unesp

Linguagens de Programação

Aula 5

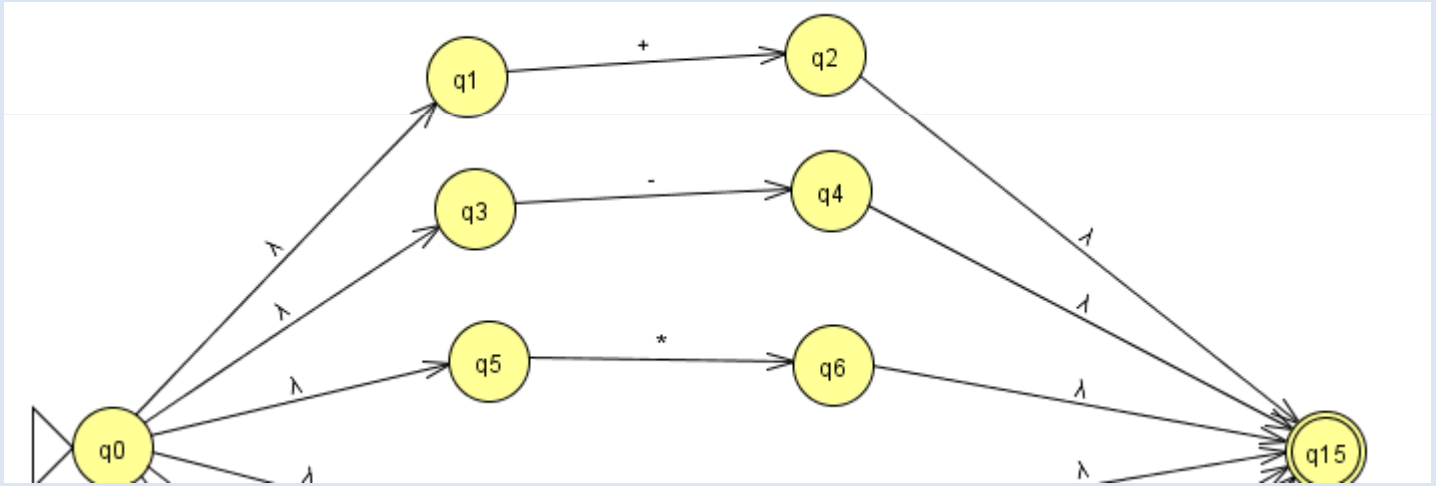
Celso Olivete Júnior

olivete@fct.unesp.br

Na aula anterior

- Ambiguidade
- Análise léxica
  - Exercício calculadora

Na aula anterior
AF calculadora



Na aula de hoje

- Análise léxica – implementação
- Gramática e reconhecedores
- Variáveis:
  - Nomes
  - Vinculações (binding)
  - Verificação de tipo e
  - Escopo

