Prof. Dr. Daniel Leal Souza

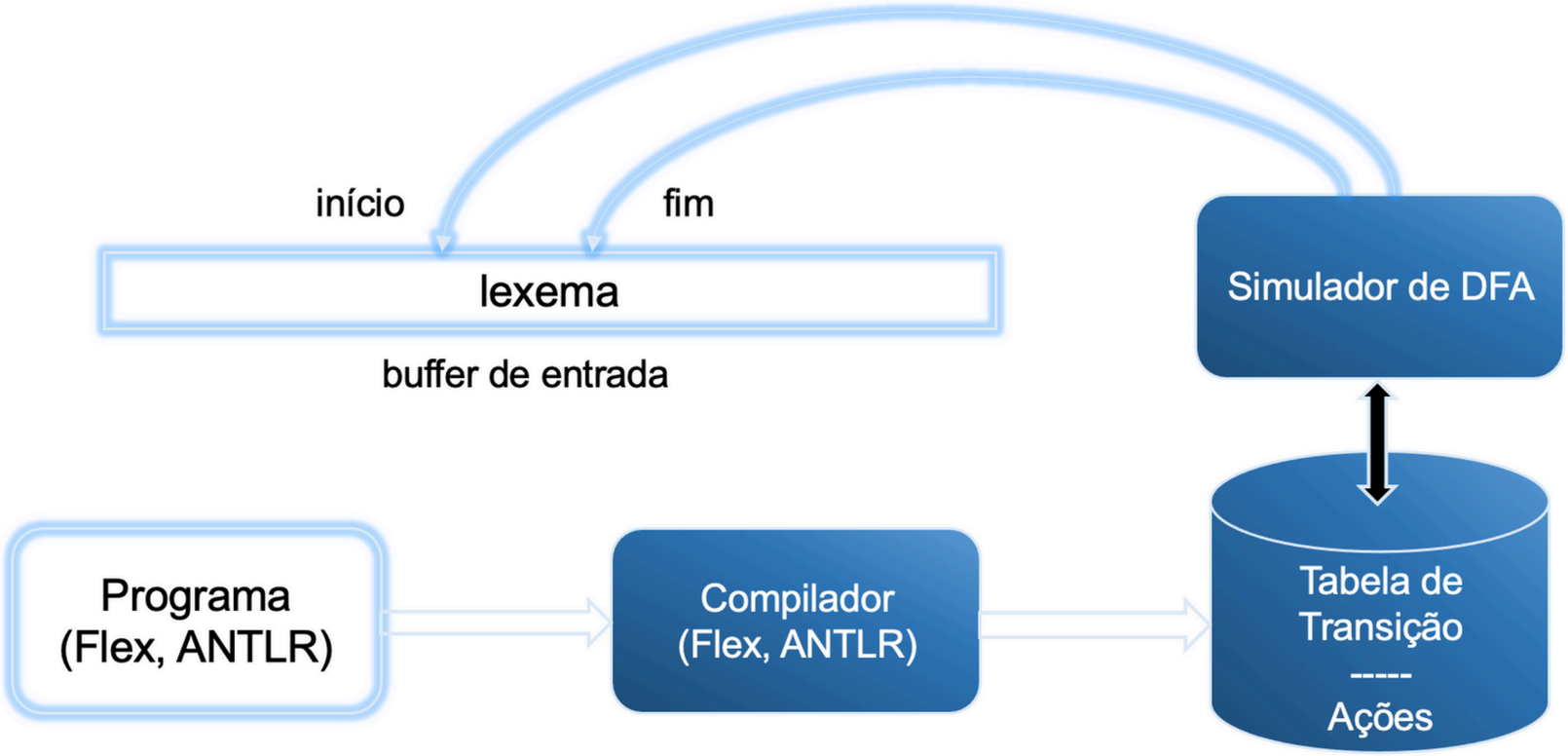
Autômatos e Compiladores (EC8MA)

Compiladores (CC6MA)

daniel.leal.souza@gmail.com
daniel.souza@prof.cesupa.br

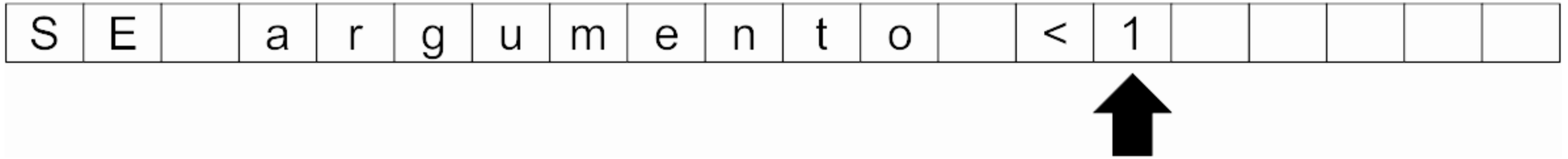
2024

ARQUITETURA (ANALISADOR LÉXICO)



