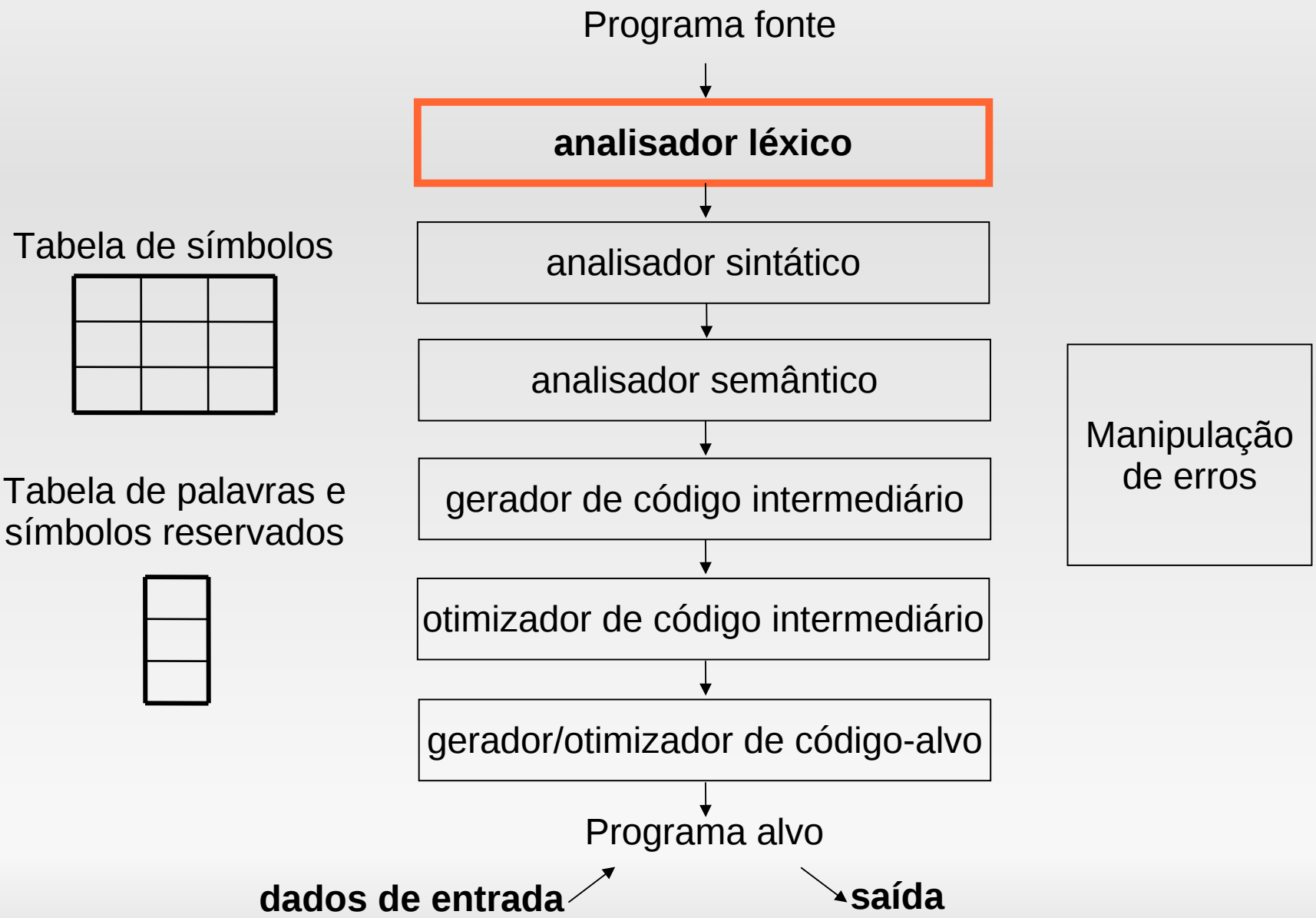
Construção de Compiladores

Análise Léxica

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

Processo de Tradução

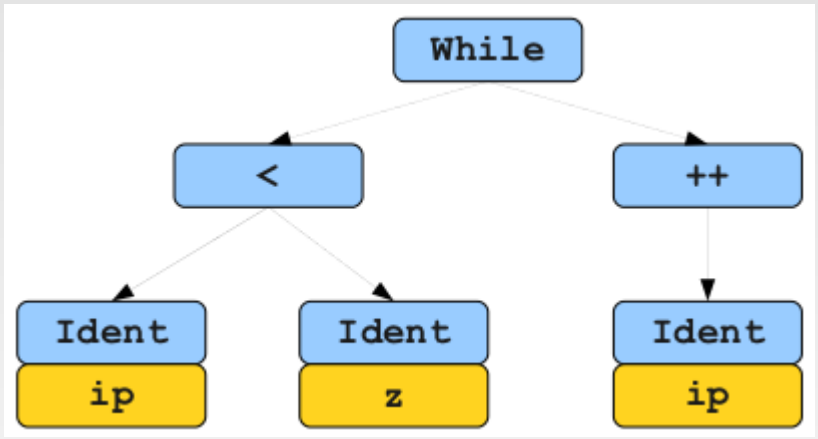


Análise Léxica

- O que é?
  - Etapa que lê o programa fonte como uma sequência de caracteres e os separa em *tokens*:
    - palavras reservadas: if, while etc.
    - identificadores
    - símbolos reservados: +, *, >=, <> etc.

