$, $s_2=T(s_1,c_1)$,..., $s_n=T(s_{n-1},c_{n-1})$, onde s_n é o estado de aceitação

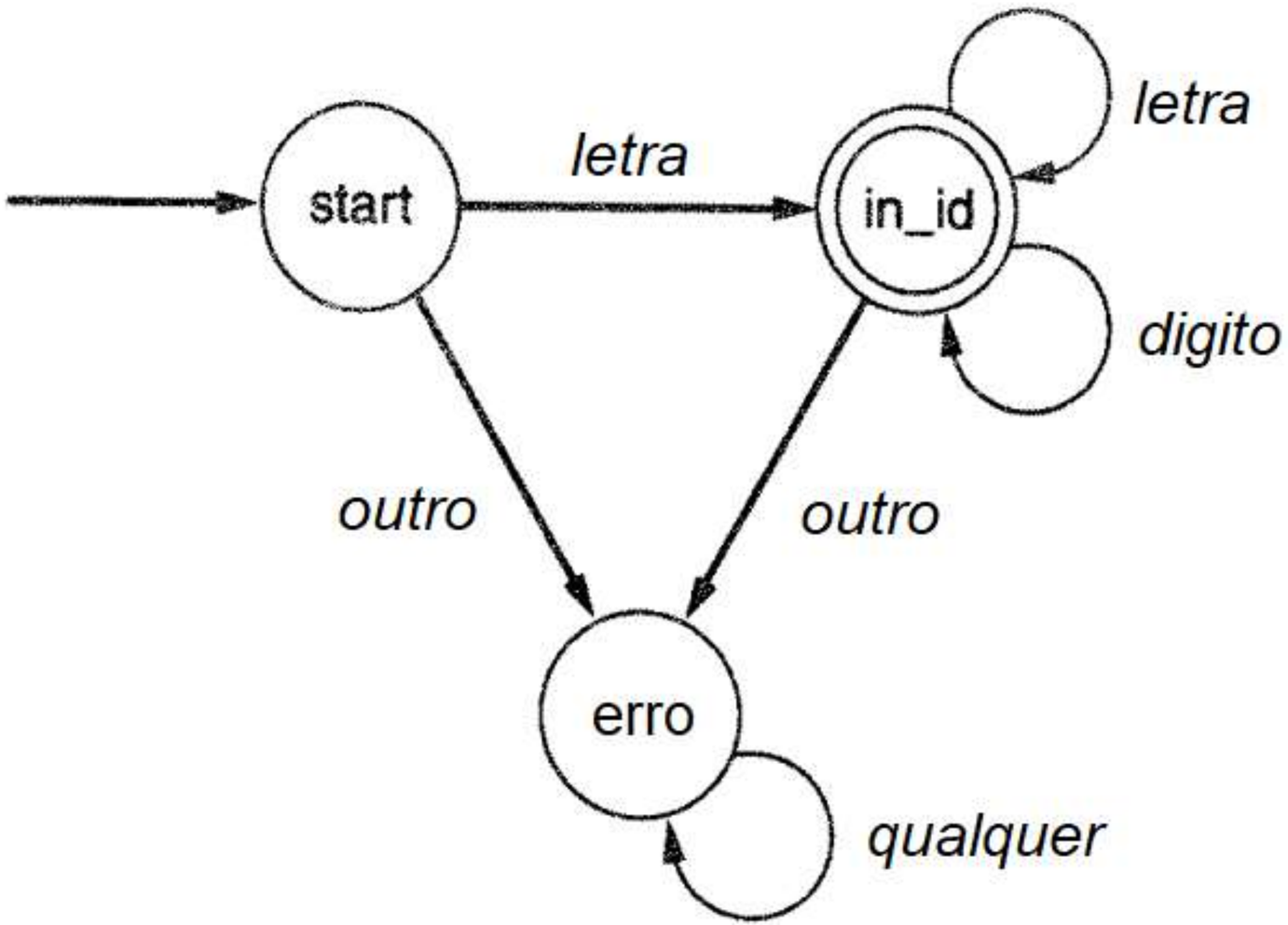
DFA

- Formalizando mais algumas convenções...
  - uso de nomes para identificar os estados: estado inicial designado como *start*; estado de aceitação designado por *in_id*
  - Rótulos não contemplam somente letras, mas rótulos que definem expressões regulares
  - Admite que sempre haverá transição de um estado a outro para todo par de um caractere do alfabeto com um rótulo, no entanto as transições de erro não são apresentadas no DFA

