# Aula 6 – Reconhecimento de Tokens


Assis Tiago
assis.filho@unicap.br

- ***Compiladores***

# *Introdução*

- Objetivos
  - Usar **Autômatos Finitos** (AF) para reconhecer tokens.
  - Conhecer AF que reconhecem os principais tokens das linguagens de programação.
  - Compreender as **diferentes abordagens** para **reconhecimento** de palavras reservadas.
  - Identificar as **diferentes formas para implementar** AF.

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# *O papel do Analisador Léxico*

- **Funções do Analisador Léxico:**
  - **Ler os caracteres do código-fonte.**
    - Remover espaços em branco, tabulações e comentários.
  - **Agrupar os caracteres em lexemas e classificá-los.**
    - Detectar erros e relacionar com a posição no programa.
  - **Gerar a lista de tokens (marcas)**
  - **Manipular a Tabela de Símbolos**

# *Especificação de tokens*

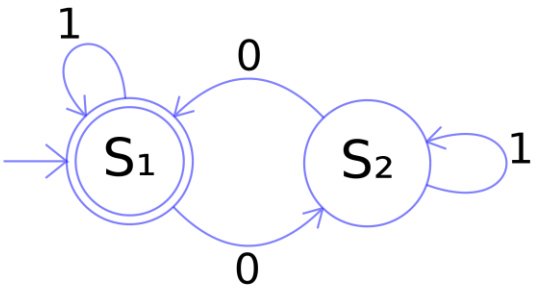
- A especificação dos tokens é feita usando **Expressões Regulares (ER)**
- As ERs definem os **padrões** que especificam como os símbolos do alfabeto podem ser agrupados para formar cadeiras (**lexemas**), que são classificadas como diferentes tipos de **tokens**.

# *Especificação de tokens*

- **Alfabeto:** conjunto finito de símbolos (letras, dígitos, sinais de pontuação, etc.);
- **Cadeia, sentença** ou **palavra:** sequência finita de caracteres retirada de um alfabeto;
- **Linguagem:** conjunto contável de cadeias de um determinado alfabeto

# *Reconhecimento de Tokens*

- AFs são formados por um **conjunto** de **estados** e **arestas** e servem para indicar se uma cadeia é ou não reconhecida.
- Cada **estado representa uma condição** que pode ocorrer durante a análise da entrada na busca por um **lexema que case** com um **padrão**.
- Cada **aresta** é rotulada por **um** ou **mais símbolos** que levam de um estado a outro.



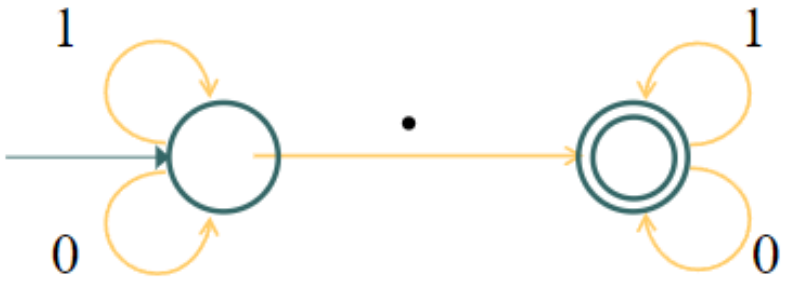
# *Autómatos Finitos*

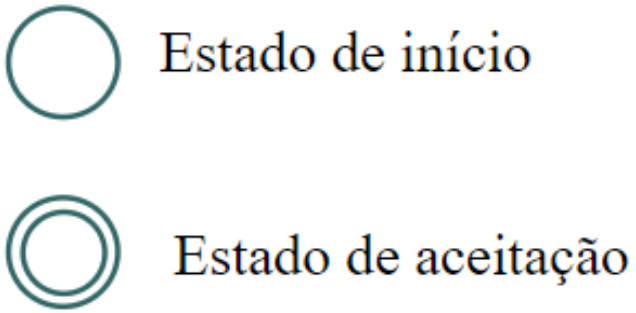
- Conjunto de estados
  - 1 estado de entrada
  - 1 ou mais estados terminais ( ou estados de aceitação)
- Alfabeto de símbolos (inclui o símbolo de string de tamanho zero: 'E'
- Transições entre estados despoletadas pela ocorrência de um determinado símbolo do alfabeto
- Transições rotuladas com símbolos