Análise Léxica

- O que é?
  - Etapa que lê o programa fonte como uma sequência de caracteres e os separa em *tokens*

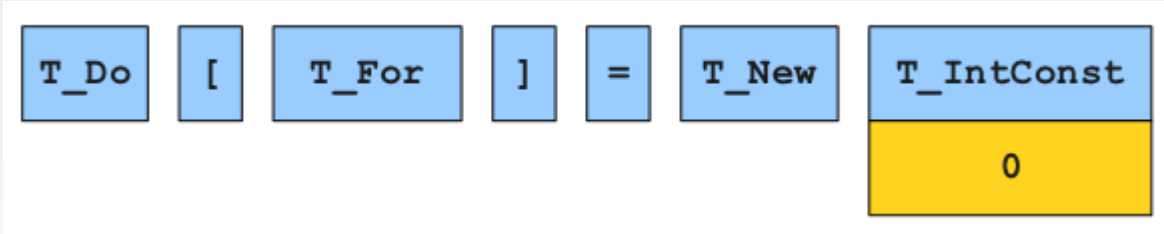


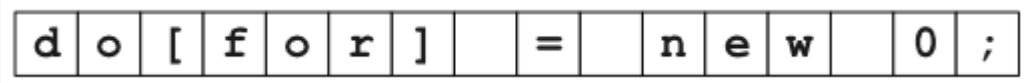

```
while (ip < z)
    ++ip;
```

http://www.stanford.edu/class/cs143/

Análise Léxica

- O que é?
  - Etapa que lê o programa fonte como uma sequência de caracteres e os separa em *tokens*






```
do[for] = new 0;
```

http://www.stanford.edu/class/cs143/

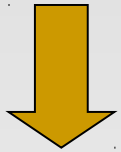
Análise Léxica

- O que é?
  - O *token*
    - Representa um determinado padrão de caracteres reconhecido pela varredura a partir do início à medida que esses caracteres são fornecidos
    - Possui várias informações associadas
      - categoria (identificador, símbolo, número, palavra reservada, etc.)
      - cadeia (valor ou lexema)
      - posição (linha e coluna em que ocorre no programa fonte)
    - Pode ter apenas um lexema (*token* simples) ou vários lexemas (*token* com argumentos)
      - Um lexema: palavras reservadas (if, then, else, while, etc.)
      - Vários lexemas: identificadores (var1, x, y, proc1, etc.)
        - todos *tokens* ” id”

Análise Léxica

- Exemplo

x:=x+y*2



<x,id> <:=,:=> <x,id> <+,op> <y,id> <*,op> <2,num>

cadeia categoria

- Como particionar o programa em *tokens*?

```
DO 5 I = 1,25
DO 5 I = 1.25
```

X

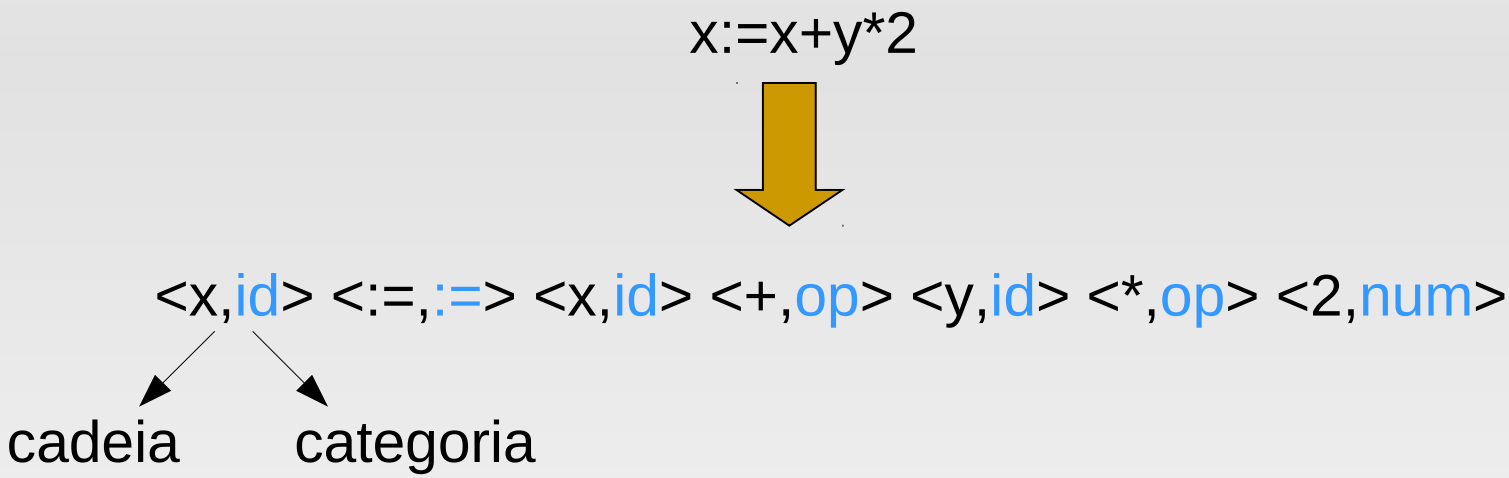
```
DO 5 I = 1,25
 DO5I  = 1.25
```

em FORTRAN espaço em branco é irrelevante

http://www.stanford.edu/class/cs143/

Análise Léxica

- Exemplo



- Como classificar cada *token* corretamente?