DFA

- Exemplo 1:

identificador

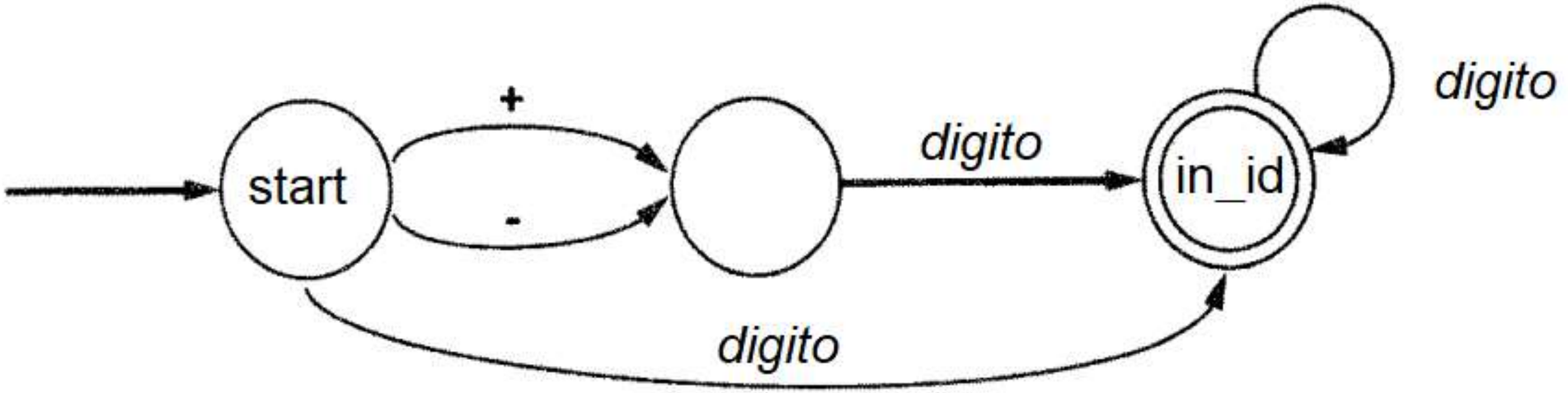


DFA

- Exemplo 2: natural

DFA

- Exemplo 3: sinalNatural



Outras definições – DFA

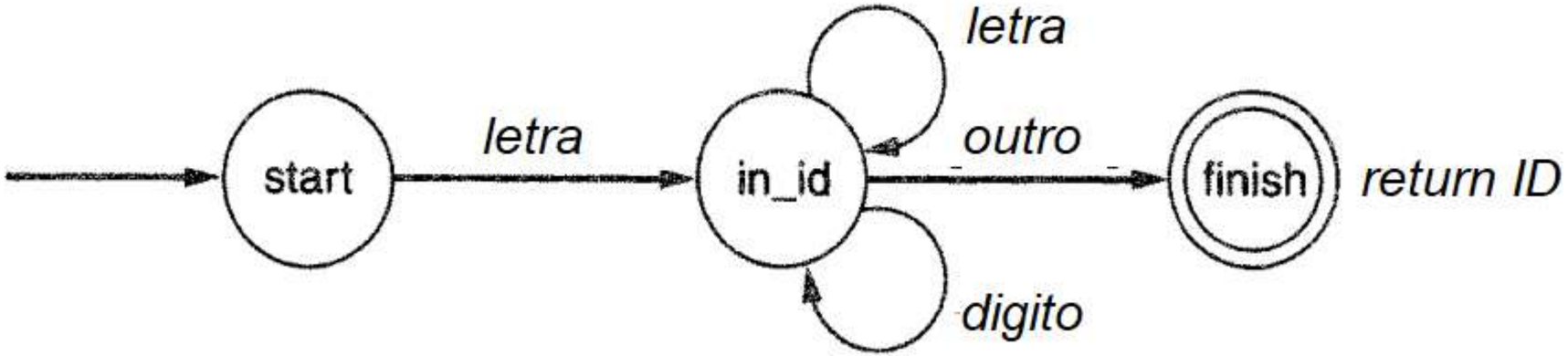
1. Retornar o caractere da transição para uma *string* que armazena a cadeia de caracteres do *token*