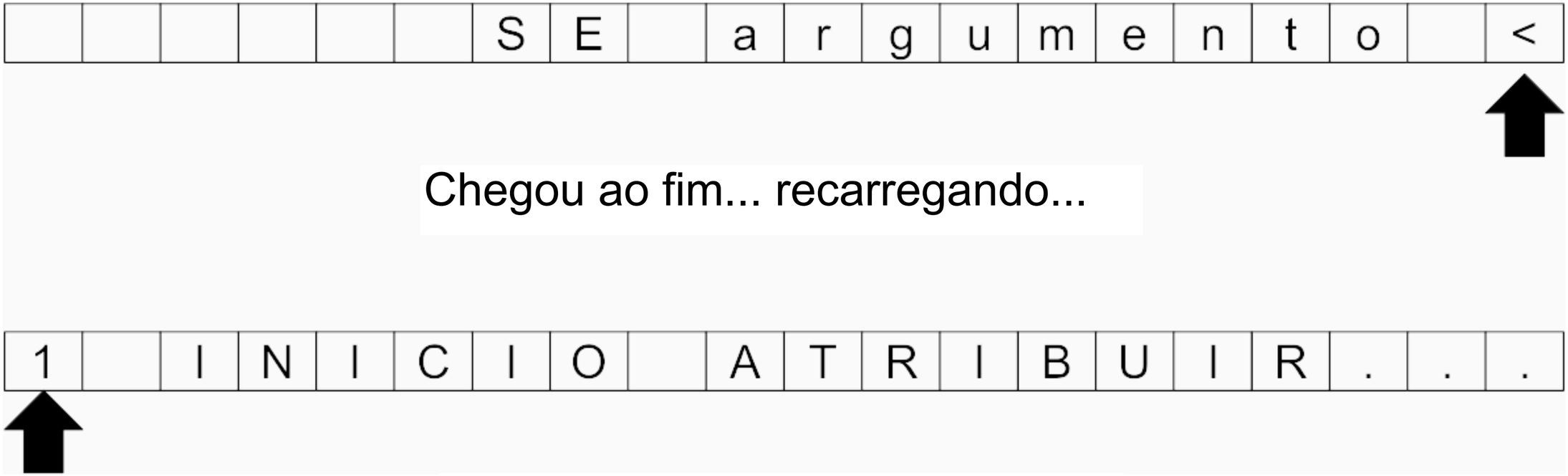
Analisador Léxico (*scanning*)

- **Continuando a implementação...**
  - Ler a entrada caractere por caractere é ruim
    - Em algumas situações, é preciso retroceder ou adiantar
  - Além disso, é ineficiente
  - Solução: usar um buffer
    - Um ponteiro aponta para o caractere atual
    - Sempre que chegar ao fim, recarregamos o buffer
    - Caso necessário, basta retroceder



Analisador Léxico (*scanning*)

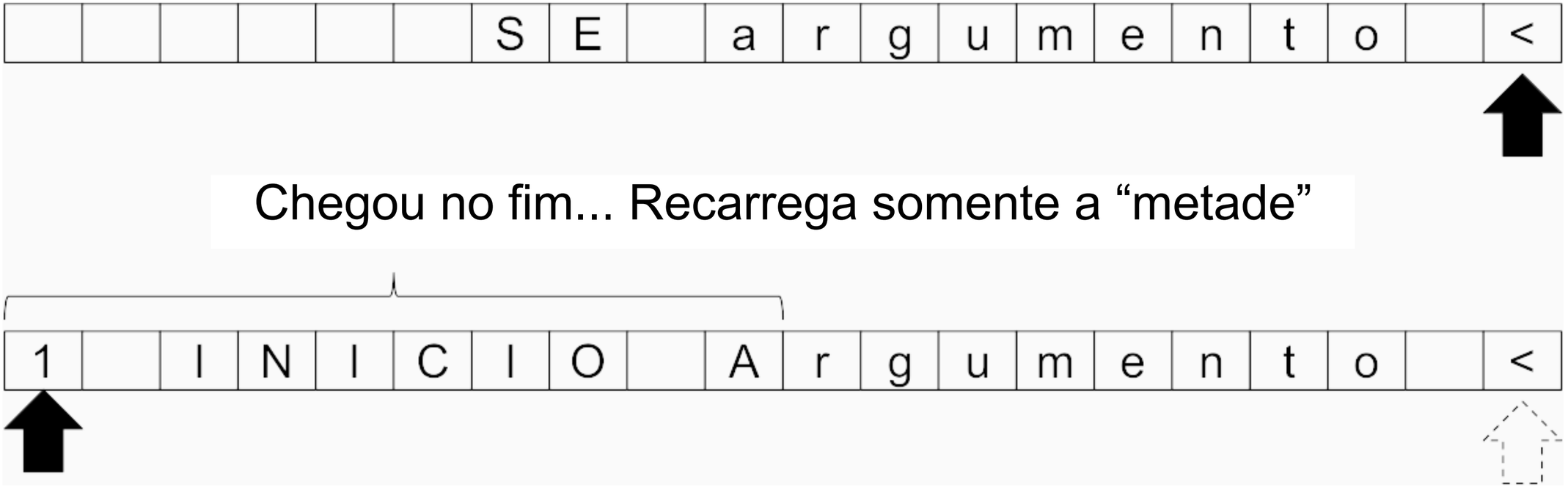
- **Buffer**
  - O buffer único tem um problema
    - Veja o exemplo (extremo) abaixo



Para retroceder, é necessário
recarregar o buffer anterior!

Analisador Léxico (*scanning*)