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

Palavras reservadas podem ser usadas como identificador em algumas linguagens

http://www.stanford.edu/class/cs143/

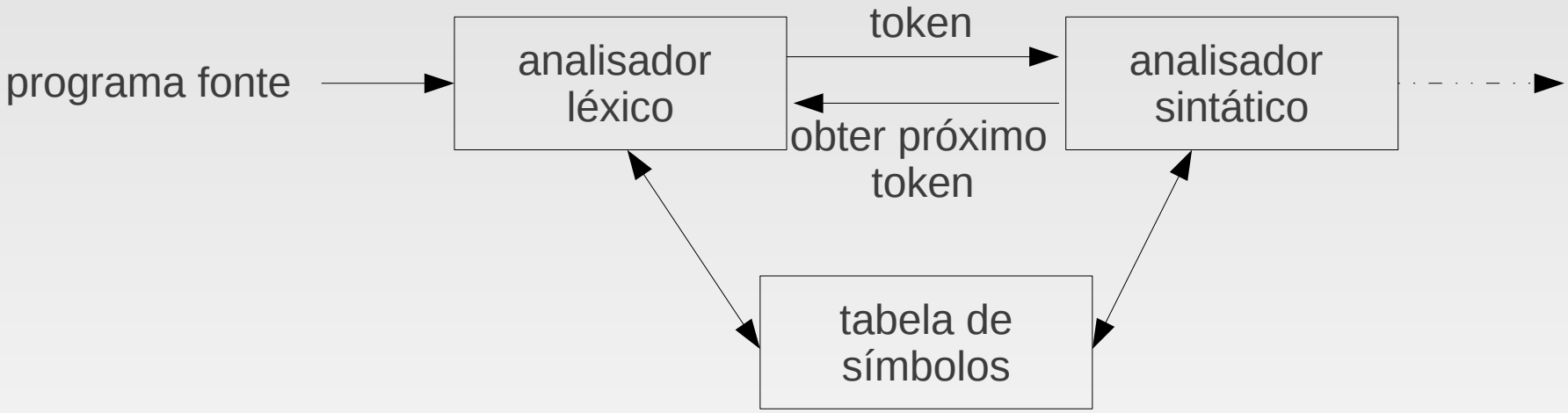
Análise Léxica

- Como é projetada?
  1. Escolha dos *tokens* (suas categorias)
     - Tipicamente: palavras e símbolos reservados, identificadores, números, etc.
     - Descartando comentários e espaços em branco
  2. Definição do conjunto de lexemas que cada categoria/*token* pode assumir
     - Combinação válida de caracteres na linguagem para formar os lexemas
     - ➔ **Expressões regulares**
  3. Projeto e implementação do algoritmo capaz de identificar e classificar corretamente os *tokens* desejados
     - ➔ **Autômatos**

http://www.stanford.edu/class/cs143/

Análise Léxica

- Como é executada?
  - Sob o comando do analisador sintático que requisita um *token* sempre que necessário



  - Motivos para dividir conceitualmente léxica de sintática
    - Simplificação, modularização e portabilidade
    - Diferentes notações garantem maior eficiência

Análise Léxica

- Como é executada?
  - Sob o comando do analisador sintático que requisita um *token* sempre que necessário
  - Exemplo em C:

```
typedef enum
{ IF, THEN, ELSE, PLUS, MINUS, NUM, ID, ...}
TokenType;

TokenType getToken(void);
```

  - A função `getToken` retorna o próximo *token* e computa todos os atributos do mesmo
  - ➔ **Expressões regulares** e **autômatos finitos**

Análise Léxica

- Expressões regulares – recordando
  - Representam padrões de cadeias de caracteres
  - Uma expressão regular é uma das seguintes:
    - Uma expressão regular básica composta por um único caractere *a* pertencente ao alfabeto ∑ de caracteres legais, a cadeia vazia (Ɛ) ou o conjunto vazio (Φ). Em que, L(*a*)={*a*}, L(Ɛ)={Ɛ} e L(Φ)={}
      - Exemplos: a, b, Ɛ

Análise Léxica

- Expressões regulares – recordando
  - Representam padrões de cadeias de caracteres
  - Uma expressão regular é uma das seguintes:
    - Uma expressão regular básica composta por um único caractere *a* pertencente ao alfabeto ∑ de caracteres legais, a cadeia vazia (Ɛ) ou o conjunto vazio (Φ). Em que, L(*a*)={*a*}, L(Ɛ)={Ɛ} e L(Φ)={}
    - Uma expressão da forma *r*|*s* onde *r* e *s* são expressões regulares. Em que, L(*r*|*s*) = L(*r*) ∪ L(*s*)
      - Exemplos: a|b, 0|Ɛ, a|b|c|d, 0|...|9

Análise Léxica

- Expressões regulares – recordando
  - Representam padrões de cadeias de caracteres
  - Uma expressão regular é uma das seguintes:
    - Uma expressão regular básica composta por um único caractere *a* pertencente ao alfabeto ∑ de caracteres legais, a cadeia vazia (Ɛ) ou o conjunto vazio (Φ). Em que, L(*a*)={*a*}, L(Ɛ)={Ɛ} e L(Φ)={}
    - Uma expressão da forma *r*|*s* onde *r* e *s* são expressões regulares. Em que, L(*r*|*s*) = L(*r*) ∪ L(*s*)
    - Uma expressão da forma *rs* onde *r* e *s* são expressões regulares. Em que, L(*rs*) = L(*r*)L(*s*)
      - Exemplos: ab, (a|b)c, (a|b)c(b|f)

Análise Léxica

- Expressões regulares – recordando
  - Representam padrões de cadeias de caracteres
  - Uma expressão regular é uma das seguintes:
    - Uma expressão regular básica composta por um único caractere *a* pertencente ao alfabeto ∑ de caracteres legais, a cadeia vazia (Ɛ) ou o conjunto vazio (Φ). Em que, L(*a*)={*a*}, L(Ɛ)={Ɛ} e L(Φ)={}
    - Uma expressão da forma *r*|*s* onde *r* e *s* são expressões regulares. Em que, L(*r*|*s*) = L(*r*) ∪ L(*s*)
    - Uma expressão da forma *rs* onde *r* e *s* são expressões regulares. Em que, L(*rs*) = L(*r*)L(*s*)
    - Uma expressão da forma *r** onde *r* é uma expressão regular. Em que, L(*r**) = L(*r*)*
      - Exemplo: c*, (ab)*, (a|bb)*

Análise Léxica

- Expressões regulares – recordando
  - Representam padrões de cadeias de caracteres
  - Uma expressão regular é uma das seguintes:
    - Uma expressão regular básica composta por um único caractere *a* pertencente ao alfabeto ∑ de caracteres legais, a cadeia vazia (Ɛ) ou o conjunto vazio (Φ). Em que, L(*a*)={*a*}, L(Ɛ)={Ɛ} e L(Φ)={}
    - Uma expressão da forma *r*|*s* onde *r* e *s* são expressões regulares. Em que, L(*r*|*s*) = L(*r*) ∪ L(*s*)
    - Uma expressão da forma *rs* onde *r* e *s* são expressões regulares. Em que, L(*rs*) = L(*r*)L(*s*)
    - Uma expressão da forma *r** onde *r* é uma expressão regular. Em que, L(*r**) = L(*r*)*
    - Os parênteses não modificam a linguagem e são usados apenas para ajustar precedência de operadores: * > concatenação > |

Análise Léxica

- Expressões regulares – recordando
  - Exercícios
    - Dê a expressão regular que representa o conjunto de todas as cadeias sobre o alfabeto $\sum = \{a, b, c\}$ com
      1. Exatamente um *b* (a|c)*b(a|c)*
      2. No máximo um *b* (a|c)*(b|Ɛ)(a|c)*
      3. *b*s ocorrendo em pares (2 *b*s em sequência) (a|c|bb)*