2. Retornar o *token* reconhecido quando for alcançado o estado de aceitação

3. Retornar o *token* quando houver uma identificação de erro

Outras definições - DFA

- Exemplo: identificador



Autômatos Finitos

- **Autômatos Finitos Não Determinísticos (NFA)** – possibilita várias transições para o mesmo caractere

- Inclui a **transição vazia** (transição ε): transição que ocorre sem consulta ao sinal de entrada (e sem ler qualquer caractere).

- Representada por

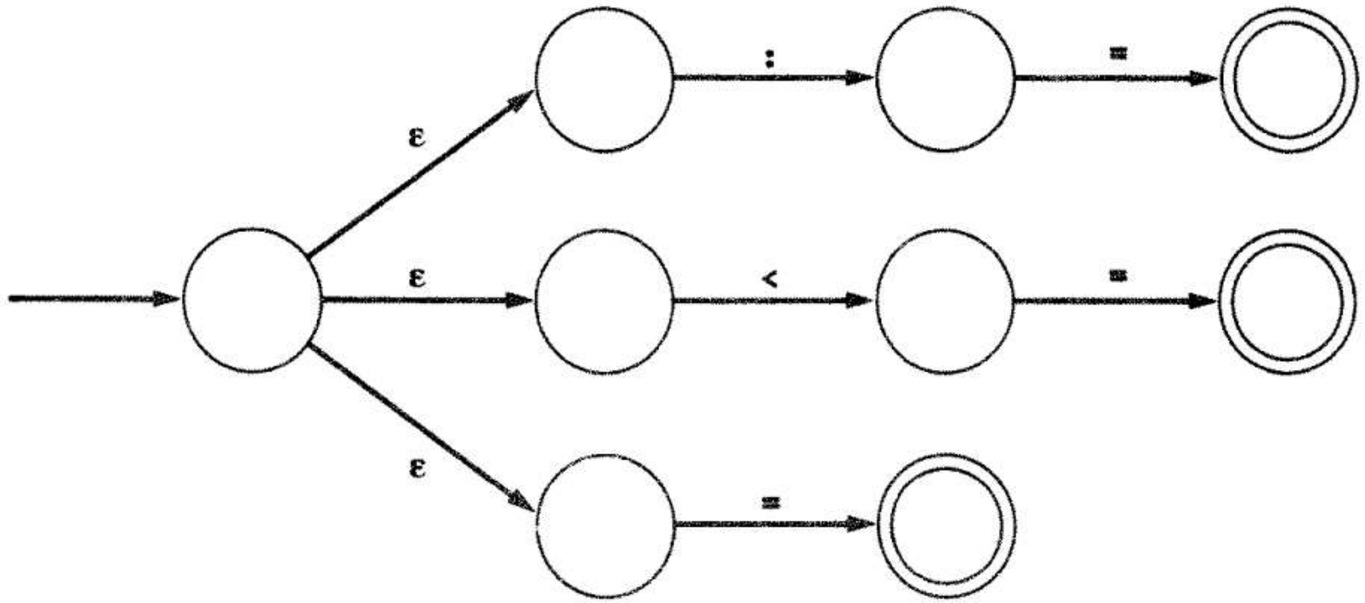
NFA

- **Autômatos Finitos Não Determinísticos (NFA)** – possibilita várias transições para o mesmo caractere
- Transição vazia (transição ε): transição que ocorre sem consulta ao sinal de entrada (e sem ler qualquer caractere).
- Representada por

NFA

- As transições vazias são úteis por dois motivos, sem usar *lookahead* e sem mudanças do caracteres à frente:

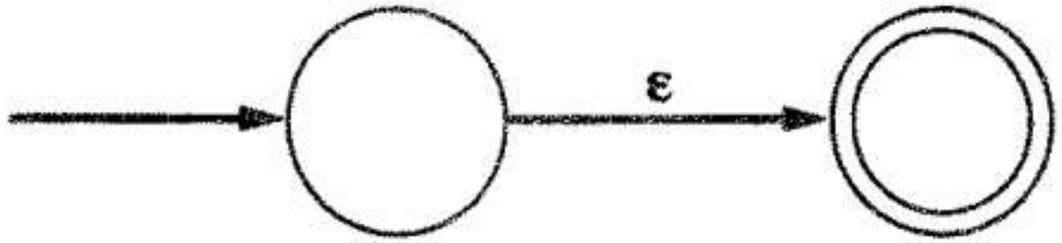
  - pode expressar escolha de alternativas sem envolver combinação de estados



NFA

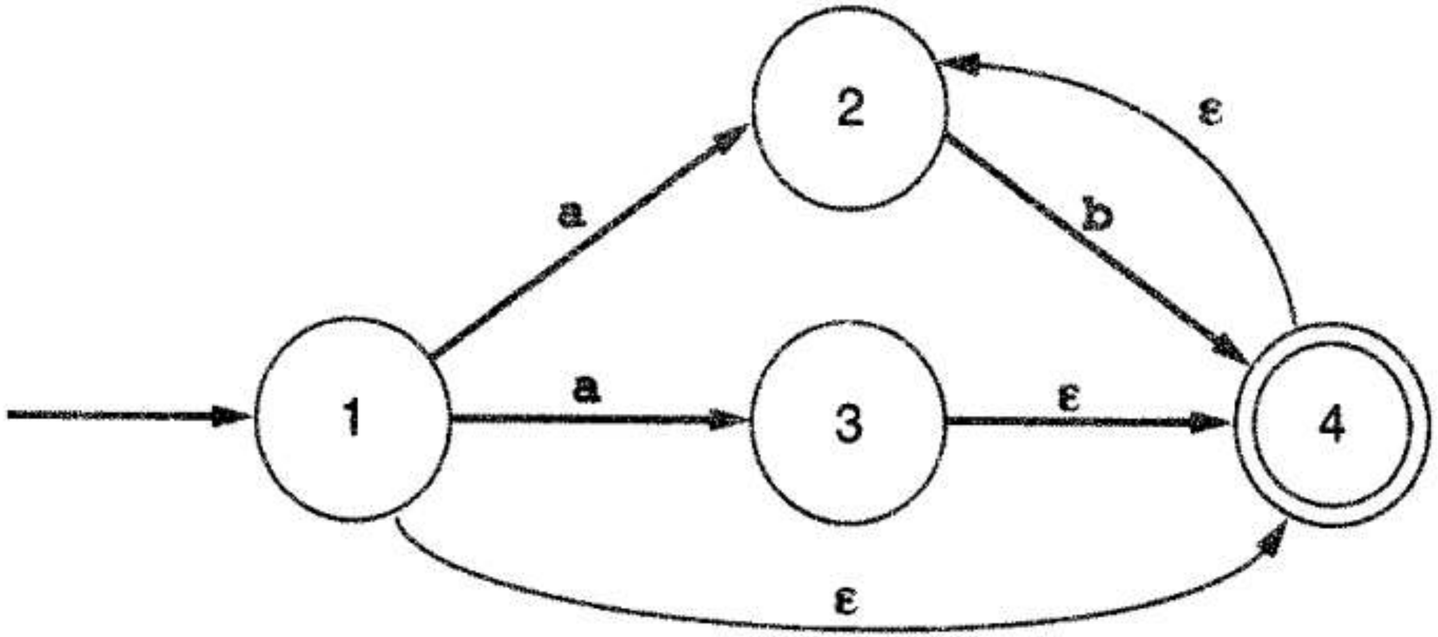
- As transições vazias são úteis por dois motivos, sem usar :

  - pode descrever o casamento de um caractere vazio, ou seja, dispensa a necessidade de um rótulo para a transição