# Reconhecimento de Tokens

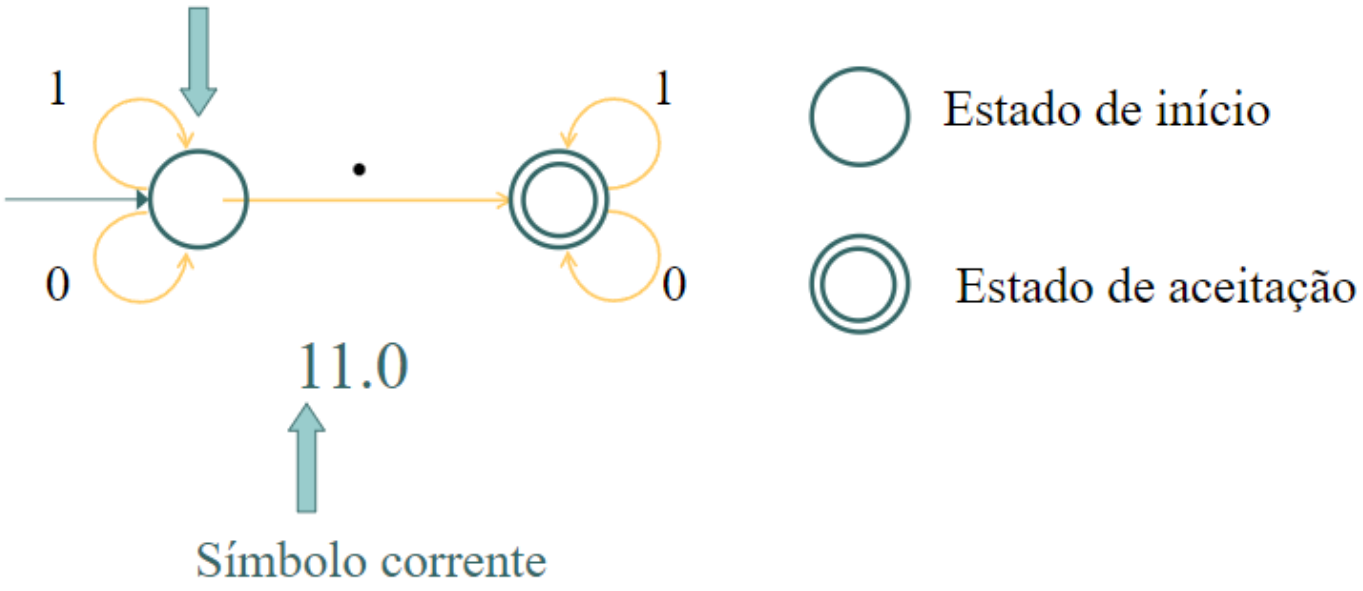
- Exemplo

(0 | 1)*”.”(0 | 1)*

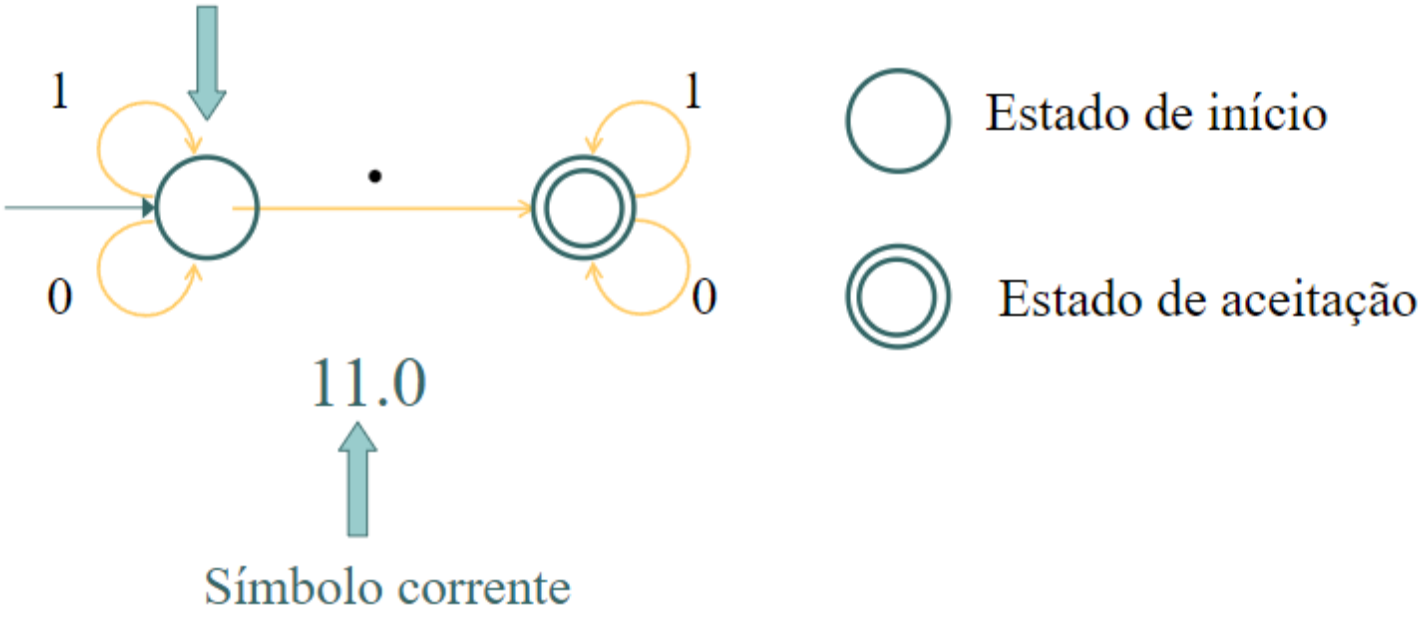