- **Buffer**
  - Portanto, é comum o uso de um buffer duplo



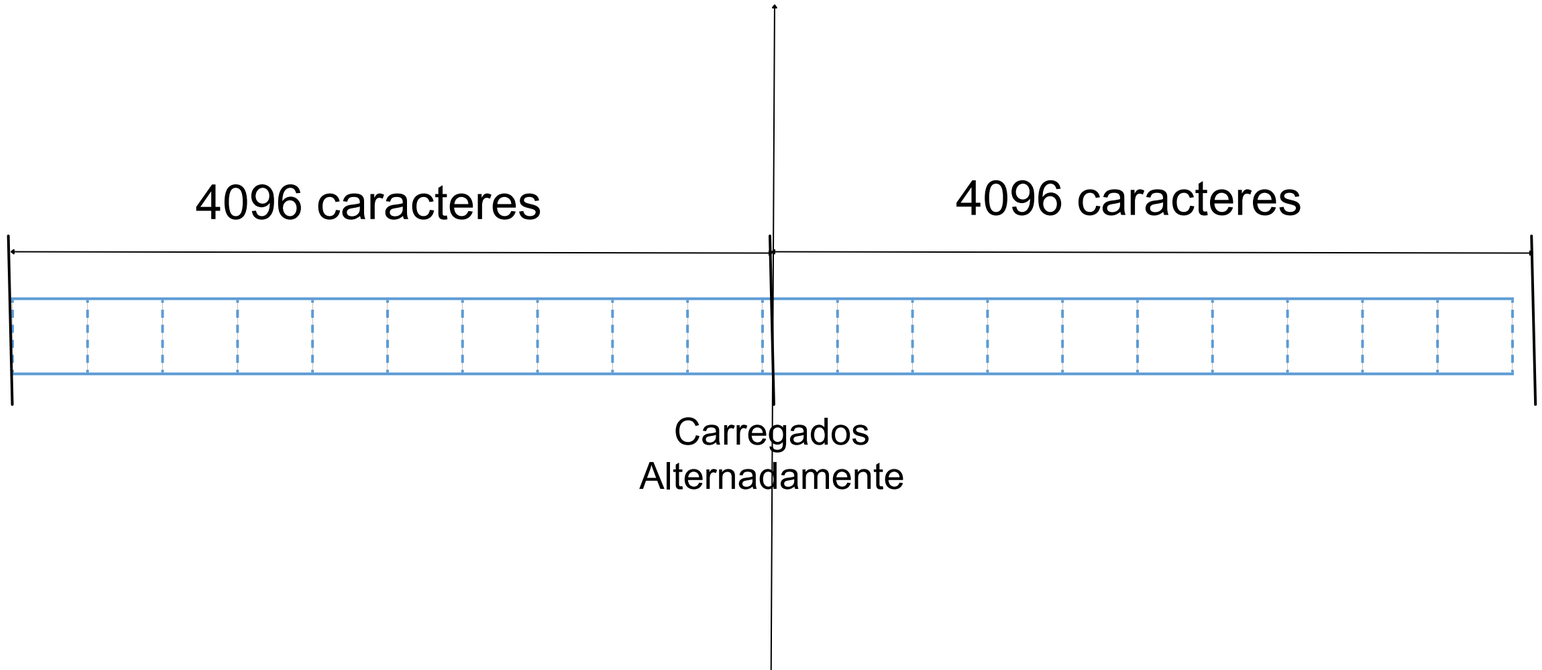
Se precisar retroceder, é só
voltar à outra “metade”

Buffer de Entrada

- Com frequência é necessário examinar um ou mais caracteres à frente, para ter certeza sobre o **token** reconhecido
  Ex.: para reconhecer '>' é preciso ver se o próximo caractere não é '='
- Essa tarefa pode ser realizada com o uso de buffers
  - Acelera a leitura do código
  - Permite usar *lookaheads* grandes com segurança
- Uma técnica eficiente utiliza dois buffers
  - São carregados alternadamente

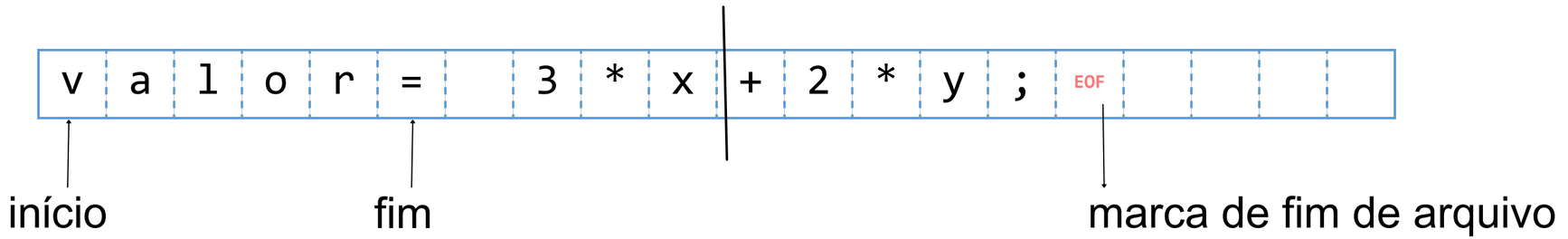
Buffer de Entrada

- Os buffers possuem o mesmo tamanho N
  - O valor de N corresponde ao tamanho de uma leitura no disco
    - Comumente, o tamanho de um bloco do disco é 512 bytes
    - Ler vários blocos de uma vez diminui a quantidade de interrupções da CPU
    - A maioria dos sistemas fazem leituras de 4096 bytes