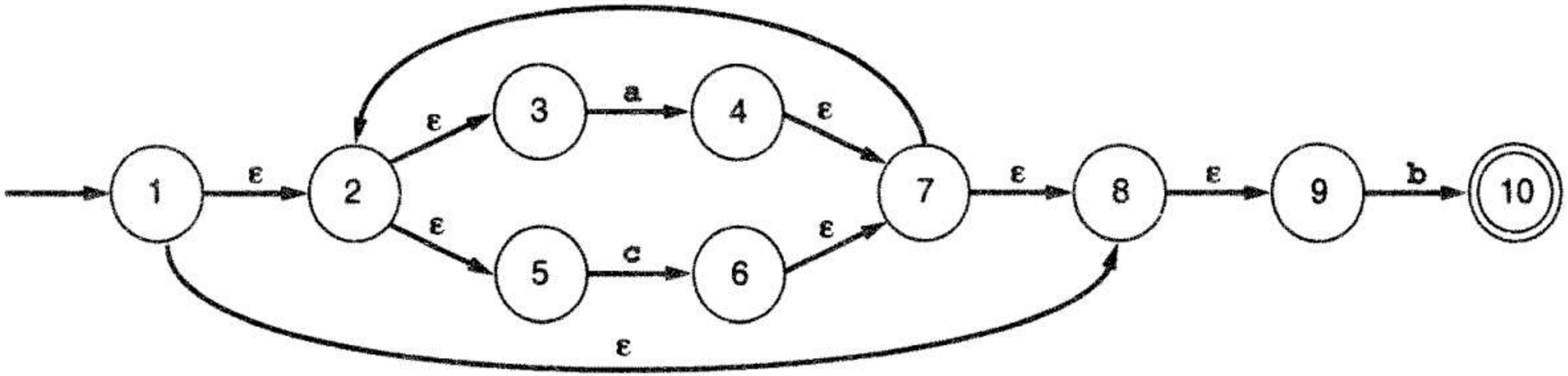
NFA

- Exemplo 1: a cadeia de caracteres *abb* pode ser aceita



NFA

- Exemplo 2: a cadeia de caracteres *acab* pode ser aceita



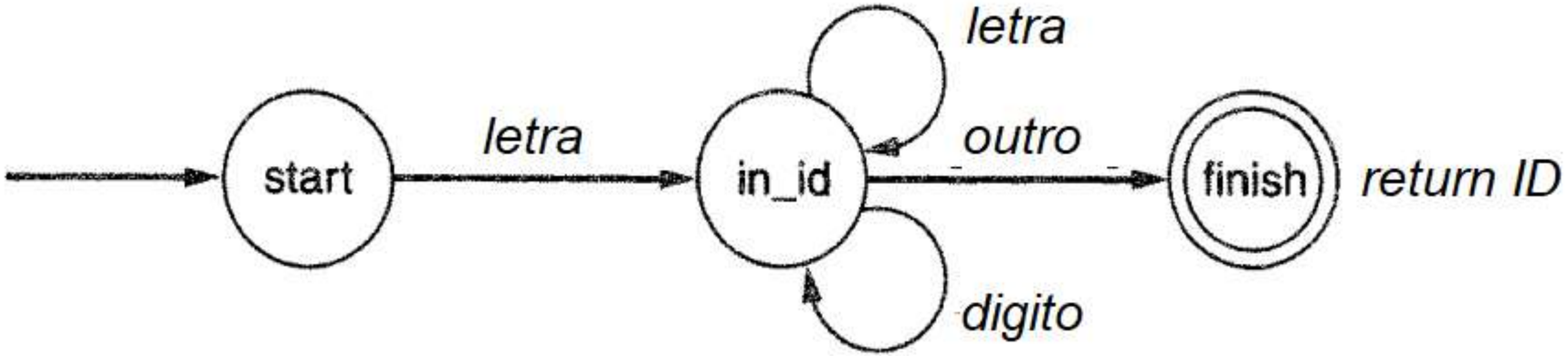
Implementando código para DFA

1. Aninhar os códigos dos estados com os testes

2. Usar uma variável para manter o estado corrente e escrever as transições como múltiplos *if*, em que os comandos do primeiro teste de condição