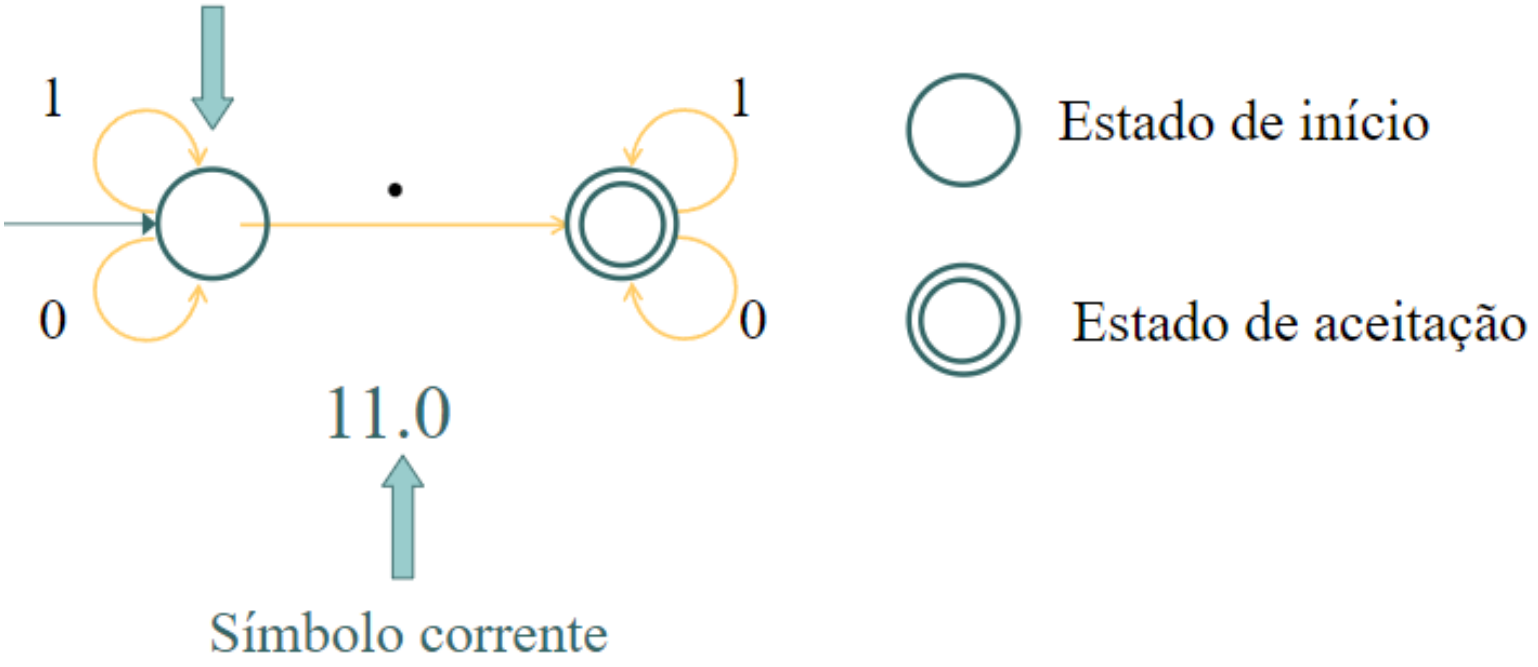
# Reconhecimento de Tokens



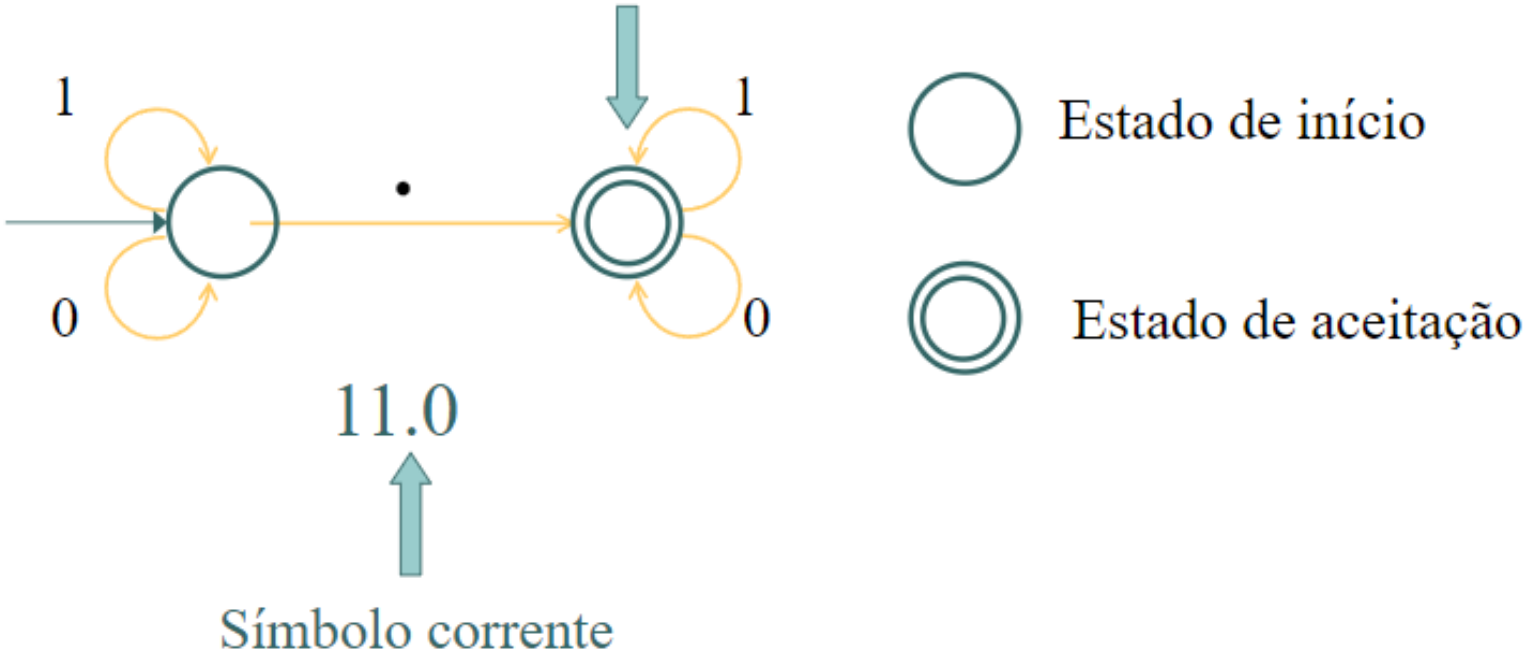
# Reconhecimento de Tokens



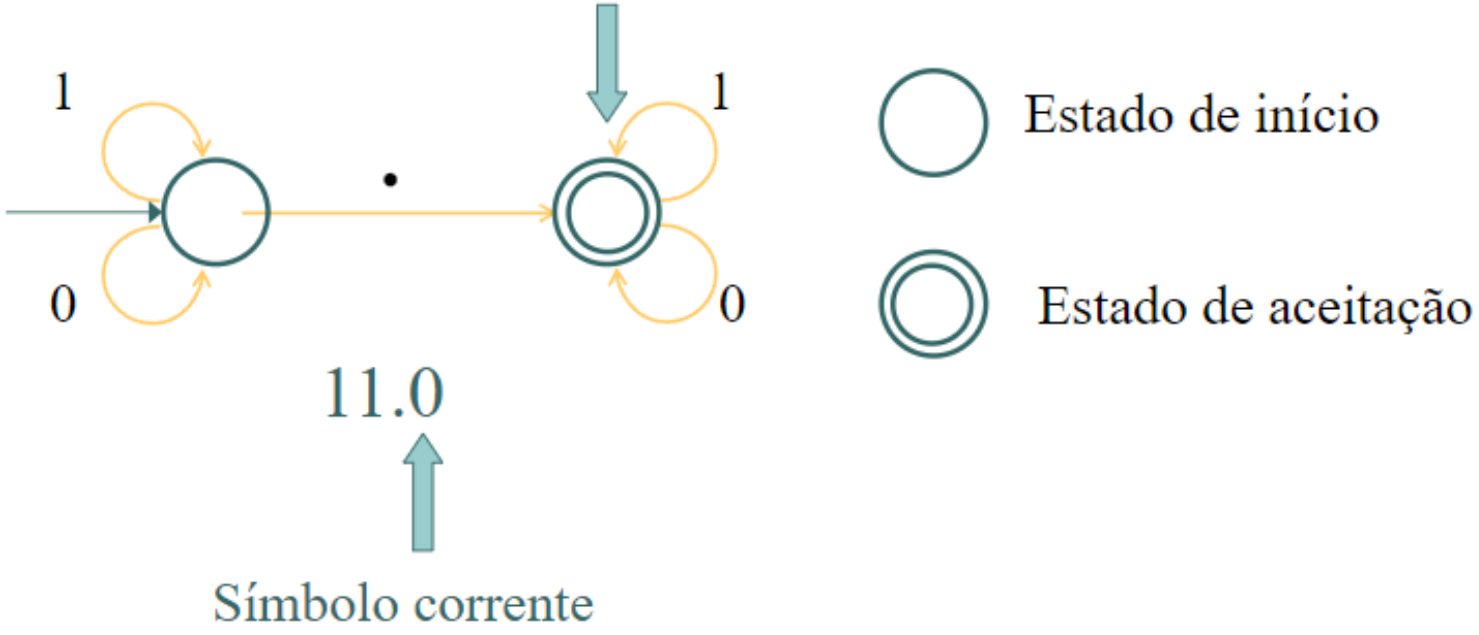