Análise Léxica

- Expressões regulares – recordando
  - Notação estendida
    - Uma ou mais repetições: +
      - Exemplo: $(0|1)^+$ é equivalente a (0|1)(0|1)*
    - Qualquer caractere: .
      - Exemplo: .**b*.* representa todas as cadeias que contêm pelo menos um *b*
    - Intervalo de caracteres: [ ]
      - Exemplos: [abc] equivale a [a|b|c] e [0-9] a 0|...|9
    - Exceção de caractere: ~ ou ^
      - Exemplo: ~(a|b|c) ou [^abc] equivale a qualquer caractere do alfabeto exceto *a*, *b* ou *c*
    - Subexpressão opcional: r?
      - Exemplo: (+|-)? $[0-9]^+$ representa números com sinal

Análise Léxica

- Expressões regulares
  - Podem ser usadas para representar as categorias normalmente presentes em uma linguagem
    - Identificadores
    - Palavras reservadas
    - Símbolos especiais
    - Comentários
    - Números inteiros, reais, com ou sem sinal
  - Porém, não podem resolver ambiguidades
    - Nesse caso, algumas regras são normalmente seguidas
      - Palavras reservadas não podem ser identificadores
      - Símbolos maiores são preferidos (identifica <> e não < e >)

Análise Léxica

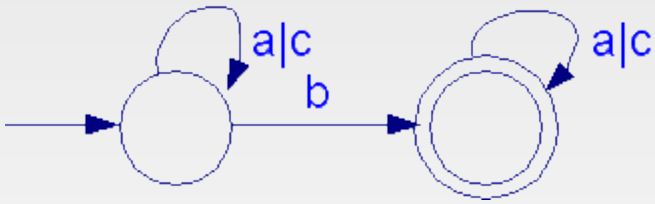
- Autômatos finitos – recordando
  - Uma forma matemática de descrever tipos particulares de algoritmos
  - Um autômato finito determinístico M é composto por
    - Um alfabeto $\sum$
    - Um conjunto de estados S
    - Uma função de transição T: S x $\sum$ → S
    - Um estado inicial $s_0 \in S$
    - Um conjunto de estados de aceitação $A \subset S$
    - ➔ A linguagem aceita por M, L(M), é o conjunto das cadeias de caracteres $c_1c_2...c_n$ onde cada $c_i \in \sum$ e existem estados $s_1=T(s_0,c_1)$, $s_2=T(s_1,c_2)$, ..., $s_n=T(s_{n-1},c_n)$ em que $s_n \in A$

Análise Léxica

- Autômatos finitos – recordando
  - Exercícios
    - Dê o autômato finito determinístico que reconhece o conjunto de todas as cadeias sobre o alfabeto $\sum = \{a,b,c\}$ com
      1. Exatamente um *b*
      2. No máximo um *b*
      3. *b*s ocorrendo em pares

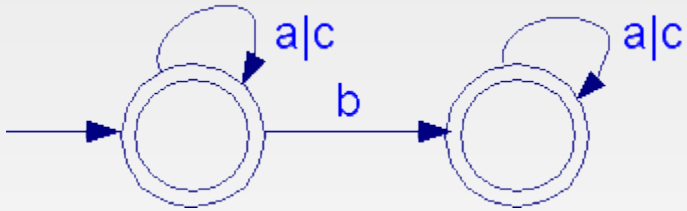
Análise Léxica

- Autômatos finitos – recordando
  - Exercícios
    - Dê o autômato finito determinístico que reconhece o conjunto de todas as cadeias sobre o alfabeto $\sum = \{a,b,c\}$ com
      1. Exatamente um *b* (a|c)*b(a|c)*



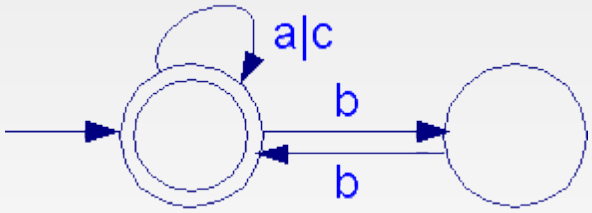
Análise Léxica

- Autômatos finitos – recordando
  - Exercícios
    - Dê o autômato finito determinístico que reconhece o conjunto de todas as cadeias sobre o alfabeto ∑ = {*a*,*b*,*c*} com
      1. Exatamente um *b*
      2. No máximo um *b* (a|c)*(b|Ɛ)(a|c)*



Análise Léxica

- Autômatos finitos – recordando
  - Exercícios
    - Dê o autômato finito determinístico que reconhece o conjunto de todas as cadeias sobre o alfabeto $\sum = \{a,b,c\}$ com
      1. Exatamente um *b*
      2. No máximo um *b*
      3. *b*s ocorrendo em pares (a|c|bb)*



Análise Léxica

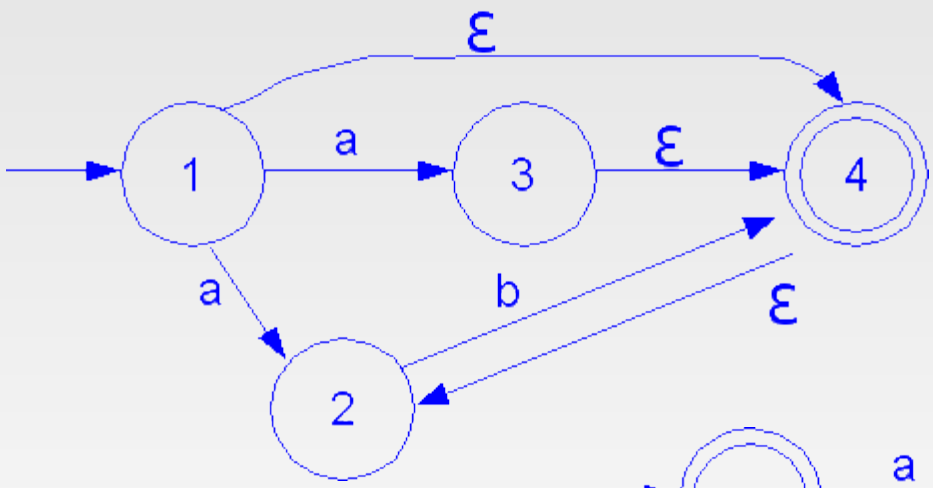
- Autômatos finitos
  - Podem ser utilizados para descrever o processo de reconhecimento de padrões em cadeias de entrada
    - Identificadores e palavras reservadas
    - Símbolos especiais
    - Comentários
    - Números inteiros, reais, com ou sem sinal
    - → Cada um reconhecido por seu autômato próprio
  - Para agrupar todos esses autômatos utiliza-se a noção de autômato finito não determinístico