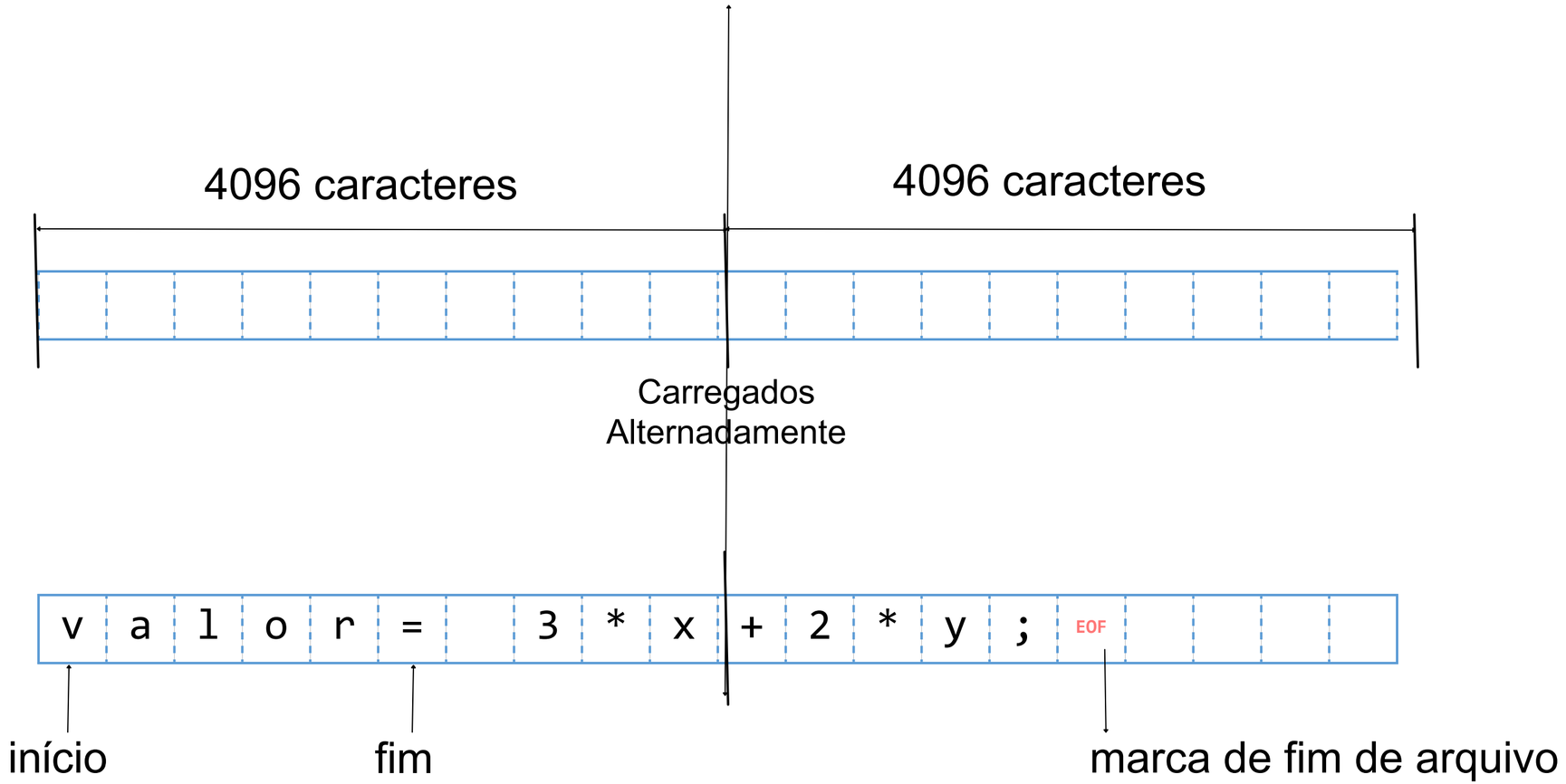


Buffer de Entrada

- Os buffers utilizam dois ponteiros
  - O ponteiro início marca o início de um lexema
  - O ponteiro fim lê adiante até que haja um casamento com um padrão
    - Ao encontrar um casamento, fim retorna para o último caractere do lexema
    - Após construção do token, início avança para o caractere seguinte ao lexema



Buffer de Entrada

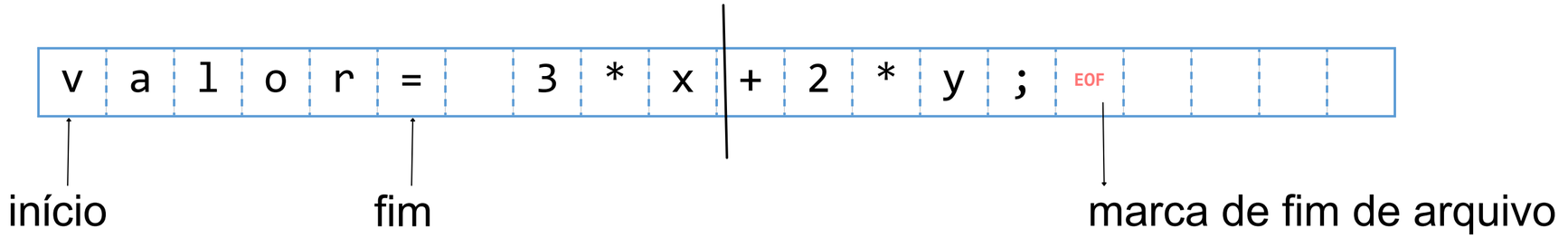


Buffer de Entrada

- Avançar o apontador de fim requer
  - Testar se o fim do buffer foi atingido e em caso positivo
    - Carregar o outro buffer
    - Mover o ponteiro fim para o início do buffer recém carregado
- Espaço em buffer pode se esgotar
  - Tamanho do lexema + avanço adiante for maior que N

Se “x” no exemplo fosse o início de um lexema muito grande, que não pudesse ser encontrado sem ler menos de N caracteres a frente, acarretaria na substituição do primeiro buffer por novos dados, o que invalidaria o ponteiro início.

Uma solução para tratar cadeias muito longas (normalmente literais string) é tratar estas cadeias como a concatenação de cadeias menores.



Demonstração 2

Analisador Léxico (*scanning*)

- Como traduzir os padrões em código da linguagem ALGUMA?