Projeto de um analisador léxico

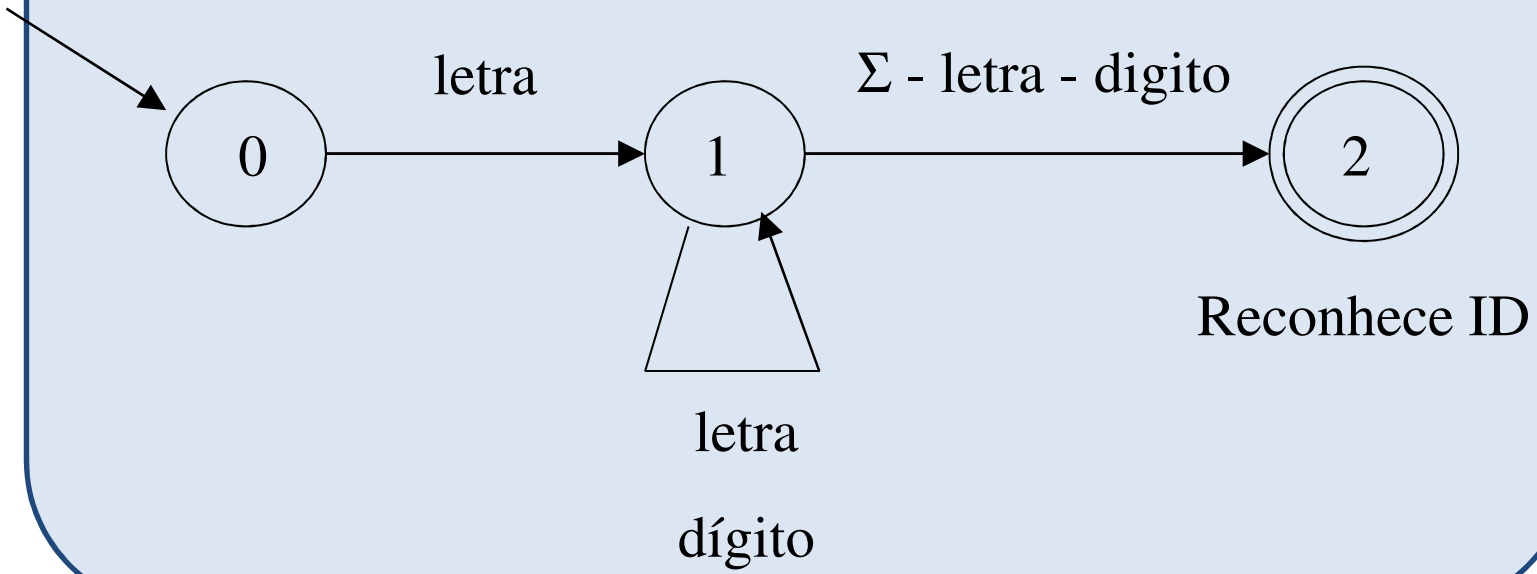
- Existem duas formas básicas para implementar os autômatos: usando a tabela de transição ou **diretamente em código**

Estilo de implementação
diretamente em código

- Cada ***token*** listado é codificado em um número natural
- Deve haver uma variável para controlar o **estado** corrente do autômato e outro para indicar o estado de **partida** do autômato em uso
- Uma função **falhar** é usada para desviar o estado corrente para um outro autômato no caso de um estado não reconhecer uma letra

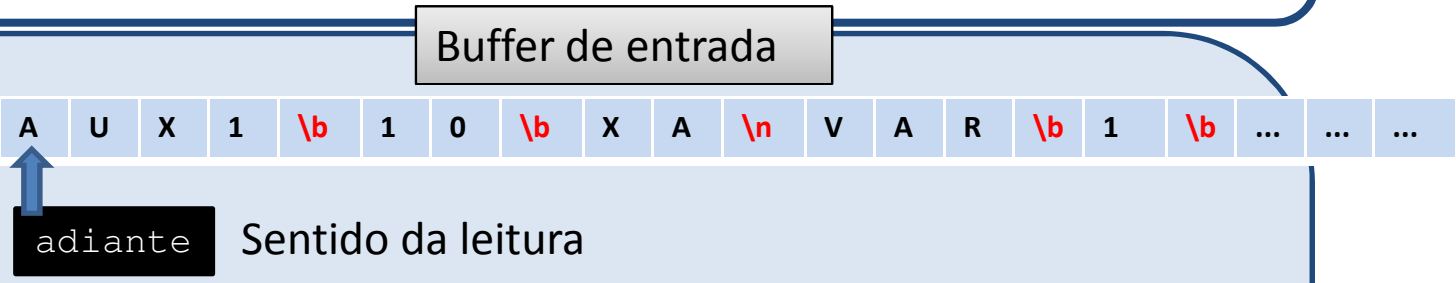
Estilo de implementação

- Cada estado é analisado individualmente em uma estrutura do tipo **switch...case**



