



Judson Santos Santiago

Reconhecimento de Tokens

Compiladores

Introdução

- **Expressões regulares** podem ser usadas para representar os padrões dos tokens válidos de uma linguagem

- União, concatenação e fechamento são as operações básicas


$a|a^*b = \{a, b, ab, aab, aaab, \dots\}$

- **Definições regulares** permitem dar nomes a expressões

regulares

letra_  A | B | ... | Z | a | b | ... | z | _

dígito  0 | 1 | ... | 9

id  *letra_*(*letra_*|*dígito*)*

Extensões de Expressões Regulares

- Desde que Kleene introduziu as expressões regulares em 1950, muitas extensões foram criadas
 - Algumas extensões importantes são aquelas incorporadas no Lex
 1. Uma ou mais instâncias: o operador + representa o fechamento positivo
 2. Zero ou uma instância: o operador ? representa zero ou uma ocorrência
 3. Classe de caracteres: [abc] representa a|b|c, [a-z] representa a|b|...|z

letra_ ≡ [A-Za-z_]

dígito ≡ [0-9]

id ≡ *letra_*(*letra_*|*dígito*)*

dígito ≡ [0-9]

dígitos ≡ *dígito*+

número ≡ *dígitos*(.*dígitos*)?(E[+-]?*dígitos*)?

Extensões de Expressões Regulares

Expressão	Casa com	Exemplo
\c	O caractere c literalmente	*
"s"	A cadeia s literalmente	"**"
.	Qualquer caractere menos quebra de linha	a.*b
^	O início de uma linha	^abc
\$	O fim de uma linha	abc\$
[s]	Qualquer um dos caracteres na cadeia s	[abc]
[^s]	Qualquer caractere não presente na cadeia s	[^abc]

Expressões Regulares do Lex

Extensões de Expressões Regulares

Expressão	Casa com	Exemplo
r^*	Zero ou mais cadeias casando com r	a^*
r^+	Uma ou mais cadeias casando com r	a^+
$r^?$	Zero ou um r	$a^?$
$r\{m,n\}$	Entre m e n ocorrências de r	$a\{1,5\}$
r_1r_2	Um r_1 seguido por um r_2	ab
$r_1 r_2$	Um r_1 ou um r_2	$a b$
(r)	O mesmo que r	$(a b)$
r_1/r_2	r_1 quando seguido por r_2	$abc/123$

Expressões Regulares do Lex

Extensões de Expressões Regulares

- As expressões regulares no Lex dão a todos estes símbolos um **significado especial**:

`\ " . ^ [] * + ? { } | /`

- Seu significado **precisa ser desativado** se eles forem necessários para representar a si mesmos em uma cadeia. Podemos fazer isso:
 - Colocando o caractere dentro de uma cadeia
`"**" casa com **`
 - Precedendo o caractere com uma barra invertida
`** casa com **`

Extensões de Expressões Regulares

- Uma **classe de caractere complementada** representa qualquer caractere, exceto os indicados na classe
 - Indicada pelo uso do **^ como primeiro caractere**
 - O caractere ^ não faz parte da classe: `[^A-Za-z]`
 - A menos que seja listado dentro da própria classe: `[^\^]`
 - A **quebra de linha** não pode estar em nenhuma classe de caractere
 - `[^\^]` denota qualquer caractere menos o circunflexo e a quebra de linha
 - O caractere ^ também indica o **início de uma linha**
 - A diferença é feita pelo **contexto**: `^[^aeiou]*$`

Exercícios

1. Escreva **classes de caracteres** para os seguintes conjuntos:
 - a) As vogais minúsculas
 - b) As dez primeiras letras (até j) em maiúsculo ou minúsculo
 - c) Os "dígitos" em um número hexadecimal
(letras maiúsculas para os "dígitos" acima de 9)
 - d) Os caracteres que podem aparecer no fim de uma sentença legítima em português (por exemplo, ponto de exclamação)
2. Escreva uma **definição regular** para representar números no formato hexadecimal, iniciando com 0x (**Ex.:** 0x27FD01).