Implementando código para DFA

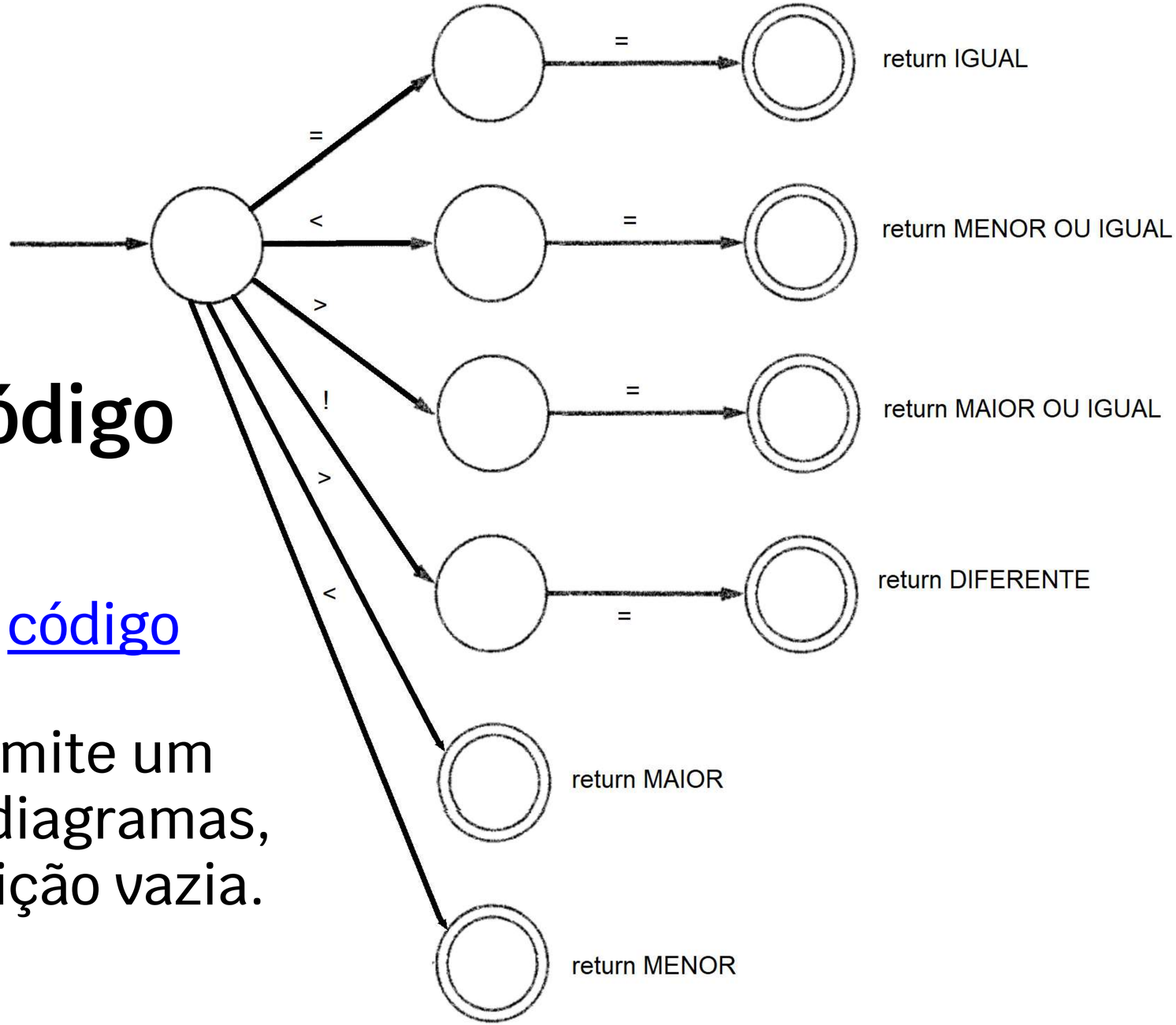
- Exemplo: identificador – código



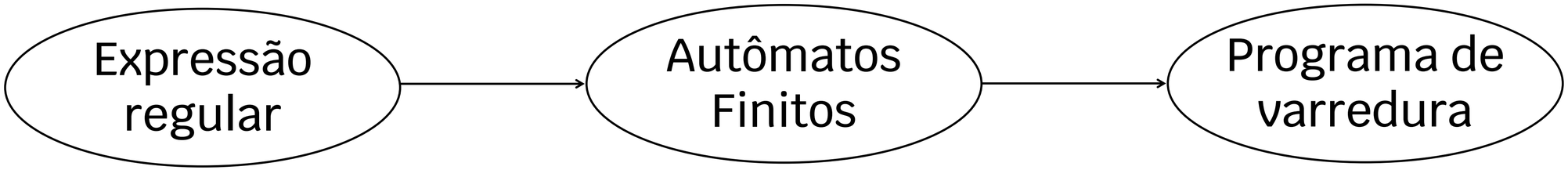
Implementando código para DFA

- Exemplo: comparador – código

Observe que o DFA não admite um diagrama, mas múltiplos diagramas, pela inexistência da transição vazia.