Estilo de implementação

```
//exemplo de arquivo
AUX1  10 XA
VAR  1 ...
```

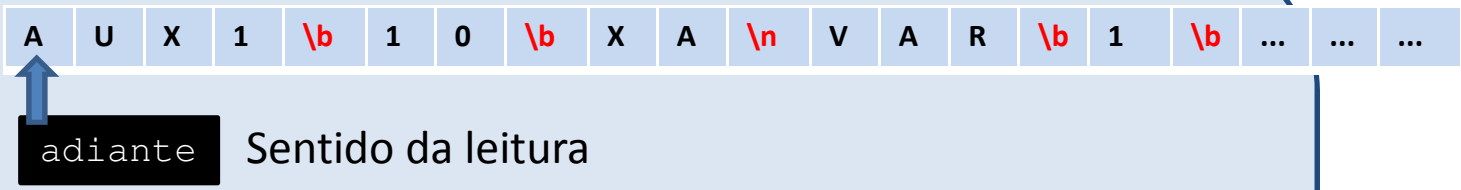


1) A entrada de dados é obtida a partir de um arquivo texto.
2) Cria-se uma estrutura (buffer) para armazená-los em memória (principal)
3) Cada caractere é avaliado individualmente → são separados por “\b” espaço, “\t” tabulação ou “\n” quebra de linha

Estilo de implementação

AUX1 10 XA
VAR 1 ...

Buffer de entrada



➔ O analisador léxico deverá ler e classificar as lexemas considerando o AF e retornar:

1. AUX1 → ID
2. 10 → ID inválido
3. XA → ID
4. VAR → ID
5. 1 → ID inválido

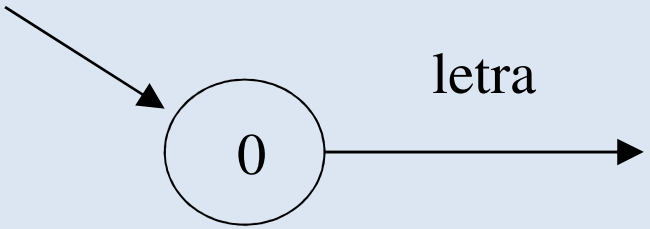
Estilo de implementação

Buffer de entrada

A	U	X	1	\b	1	0	\b	X	A	\n	V	A	R	\b	1	\b	...	...	...

adiante

```
int lexico()
{
   while (1)
   {
      switch (estado)
      {
         case 0: c= proximo_caracter();
                 if (isalpha(c))
                 {
                    estado= 1;
                    adiante++;
                 }
                 else
                 {
                     falhar(); //identificador inválido
                 }
                 break;
         …
      }
}
```



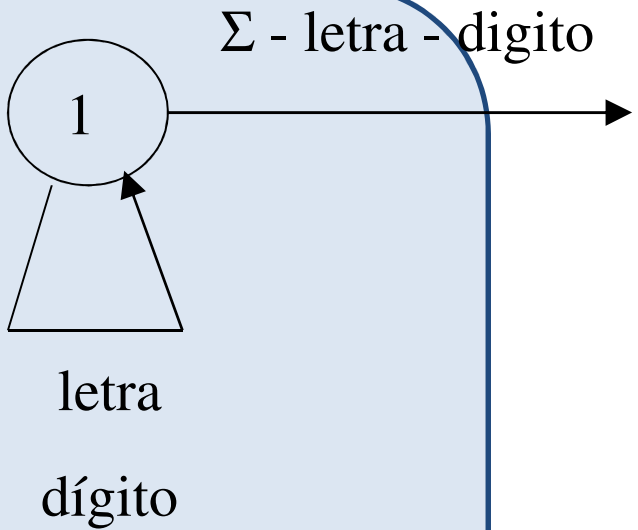
Estilo de implementação

Buffer de entrada

A	U	X	1	\b	1	0	\b	X	A	\n	V	A	R	\b	1	\b	…	…	…

adiante

```
…
case 1: c= proximo_caracter();
            if (isalpha(c) || isdigit(c))
            {
               estado= 1;
               adiante++;
            }
            else
            {

            if ((c == '\n') || (c == '\t') || (c == '\b'))
                  estado= 2;
                  else falhar();
            }
            break;
…
   }
}
```