Padrão	Sigla
DECLARACOES, ALGORITMO, INTEIRO, REAL, ATRIBUIR, A, LER, IMPRIMIR, SE, ENTAO, ENQUANTO, INICIO, FIM	3 primeiras letras
*, /, +, -	OpArit
<, <=, >=, >, =, <>	OpRel
E, OU	OpBool
:	Delim
(,)	AP / FP
Seq. de letras ou números que começam com letra minúscula	Var
Sequências de dígitos (sem vírgula)	NumI
Sequências de dígitos (com vírgula)	NumR
Sequências de caracteres envolta por aspas	Str

Analisador Léxico (*scanning*)

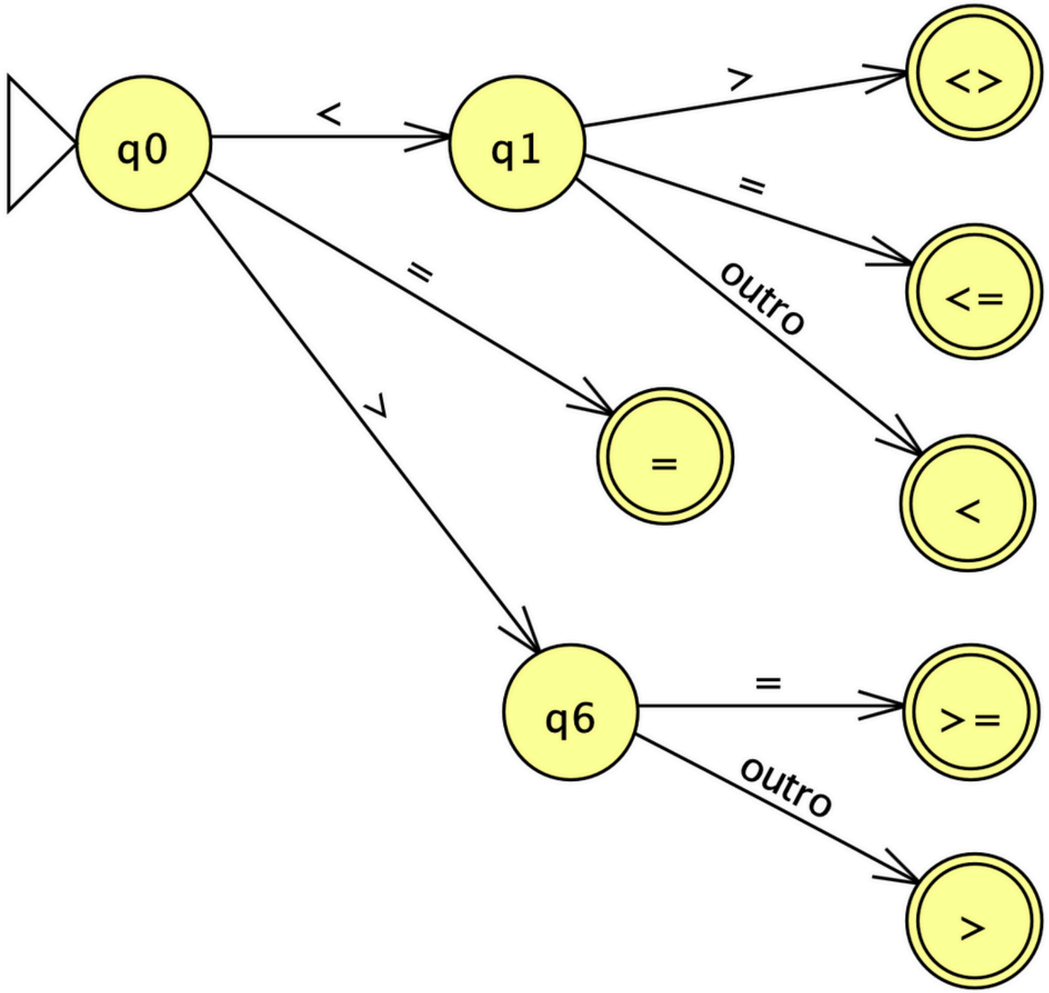
- **Reconhecimento de padrões**
  - Opção 1
    - Basta implementar a lógica utilizando nossa criatividade
    - E ferramentas disponíveis na maioria das linguagens de programação
    - Diagramas podem ajudar

Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - Diagrama de estados
    - Modelo visual que facilita a implementação da lógica do reconhecimento
  - Autômato finito determinístico
    - Com indicação de retrocesso no final

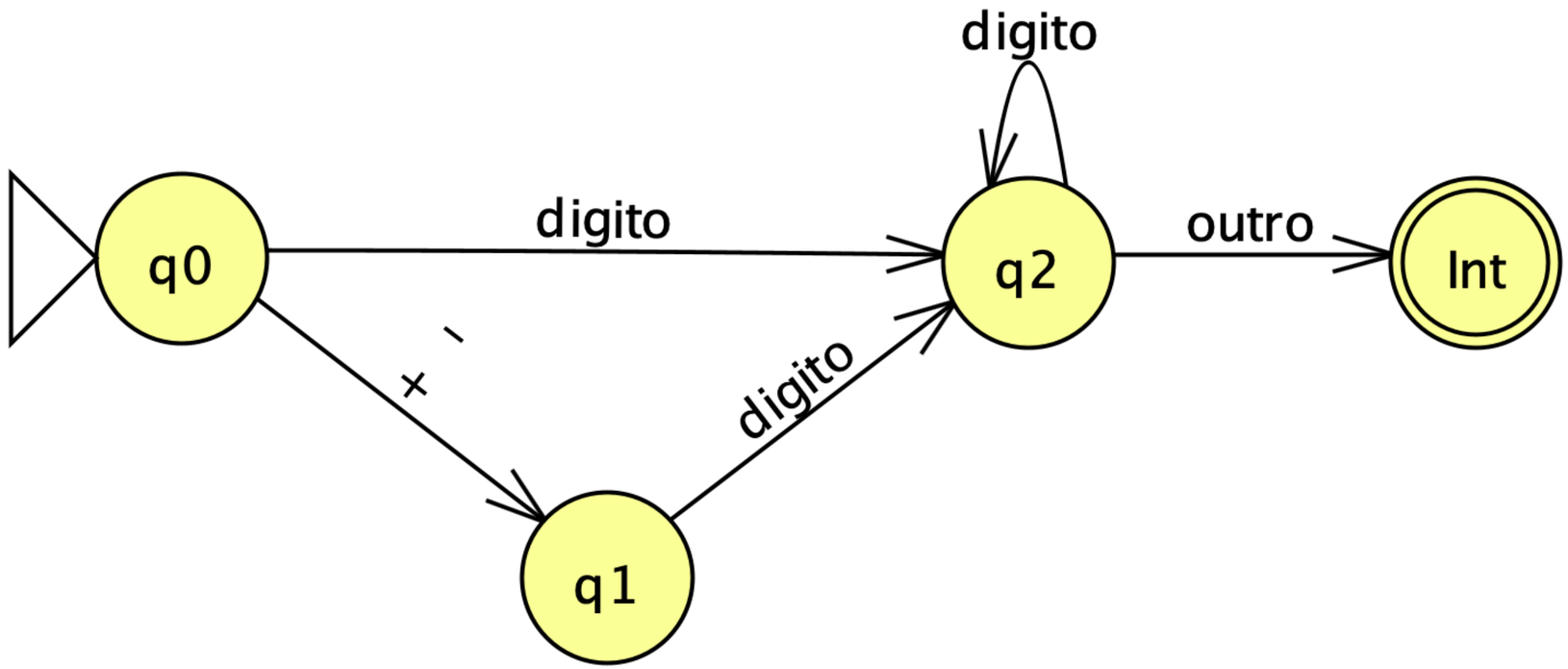
Analisador Léxico (*scanning*)