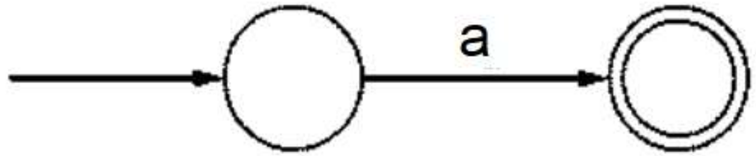


Traduzindo uma expressão regular para um NFA

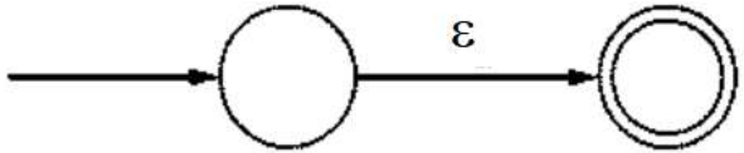


Traduzindo uma expressão regular para um NFA
Construção de Thompson

- De uma expressão regular básica para NFA

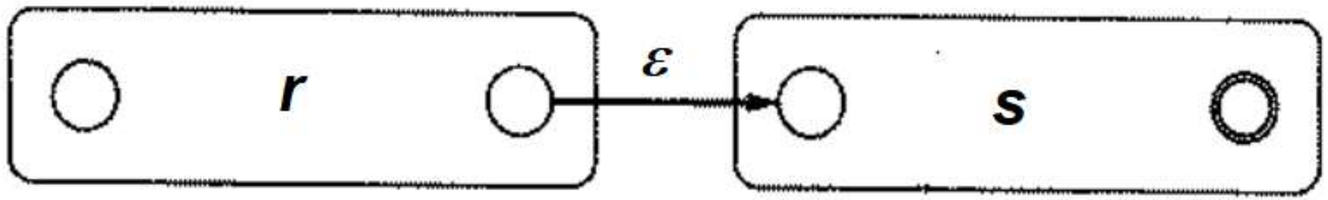


- De uma transição vazia



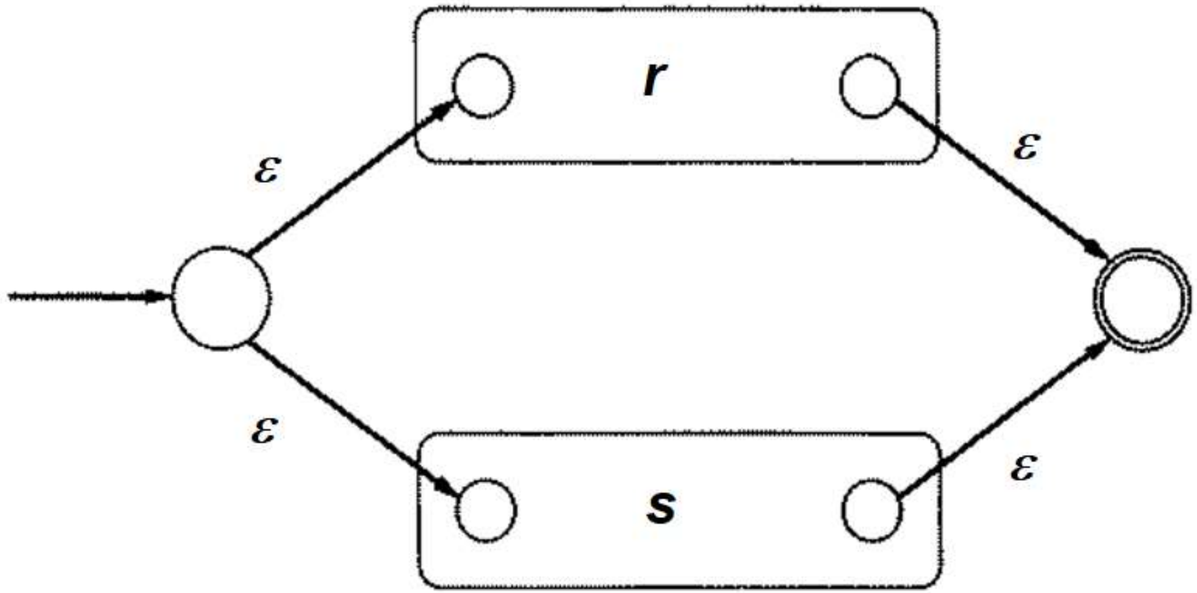
Traduzindo uma expressão regular para um NFA
Construção de Thompson

- De uma concatenação de expressões ***rs*** para NFA