Análise Léxica

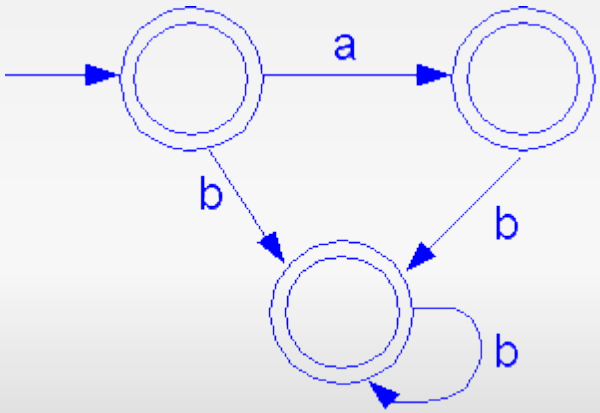
- Autômatos finitos – recordando
  - Um autômato finito não determinístico M é composto por
    - Um alfabeto $\sum$
    - Um conjunto de estados S
    - Uma função de transição T: S x $\sum \cup \{\varepsilon\} \rightarrow \wp(S)$
    - Um estado inicial $s_0 \in S$
    - Um conjunto de estados de aceitação $A \subset S$
    - ➔ A linguagem aceita por M, L(M), é o conjunto das cadeias de caracteres $c_1c_2...c_n$ onde cada $c_i \in \sum \cup \{\varepsilon\}$ e existem estados s_1 em $T(s_0,c_1)$, s_2 em $T(s_1,c_2)$, ..., s_n em $T(s_{n-1},c_n)$ em que $s_n \in A$
  - Diferente de um AFD, um AFND pode possuir mais de um estado resultante em cada transição e transições com a cadeia vazia (ε)

Análise Léxica

- Autômatos finitos – recordando
  - Exemplo
    - O autômato finito não determinístico a seguir aceita a linguagem gerada pela expressão regular (a|Ɛ)b*



    - E é equivalente ao AFD



Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

1. Divida o código em tokens
2. Para cada token identificado indique: cadeia e categoria
3. Agrupe os tokens de mesma categoria
4. Defina uma regra (ou um padrão) para os tokens que fazem parte de cada categoria
5. Escreva uma expressão regular para cada categoria
6. Faça o autômato correspondente à expressão regular definida para cada categoria

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

1. Divida o código em tokens

1	int	7	;	13	)	19	(	25	}														
2	x	8	while	14	{	20	”%d\n”																
3	;	9	(	15	x	21	,																
4	x	10	x	16	--	22	x																
5	=	11	>	17	;	23	)																
6	5	12	1.0	18	printf	24	;																

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

2. Para cada token identificado indique: cadeia e categoria

	cadeia	categoria
1	int	palavra reserv.
2	x	identificador
3	;	símbolo reserv.
4	x	identificador
5	=	símbolo reserv.
6	5	número inteiro

7	;	símbolo reserv.
8	while	palavra reserv.
9	(	símbolo reserv.
10	x	identificador
11	>	símbolo reserv.
12	1.0	número real
13	)	símbolo reserv.

14	{	símbolo reserv.
15	x	identificador
16	--	símbolo reserv.
17	;	símbolo reserv.
18	printf	identificador
19	(	símbolo reserv.
20	”%d\n”	cadeia

21	,	símbolo reserv.
22	x	identificador
23	)	símbolo reserv.
24	;	símbolo reserv.
25	}	símbolo reserv.

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

3. Agrupe os tokens de mesma categoria

Palavas reservadas	int while
Símbolos reservados	; = (>) { -- , }
Identificadores	x printf
Números inteiros	5
Números reais	1.0
Cadeias	”%d\n”

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

4. Defina uma regra (ou um padrão) para os tokens que fazem parte de cada categoria

categoria	tokens	regra
Palavas reservadas	int while	A própria cadeia
Símbolos reservados	; = (>) { -- , }	A própria cadeia
Identificadores	x printf	Sequência de caracteres começando com letra
Números inteiros	5	Sequência de dígitos
Números reais	1.0	Sequência de dígitos com ponto
Cadeias	”%d\n”	Sequência de caracteres delimitados por aspas

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

5. Escreva uma expressão regular para cada categoria

categoria	tokens	expressão regular
Palavas reservadas	int while	[int|while]
Símbolos reservados	; = (>) { -- , }	[;|=|(|>|)|{|--|,|}]
Identificadores	x printf	[a-zA-Z]([a-zA-Z]|[0-9]|_)*
Números inteiros	5	[0-9]+
Números reais	1.0	[0-9]+.[0-9]+
Cadeias	”%d\n”	”[^”]*”

Análise Léxica

- Exercícios
  - Considere o programa fonte a seguir em C

```
int x;
x = 5;
while (x > 1.0) {
    x--;
    printf("%d\n",x);
}
```

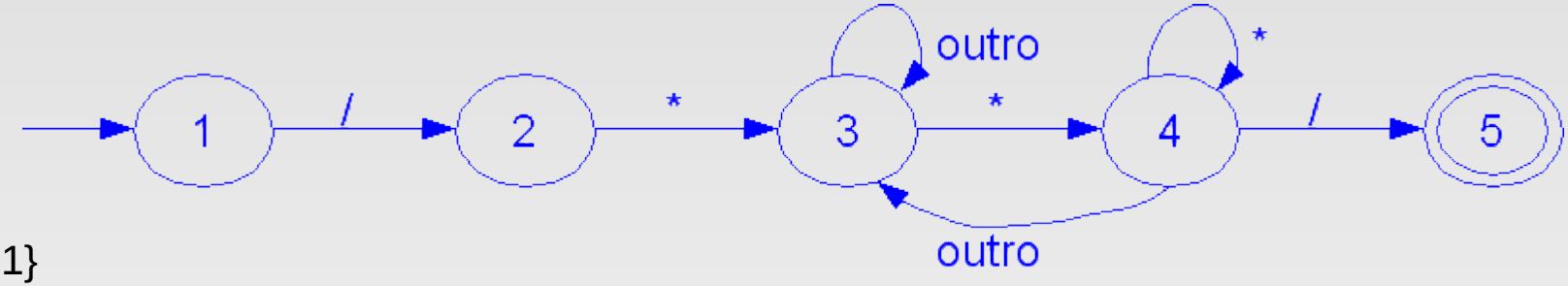
6. Faça o autômato correspondente à expressão regular definida para cada categoria