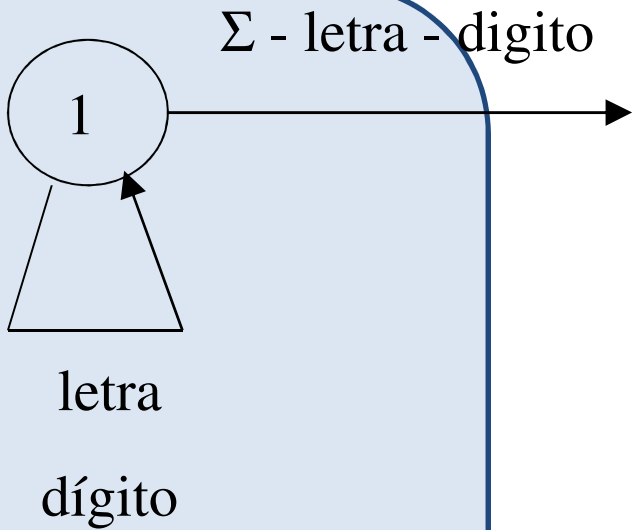
Estilo de implementação

Buffer de entrada

A	U	X	1	\b	1	0	\b	X	A	\n	V	A	R	\b	1	\b	…	…	…

adiante

```
        …
        case 1: c= proximo_caracter();
                    if (isalpha(c) || isdigit(c))
                    {
                       estado= 1;
                       adiante++;
                    }
                    else
                    {

                    if ((c == '\n') || (c == '\t') || (c == '\b'))
                           estado= 2;
                           else falhar();
                    }
                    break;
        …
    }
}
```