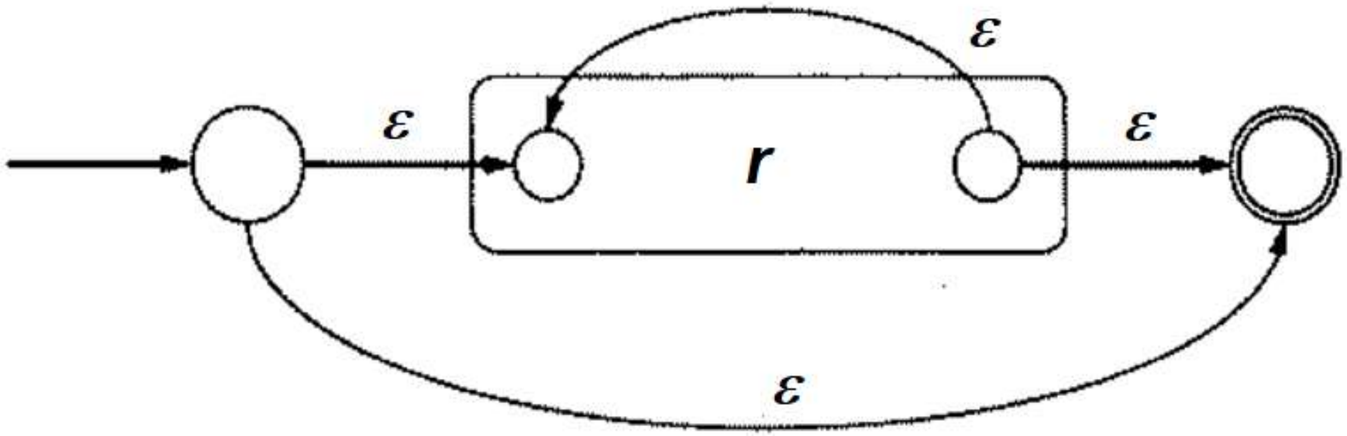
Traduzindo uma expressão regular para um NFA
Construção de Thompson

- De uma escolha entre alternativas de expressões ***r***|***s*** para NFA



Traduzindo uma expressão regular para um NFA
Construção de Thompson

- De repetição de expressões r^* para NFA



Referências

- Louden, Kenneth C.; Compiladores princípios e práticas; Capítulos 2.1 e 2.2; THOMSON

IBMEC.BR

/IBMEC

IBMEC

@IBMEC_OFICIAL

@IBMEC

Ibmec