Análise Léxica

- Autômatos finitos
  - Podem ser utilizados para descrever o processo de reconhecimento de padrões em cadeias de entrada
    - Identificadores e palavras reservadas
    - Símbolos especiais
    - Comentários
    - Números inteiros, reais, com ou sem sinal
    - → Cada um reconhecido por seu autômato próprio
  - Opções de implementação
    - Usando ifs
    - Usando cases
    - Usando tabela de transição

Análise Léxica

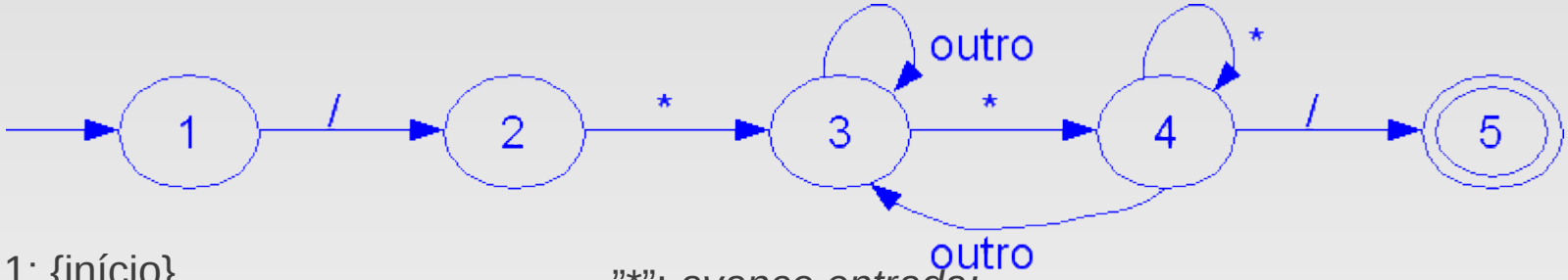
- Autômatos finitos – traduzindo AFs em código (ifs)
  - Exemplo: AFD que reconhece comentários em C (/*...*/)




```
{estado 1}
if próximo caractere for "/" then
   avance entrada; {estado 2}
   if próximo caractere for "*" then
      avance entrada; {estado 3}
      fim := false;
      while not fim do
         while próximo caractere não for "*" do
            avance entrada;
         end while;
         avance entrada; {estado 4}
         while próximo caractere for "*" do
            avance entrada;
         end while;
         if próximo caractere for "/" then
            fim := true;
         end if;
         avance entrada;
      end while;
      aceitação; {estado 5}
   else {outro processamento}
   end if;
else {outro processamento}
end if;
```

Análise Léxica

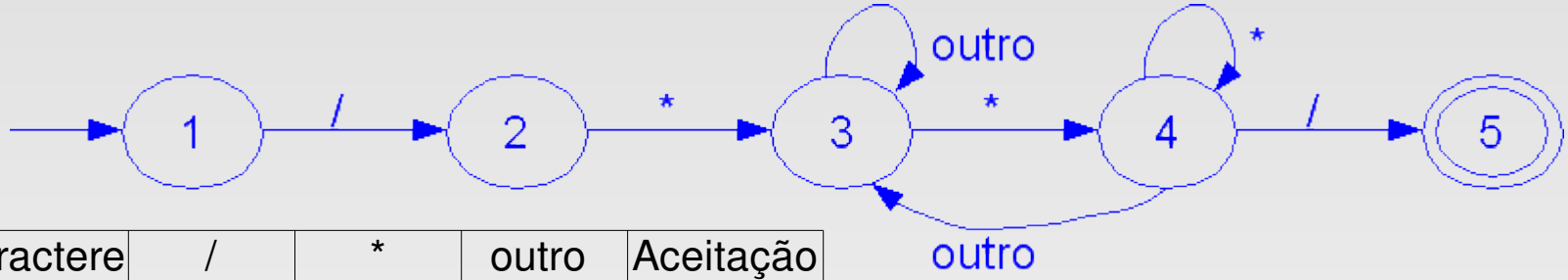
- Autômatos finitos – traduzindo AFs em código (cases)
  - Exemplo: AFD que reconhece comentários em C (/*...*/)




```
estado := 1; {início}
while estado = 1, 2, 3 ou 4 do
  case estado of
  1: case caractere de entrada of
     "/": avance entrada;
          estado := 2;
     else estado := ...; {erro ou outro}
     end case;
  2: case caractere de entrada of
     "*": avance entrada;
          estado := 3;
     else estado := ...; {erro ou outro}
     end case;
  3: case caractere de entrada of
     "*": avance entrada;
          estado := 4;
     else avance entrada; {e continue no estado 3}
     end case;
  4: case caractere de entrada of
     "/": avance entrada;
          estado := 5;
     "*": avance entrada; {e continue no estado 4}
     else avance entrada;
          estado := 3;
     end case;
  end case;
end while;
if estado = 5 then aceitação else erro;
```

Análise Léxica

- Autômatos finitos – traduzindo AFs em código (tabela)
  - Exemplo: AFD que reconhece comentários em C (/*...*/)



estado/caractere	/	*	outro	Aceitação
1	2			não
2		3		não
3	3	4	3	não
4	5	4	3	não
5				sim

```
estado := 1;
ch := próximo caractere de entrada;
while not Aceita[estado] and not erro(estado) do
   novoestado := T[estado,ch];
   if Avance[estado,ch] then ch := próximo caractere de entrada;
   estado := novoestado;
end while;
if Aceita[estado] then aceitação;
```