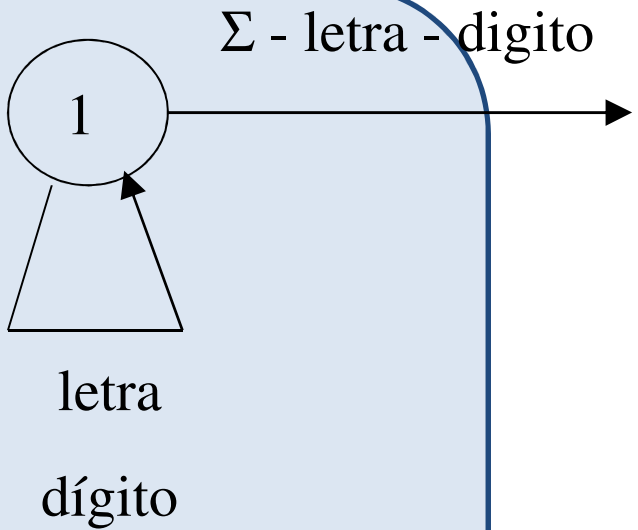
Estilo de implementação

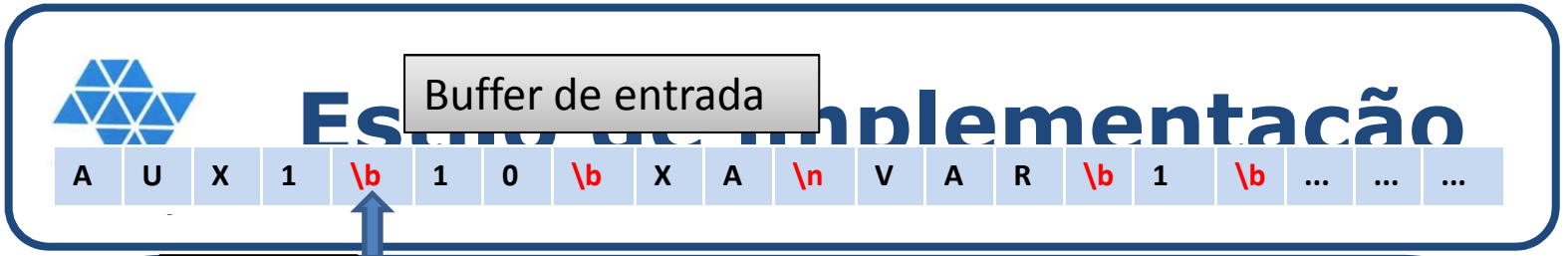
Buffer de entrada

A	U	X	1	\b	1	0	\b	X	A	\n	V	A	R	\b	1	\b	…	…	…

adiante

```
        …
        case 1: c= proximo_caracter();
                    if (isalpha(c) || isdigit(c))
                    {
                       estado= 1;
                       adiante++;
                    }
                    else
                    {

                    if ((c == '\n') || (c == '\t') || (c == '\b'))
                           estado= 2;
                           else falhar();
                    }
                    break;
        …
    }
}
```





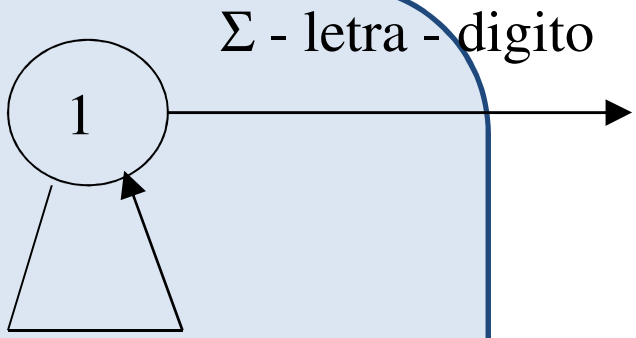
Estilo de implementação

Buffer de entrada

A	U	X	1	\b	1	0	\b	X	A	\n	V	A	R	\b	1	\b	…	…	…

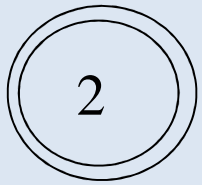
adiante

```
    …
    case 1: c= proximo_caracter();
                if (isalpha(c) || isdigit(c))
                {
                   estado= 1;
                   adiante++;
                }
                else
                {

                if ((c == '\n') || (c == '\t') || (c == '\b'))
                       estado= 2;
                       else falhar();
                }
                break;
    …
  }
}
```



Estilo de implementação

```
    …
            case 2: estado= 0;
                        partida= 0;
                        return ID;
                        break;

        }
} }
```



Reconhece ID

Gramática e Reconhecedores

- Um reconhecedor pode ser construído algoritmicamente para uma gramática livre de contexto;
- Analisadores sintáticos são chamados de ***PARSERS*** e constroem árvores para analisar determinados programas.
  - Tipos de analisadores: Análise sintática recursiva descendente e ascendente

Análise sintática

- Análise Sintática
  - Processo de rastrear ou de construir uma árvore sintática para determinada se string de entrada pode ser reconhecida pela gramática

- Analisa o código fonte

Análise sintática descendente recursiva

- Método que visa a encontrar uma derivação mais à esquerda para uma cadeia de entrada procurando construir uma árvore gramatical a partir da raiz criando nós em pré-ordem

- Funcionamento:
  - a construção da árvore gramatical começa de cima para baixo usando o símbolo de partida;
  - a medida em que são encontrados os símbolos *não-terminais* são expandidos;
  - da mesma forma são consumidos um a um os símbolos *terminais* encontrados.