# Reconhecimento de Tokens



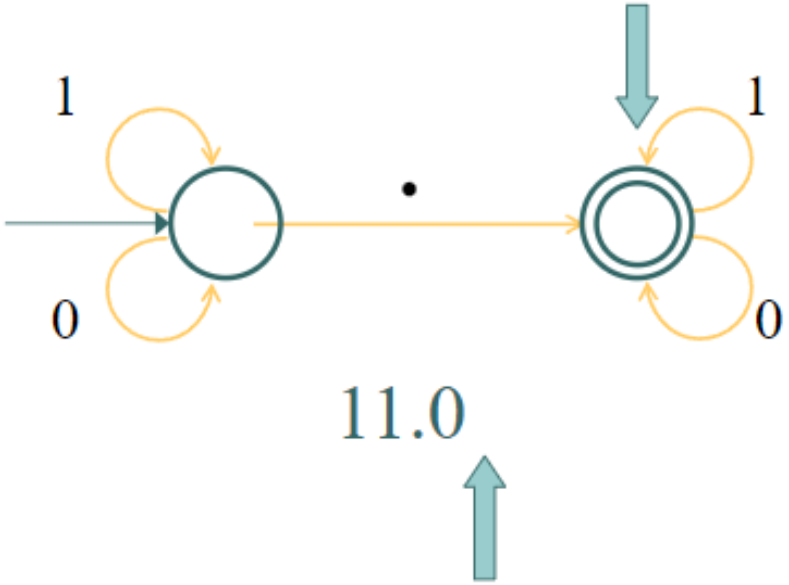
# Reconhecimento de Tokens



# Reconhecimento de Tokens



# Reconhecimento