- **Diagramas de transição**



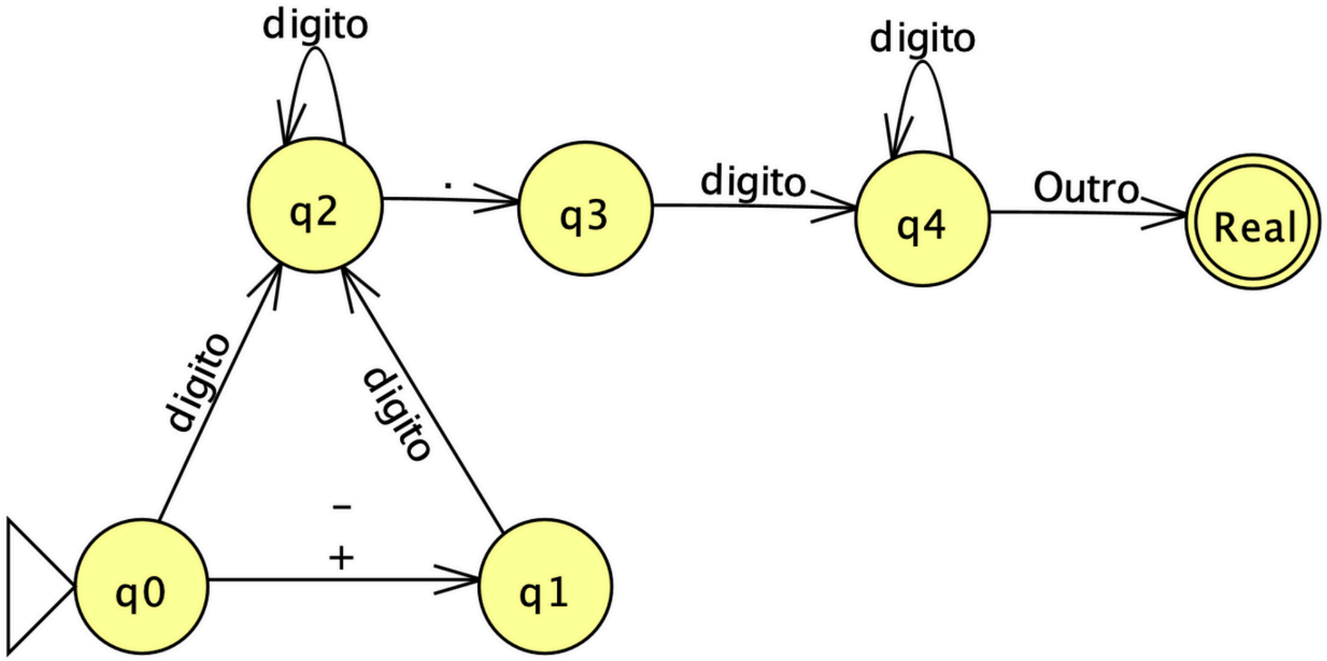
Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - AFD para reconhecer:
    - Números inteiros (NumInt)
      - +1000, -2, 00231, 2441



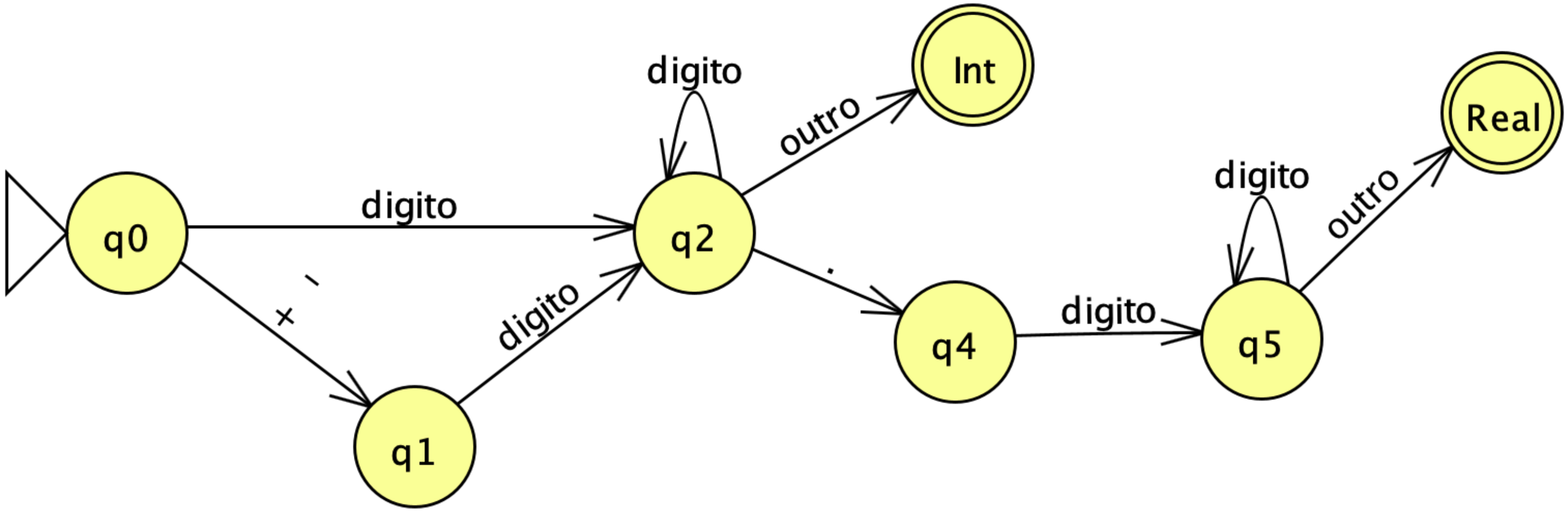
Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - AFD para reconhecer:
    - Números reais (NumReal)
      - +1000.0, -0.50, 0.314
      - Obs: .50, +.22, 30. NÃO são válidos