Análise sintática descendente recursiva

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \in$

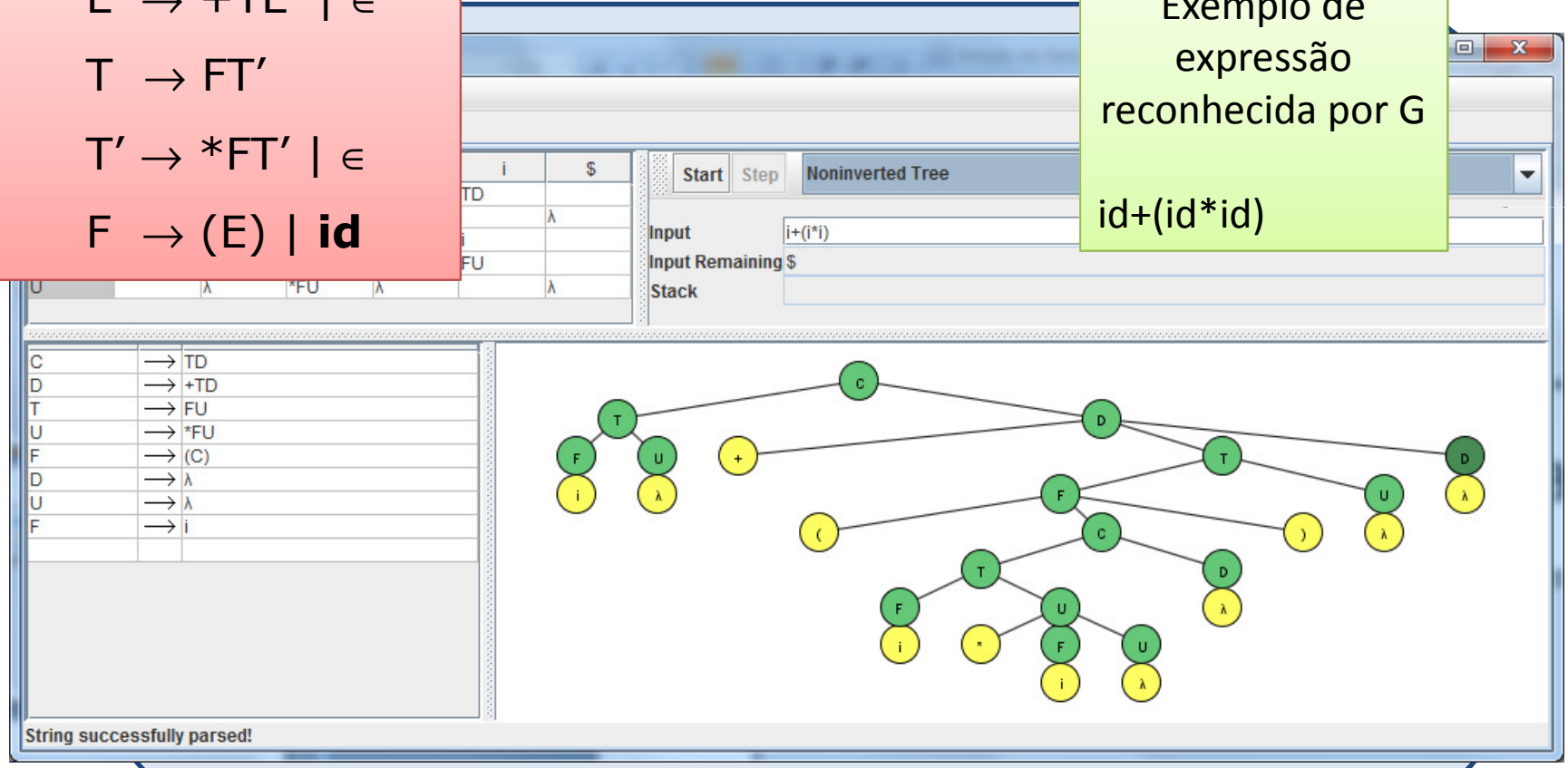
$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \in$

$F \rightarrow (E) \mid \mathbf{id}$

Exemplo de expressão reconhecida por G

id+(id*id)



Exercícios

- Referentes ao capítulo 3
  - 1, 6, 10, 11, 12 (pág 184)
  - 1, 2, 3, 6, 7, 8, 10, 11, 12 (págs. 185 e 186)
- Referentes ao capítulo 4
  - 1, 2, 3, 4, 5, 6 (pág 223)