de Tokens



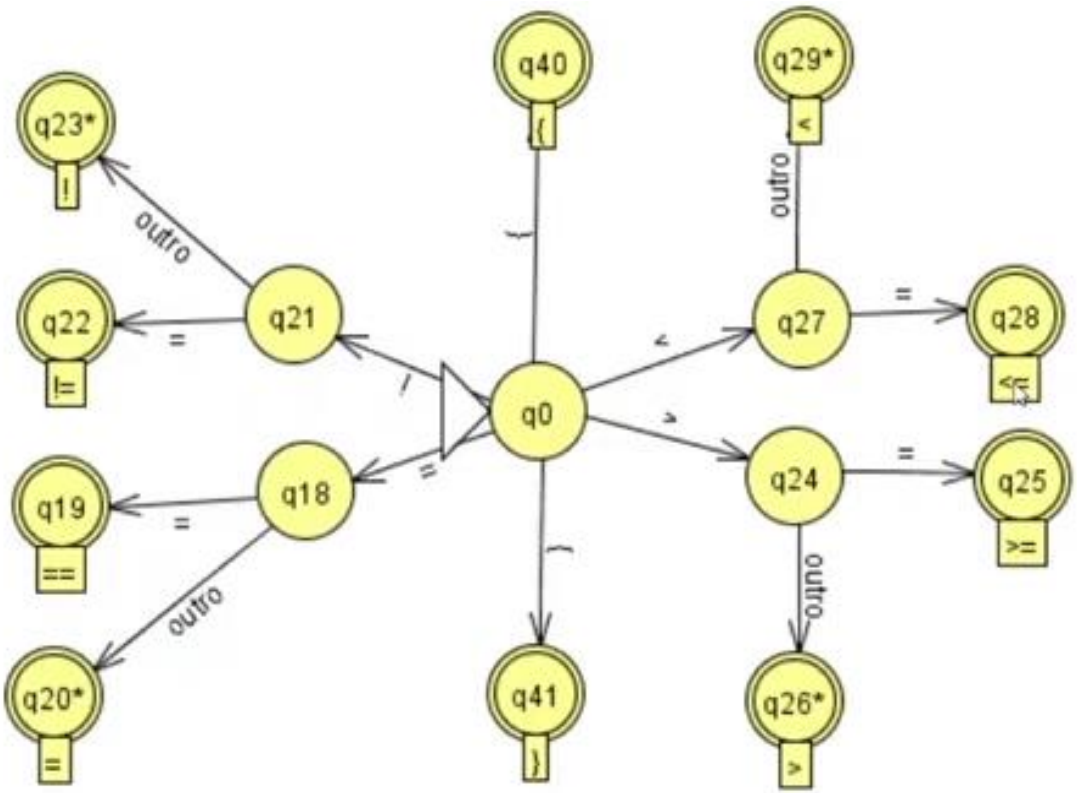

Estado de início



Estado de aceitação

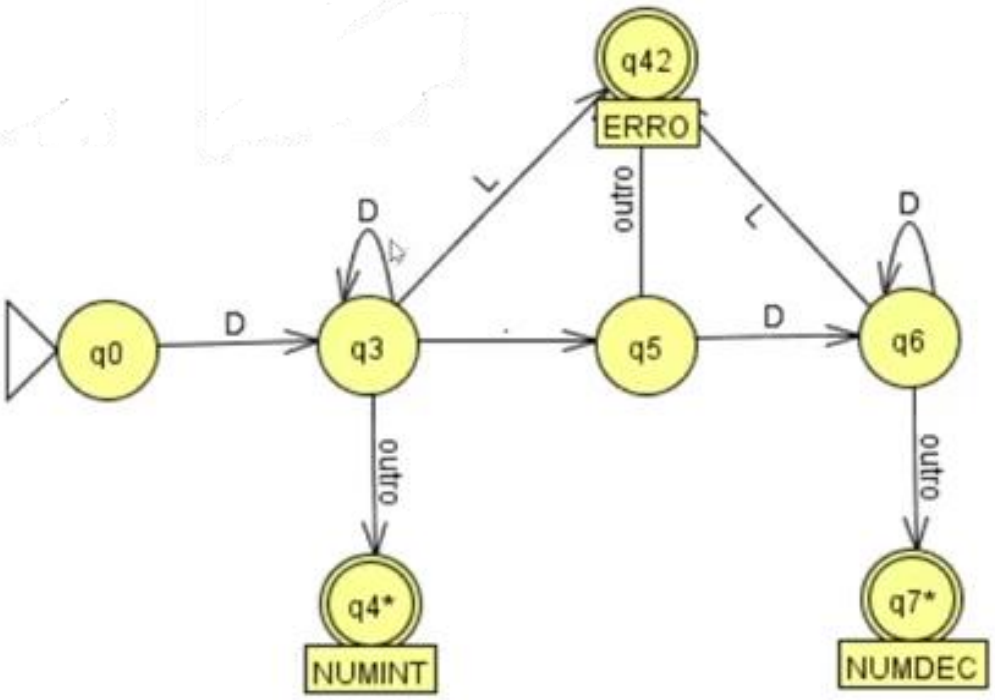
String aceite!