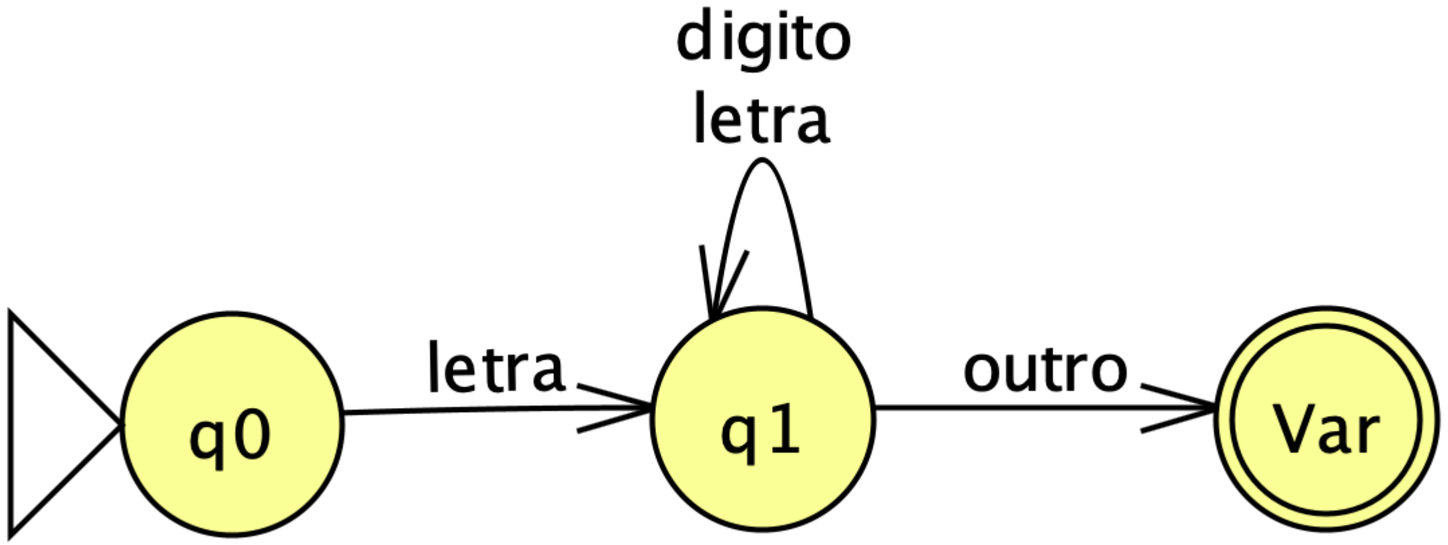
Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - Juntando os diagramas de reconhecimento de números:



Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - AFD para reconhecer:
    - Identificadores (Var, na linguagem ALGUMA)