Métodos dirigidos por tabela

Vantagens:
- Código de tamanho reduzido
- Código mais fácil de manter

Desvantagens:
- Tabelas podem ser grandes
- Aumento espaço → compressão

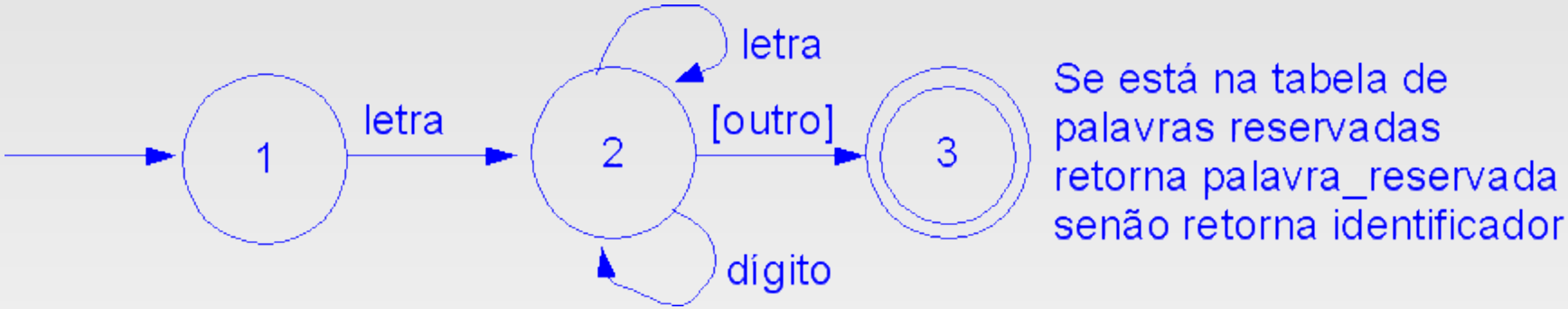
➔ Usados por programas como Lex

Análise Léxica

- Projeto de um Analisador Léxico
  - Identificação de tokens
    - → Como delimitar os tokens? Separados por espaço em branco?
  - Tipos de tokens a serem identificados
    1. Palavras reservadas: if, while, etc.
    2. Identificadores
    3. Números inteiros, reais, com ou sem sinal: 1, 0.3, -4
    4. Símbolos reservados: +, *, >=, <>, etc.
    - → Como diferenciar cada tipo?
  - Comentários
    - → O que fazer com os comentários?
  - Saída gerada
    - Categoria do token (p. ex. uma das 4 apresentadas acima)
    - Cadeia do token (p. ex. 0.3, if, >=)
    - → Ou seja, não adianta só reconhecer os caracteres, eles devem ser armazenados em cadeia que é retornada ao final

Análise Léxica

- Projeto de um Analisador Léxico
  - Exemplo de um autômato que reconhece identificadores e palavras reservadas

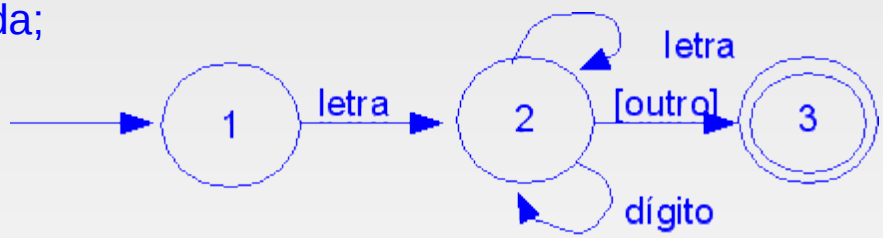


  - Como seria o algoritmo correspondente a esse autômato?

Análise Léxica

- Projeto de um Analisador Léxico
  - Exemplo de um algoritmo que reconhece identificadores e palavras reservadas

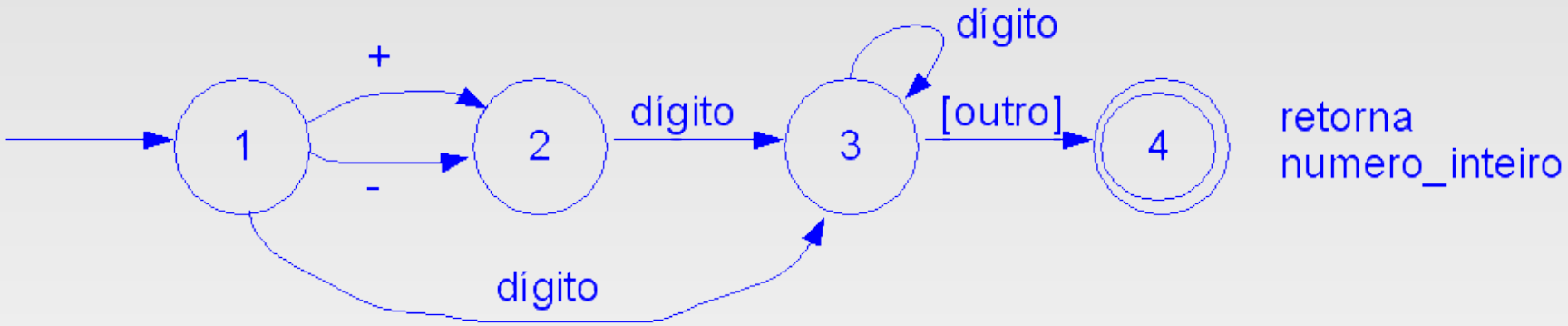
```
cadeia := ””; {cadeia que vai armazenar token lido}
estado := 1; {início}
while estado = 1 ou 2 do
   case estado of
   1: case caractere de entrada of
      letra: cadeia += caractere de entrada;
             avance entrada;
             estado := 2;
      else estado := erro;
      end case;
   2: case caractere de entrada of
      dígito: cadeia += caractere de entrada;
              avance entrada;
      letra: cadeia += caractere de entrada;
             avance entrada;
      else estado := 3;
      end case;
   end case;
end while;
```