# Reconhecimento de Tokens



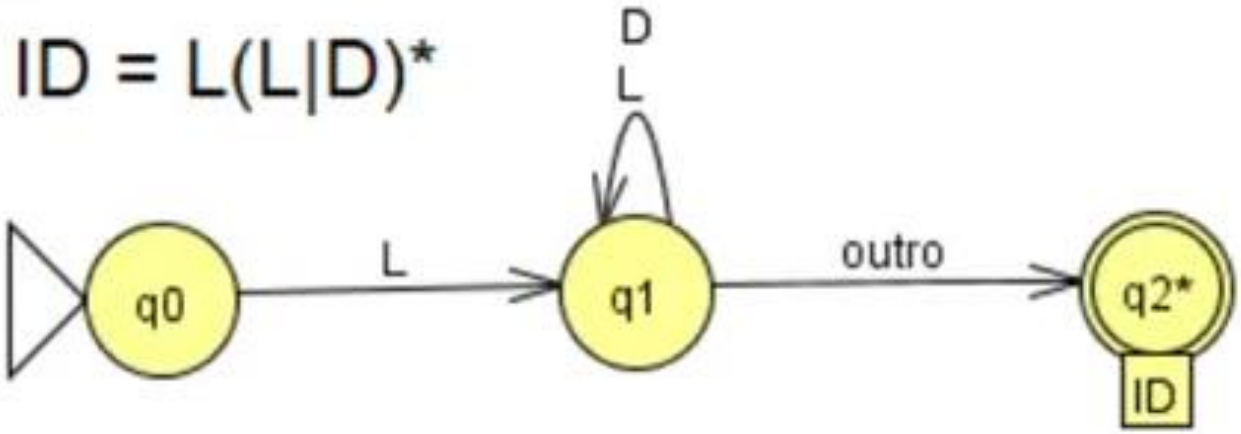
# Reconhecimento de Tokens

- NUMINT = D+
- NUMDEC = D+\.D+



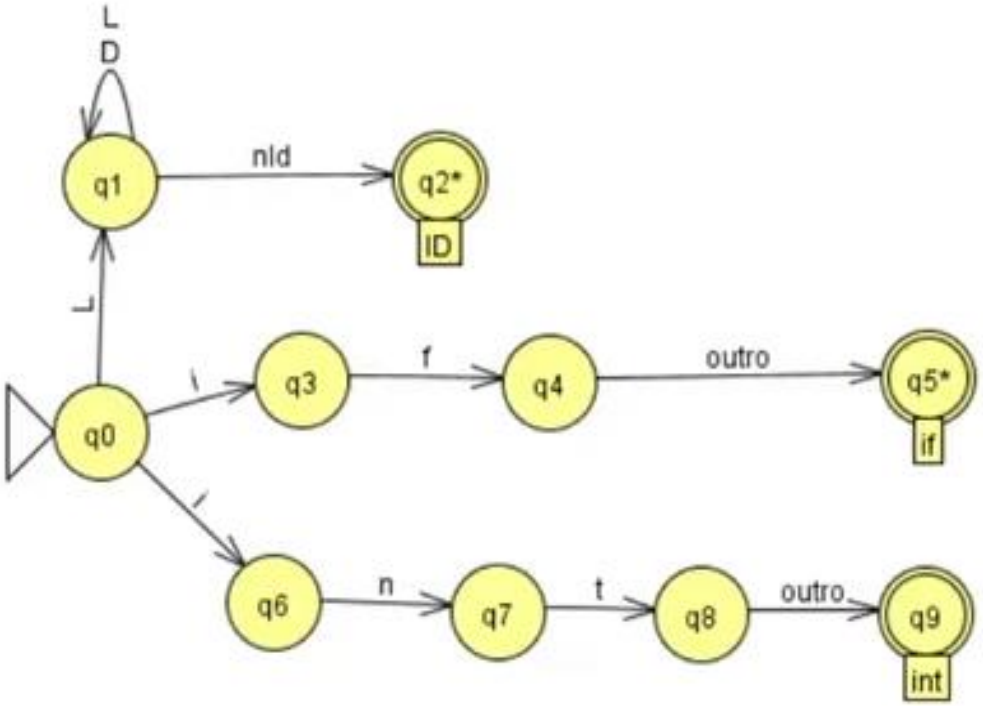
# Reconhecimento de Tokens

- ID = L(L|D)*

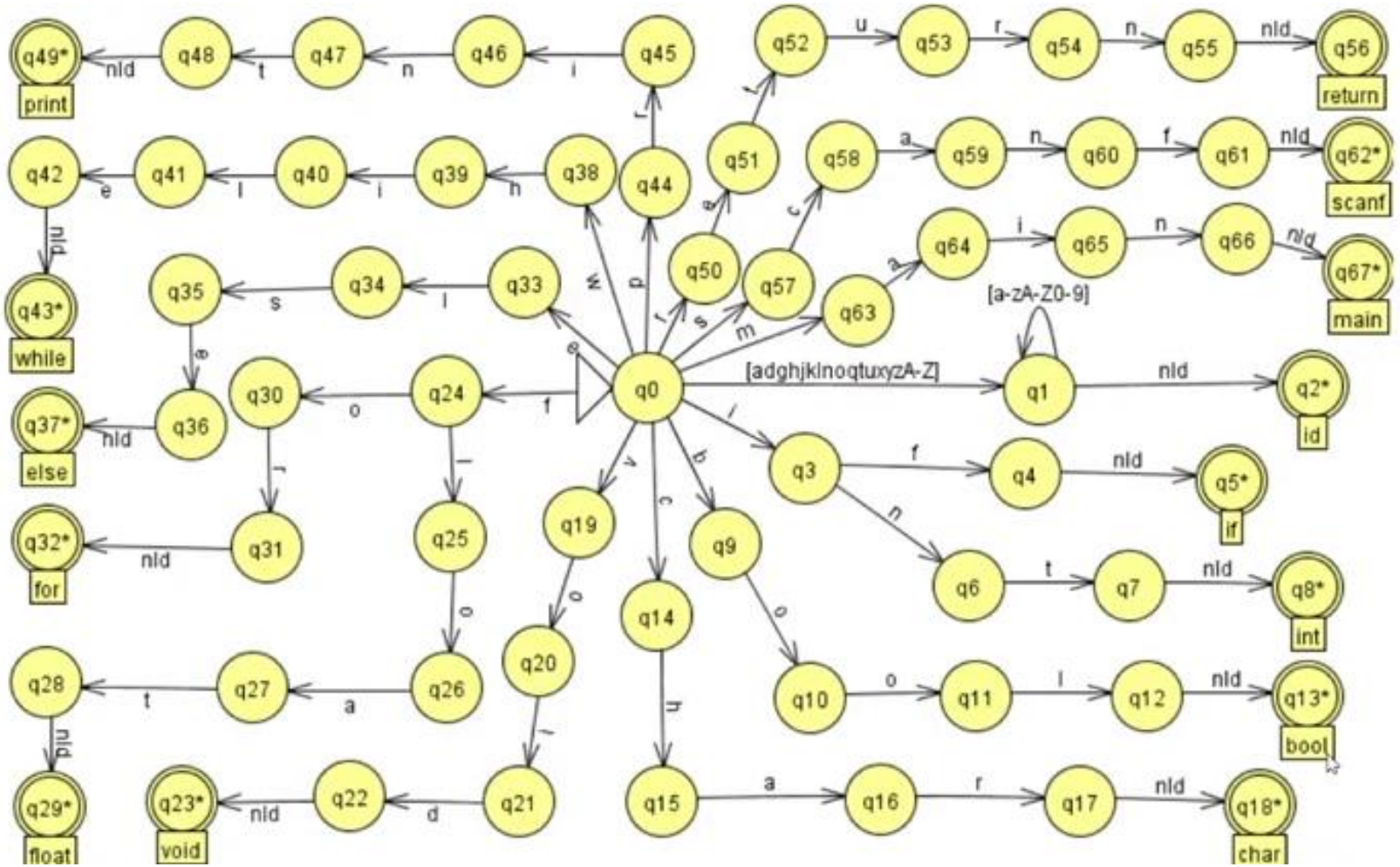


# Reconhecimento de Tokens

- O **problema** das palavras reservadas:

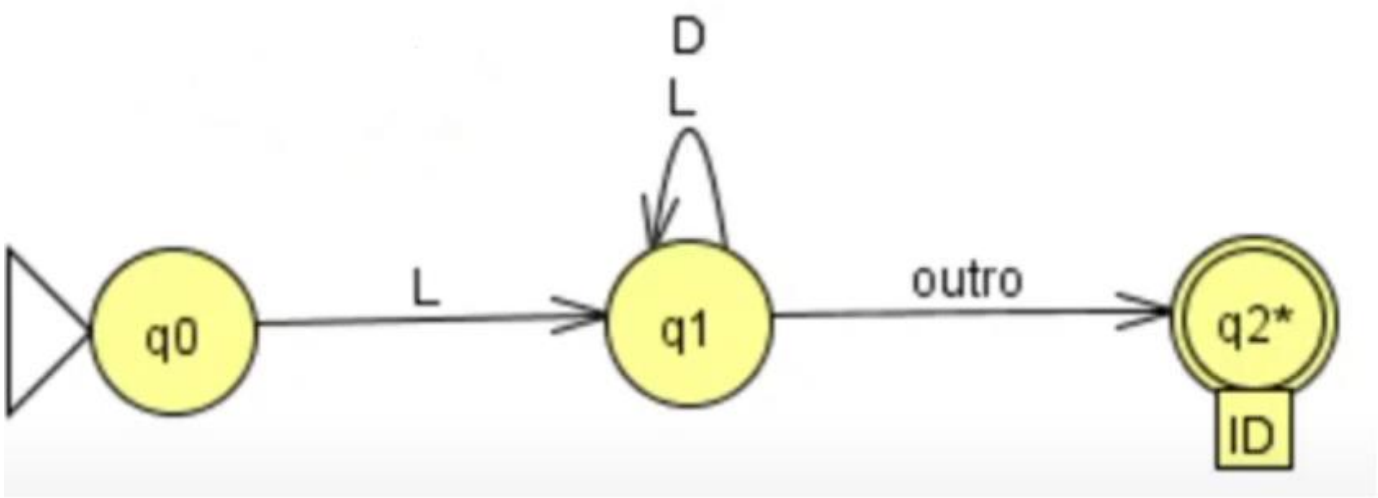


# Reconhecimento de Tokens

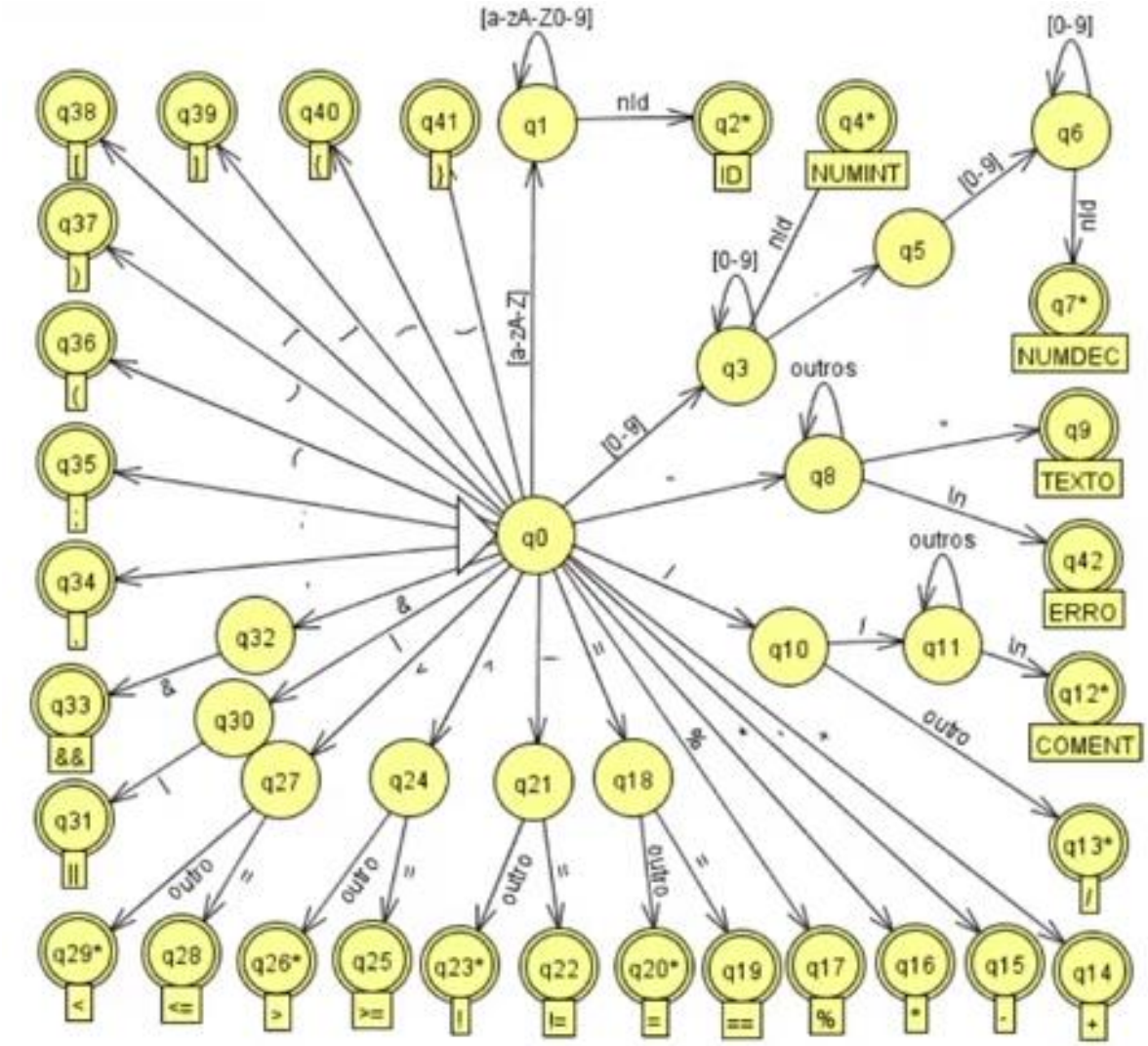


# Reconhecimento de Tokens

- ID = L(L|D)*

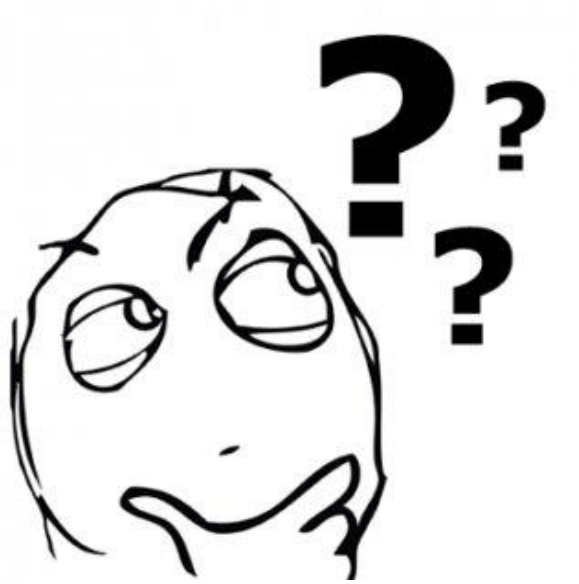


# Reconhecimento de Tokens



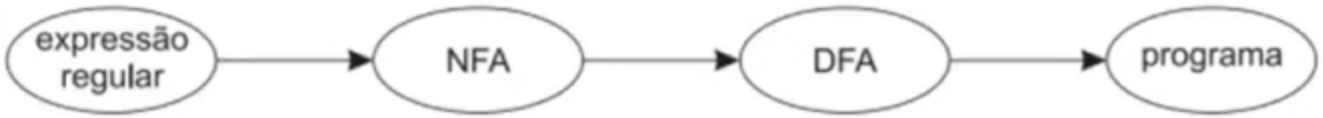
# Reconhecimento de Tokens

- De que forma um analisador léxico pode ser implementado?



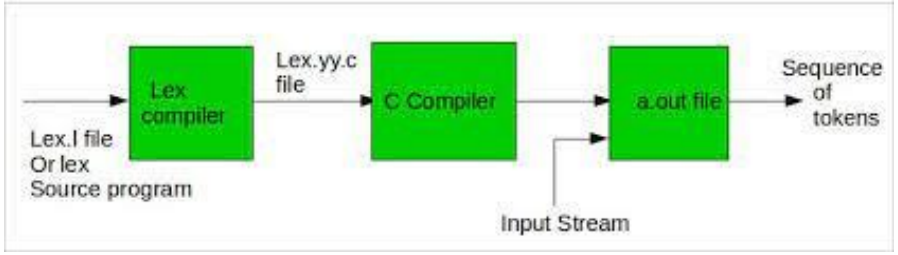
# Reconhecimento de Tokens

- A implementação de um analisador léxico se dá pela conversão de expressão regulares (ER) em autômatos finitos e pela implementação destes autômatos.
  - NFA: Autómato Finito não Determinístico
    - De um determinado estado, a mesma ocorrência pode conduzir a estados distintos
  - DFA: Autómato Finito Determinístico
    - O mesmo que NFA com a seguinte ressalva:
    - De um determinado estado, a ocorrência de um símbolo não pode ter mais do que um laço, e por isso não pode levar a estados diferentes.