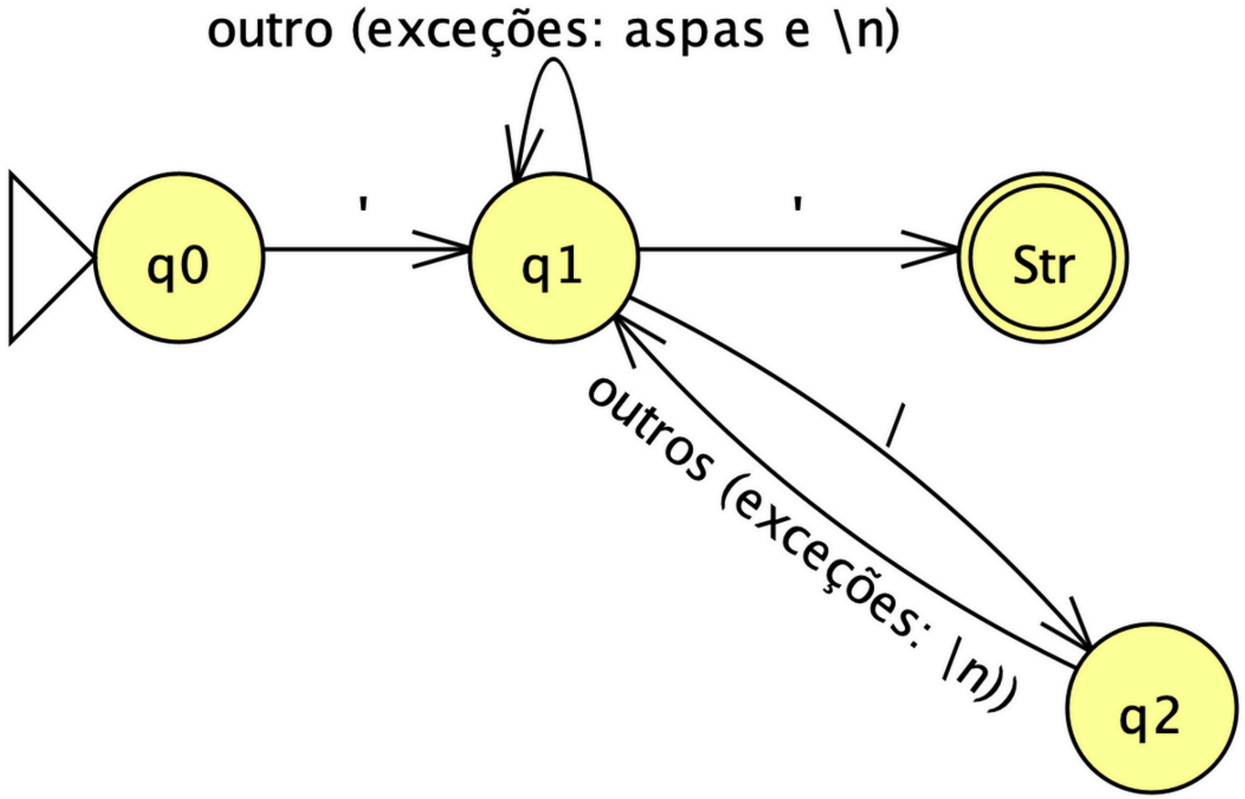


Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - AFD para reconhecer:
    - Cadeia: sequência de caracteres delimitados por aspas simples
      - Ex: ‘Testando’, ‘Uma cadeia qualquer’, ‘#$$A$ASD’
      - Obs: para permitir que as aspas simples sejam usadas dentro de uma cadeia, utilize o caractere de escape
        - Ex: ‘Cadeia que usa \’ aspas \’ simples dentro’
    - Obs: não pode haver quebra de linha (\n) dentro de uma cadeia

Analisador Léxico (*scanning*)

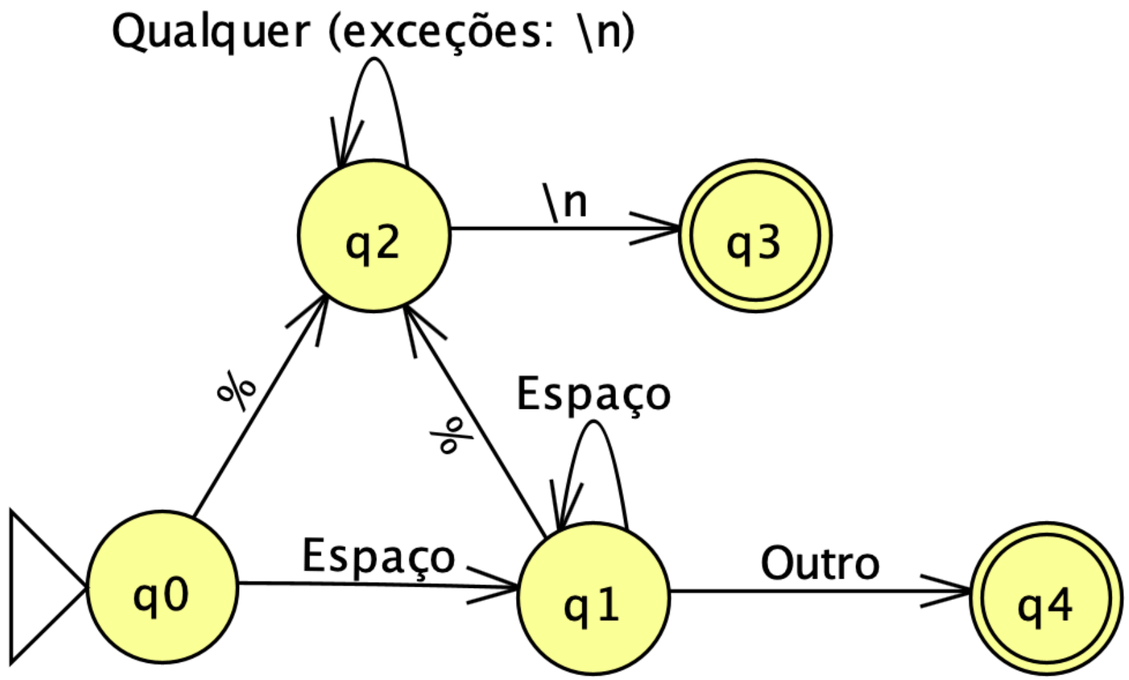
- **Diagramas de transição**
  - AFD para reconhecer:
    - Cadeia (String): sequência de caracteres delimitados por aspas simples



Note que aqui não é necessário retroceder

Analisador Léxico (*scanning*)

- **Diagramas de transição**
  - AFD para reconhecer:
    - Comentários (tudo entre % e \n)
    - Espaços em branco (incluindo \n, \t, etc)
    - Obs: faça em um único diagrama



Analisador Léxico (*scanning*)

- **Implementação**
  - Queremos um método para “zerar” o lexema atual
    - E outro para “confirmar”
  - Depois implementamos cada padrão em um método
    - E definimos uma cascata de regras:
    - Ordem lógica (Ex: identificadores depois de palavras-chave)
  - O método proximoToken testa regra a regra na ordem
    - Para cada padrão:
      - Se for bem sucedido, “confirmar”
      - Caso não seja, “zerar”

Referências

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compiladores - princípios, técnicas e ferramentas. Pearson, 2007.

José Neto, João. Apresentação à compilação. 2ª edição. Rio de Janeiro: Elsevier, 2016.

Notas de aula do professor Daniel Lucrédio - UFSCar.