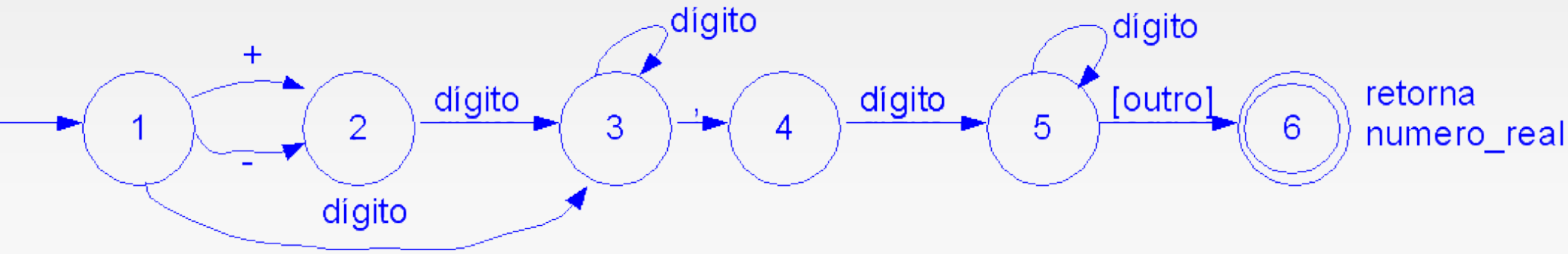



```
if estado = 3
then if cadeia esta na tabela de
          palavras reservadas
        then retorna (cadeia, cadeia)
        else retorna (cadeia, identificador)
else retorna erro
```

Análise Léxica

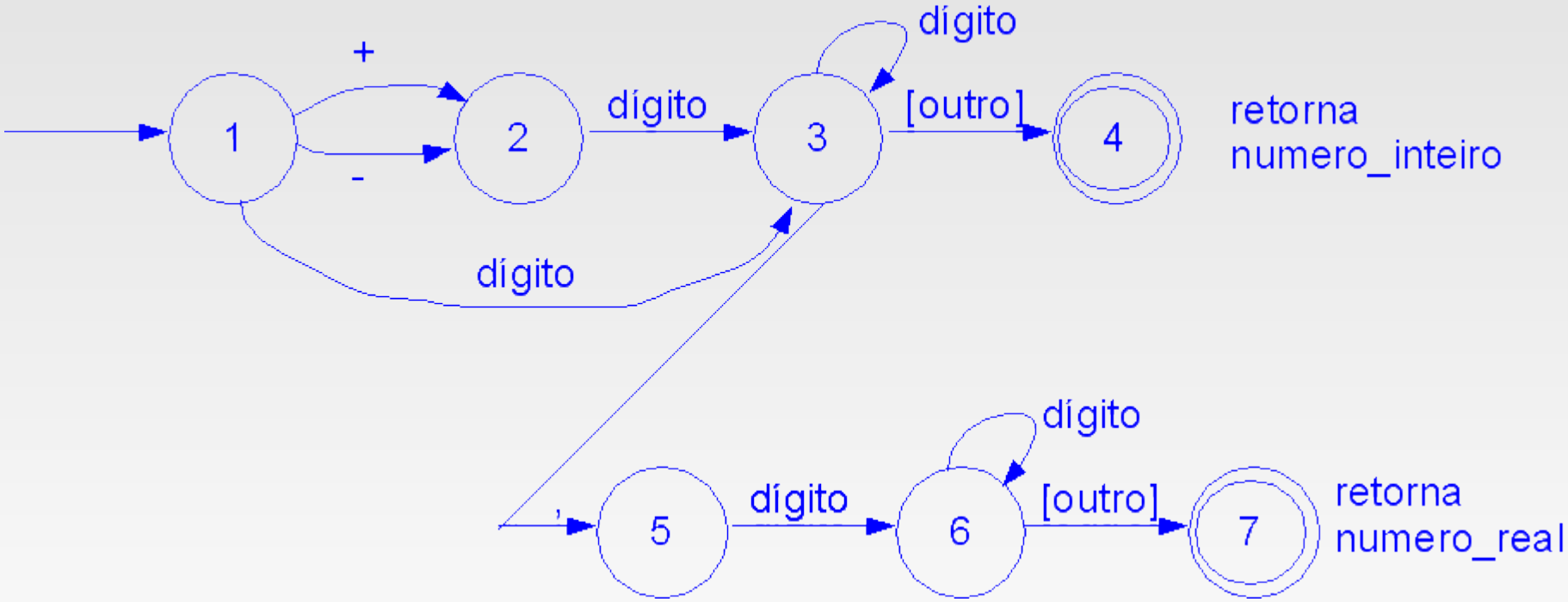
- Projeto de um Analisador Léxico
  - Exemplo de um autômato que reconhece números inteiros e outro números reais, com ou sem sinal





Análise Léxica

- Projeto de um Analisador Léxico
  - Exemplo de um autômato que reconhece números inteiros e reais, com ou sem sinal



  - Como seria o algoritmo correspondente a esse autômato?

Análise Léxica

- Projeto de um Analisador Léxico
  - Exemplo de um algoritmo que reconhece números inteiros e reais, com ou sem sinal

```
cadeia := "";
estado := 1; {início}
while estado = 1, 2, 3, 5 ou 6 do
  case estado of
  1: case caractere de entrada of
     "+": cadeia += caractere de entrada;
          avance entrada;
          estado := 2;
     "-": cadeia += caractere de entrada;
          avance entrada;
          estado := 2;
     dígito: cadeia += caractere de entrada;
             avance entrada;
             estado := 3;
     else estado := erro;
     end case;
  2: case caractere de entrada of
     dígito: cadeia += caractere de entrada;
             avance entrada;
             estado := 3;
     else estado := erro;
     end case;
  3: case caractere de entrada of
     dígito: cadeia += caractere de entrada;
             avance entrada;
     ",": cadeia += caractere de entrada;
          avance entrada;
          estado := 5;
     else estado := 4;
     end case;
  5: case caractere de entrada of
     dígito: cadeia += caractere de entrada;
             avance entrada;
             estado := 6;
     else estado := erro;
     end case;
  6: case caractere de entrada of
     dígito: cadeia += caractere de entrada;
             avance entrada;
     else estado := 7;
     end case;
  end case;
end while;
if estado = 4
then retorna (cadeia, numero_inteiro)
else if estado = 7
then retorna (cadeia, numero_real)
else retorna erro;
```