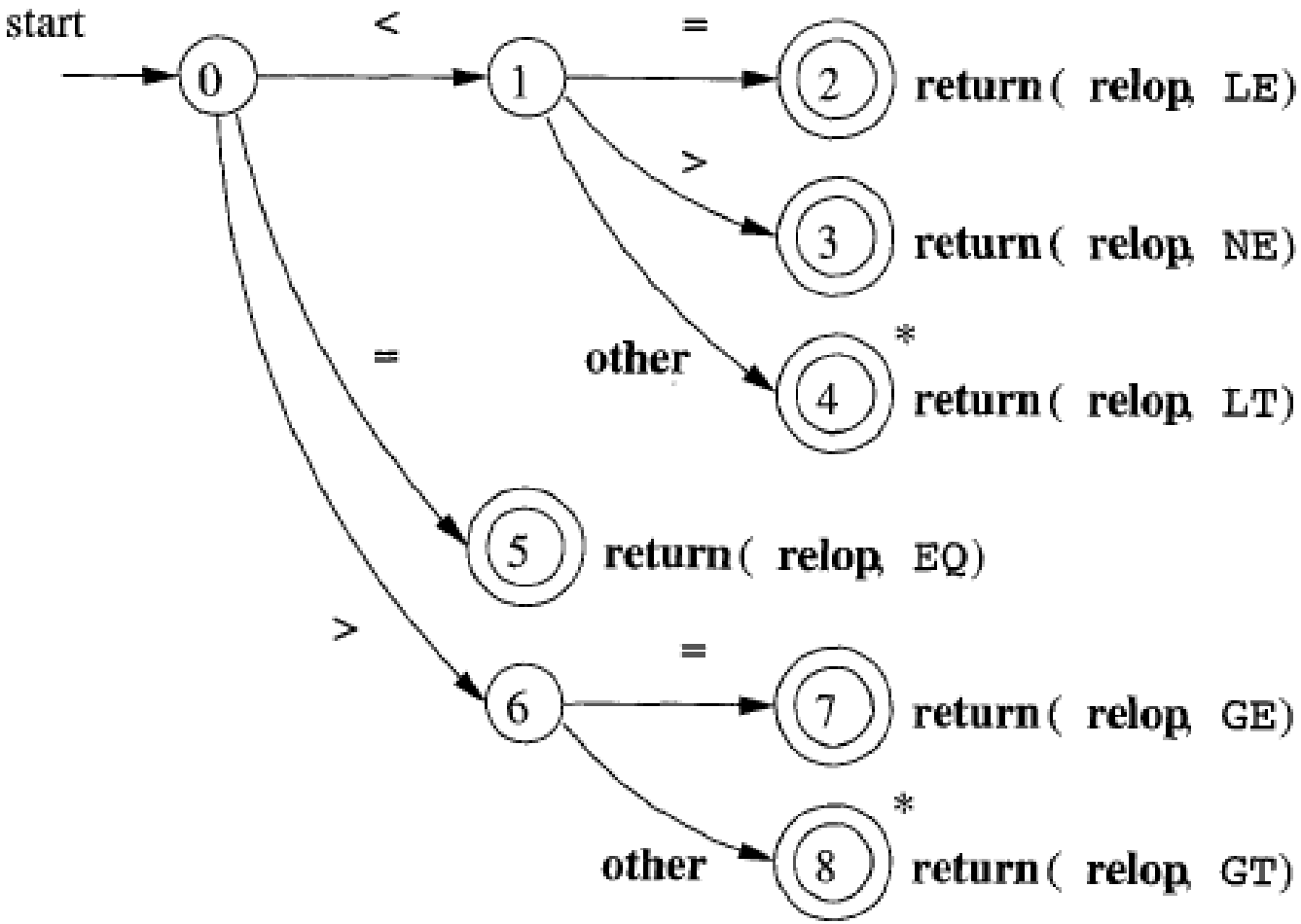


# Reconhecimento de Tokens

- Implementação de um AFD
  - Desvios de condicionais explícitos
  - Tabela de transição
    - Linhas representam estados e colunas os símbolos da entrada.
- Ferramentas como LEX e FLEX automatizam esta tarefa

# Reconhecimento de Tokens



# Reconhecimento de Tokens

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```

# Reconhecimento de Tokens

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | est | L | D | " | / | + | - | * | % | = | ! | > | < | \| | & | , | ; | . | ( | ) | [ | ] | { | } | | \n | \t | Outros |
| 2 | 0 | 1 | 3 | 8 | 10 | 14 | 15 | 16 | 17 | 18 | 21 | 24 | 27 | 30 | 32 | 34 | 35 | 42 | 36 | 37 | 38 | 39 | 40 | 41 | 0 | 0 | 0 | 42 |
| 3 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 42 |
| 4 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 3 | 42 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 42 |
| 6 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 5 | 42 | 6 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 |
| 8 | 6 | 42 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 42 |
| 9 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 8 | 8 | 8 | 9 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 42 | 8 | 8 |
| 11 | 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 10 | 13 | 13 | 13 | 11 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 42 |
| 13 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 12 | 11 | 11 |

# Reconhecimento de Tokens

```
while i < len(linha):
    car = linha[i]
    lexema += car
    coluna = self.__get_coluna(car)
    estado = self.__matriz[int(estado)][int(coluna)]
    if estado in self.__estado_final:
        if estado in self.__estado_retrocesso:
            i-=1
            lexema = lexema[:-1].strip()
        self.__add_lista_tokens(estado,lexema,l)
        estado=0
        lexema=''
    i+=1
```

# Bibliografia


- **Básica:**

· AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi (Et. al.) **Compiladores: princípios, técnicas e ferramentas**. 2. ed. Rio de Janeiro: Pearson Addison, 2008.

· COOPER, K. D.;TORCZON, L. **Construindo Compiladores**. 2. ed: Elsevier, 2014.

· APPEL, Andrew W. **Modern compiler implementation in java**. 1. ed. Cambridge: University Press, 1998.

- **Complementar:**

· LOUDEN, Kenneth C. Compiladores : princípios e práticas. São Paulo Cengage Learning Editores 2004

· SANTOS, Pedro Reis. Compiladores : da teoria à prática. Rio de Janeiro LTC 2018

· SETZER, Valdemar W; MELO, Ines S. Homem de. A construção de um compilador. 2. ed. Rio de janeiro: Campus, 1985. 175 p. : il ISBN 85-7001-187-3.

· SEBESTA, Robert W. Conceitos de linguagens de programação. 11. Porto Alegre Bookman 2018

- AGUIAR, Henrique Manoel Guedes de. Projeto de um compilador transportável para a linguagem edison e implementação dos módulos de análise léxica e sintática. 1. ed. Rio de janeiro, 1983. 110 p. : il

# Aula 6 – Reconhecimento de Tokens


Assis Tiago
assis.filho@unicap.br

- ***Compiladores***