Reconhecimento de Tokens

- As *extensões* permitem que as *expressões regulares* sejam definidas de forma mais simples

letra_  [A-Za-z_]

dígito  [0-9]

id  *letra_*(*letra_*|*dígito*)*

- Agora precisamos entender *como criar um analisador léxico* que:
 - Reconheça os tokens de uma linguagem
 - Use padrões descritos por expressões regulares

Reconhecimento de Tokens










- Uma gramática com instruções de desvio:

```
stmt  ::= if expr then stmt
       | if expr then stmt else stmt
       | ε
expr   ::= term relop term
       | term
term   ::= id
       | num
```

Os **terminais da gramática** são os tokens que precisam ser **reconhecidos** pelo analisador léxico

Reconhecimento de Tokens

- Os padrões para os tokens são descritos por definições regulares

if	 <code>if</code>
then	 <code>then</code>
else	 <code>else</code>
relop	 <code>< > <= >= = <></code>
<i>letra</i>	 <code>[A-Za-z]</code>
id	 <code><i>letra</i>(<i>letra</i> <i>dígito</i>)*</code>
<i>dígito</i>	 <code>[0-9]</code>
<i>dígitos</i>	 <code><i>dígito</i>+</code>
num	 <code><i>dígitos</i>(.<i>dígitos</i>)?(E[+-]?<i>dígitos</i>)?</code>

Para simplificar, vamos assumir que as *palavras-chave* (**if**, **then**, **else**) são reservadas

Reconhecimento de Tokens

- O analisador léxico removerá **espaços em branco**
 - Essa tarefa pode ser realizada pelo reconhecimento do **token ws** (white space)

ws  **(blank|tab|newline)+**

- blank, tab e newline são **símbolos abstratos**
 - Representam os caracteres de mesmo nome da tabela ASCII
- O **token ws é diferente** porque ele não é retornado ao analisador sintático
 - Simplesmente força o analisador léxico a continuar no caractere seguinte

Reconhecimento de Tokens

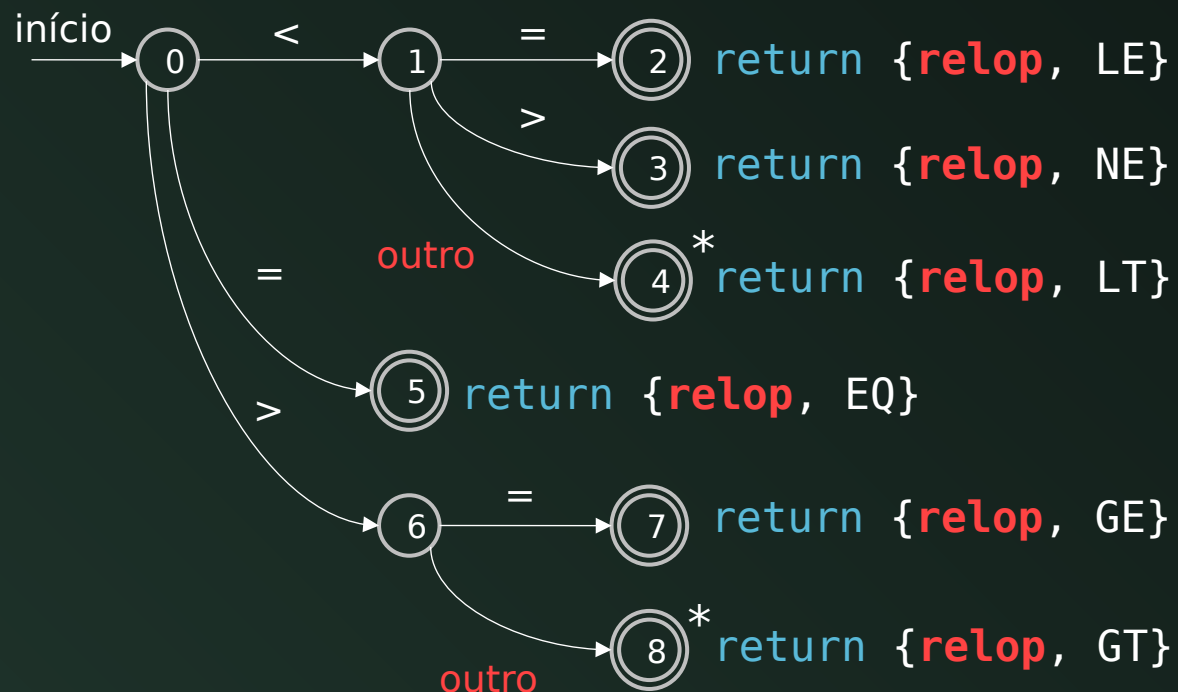
Lexemas	Token	Atributo
Espaços em branco	-	-
if	if	-
then	then	-
else	else	-
Identificadores	id	Apontador para tabela de símbolos
Números	num	Apontador para tabela de símbolos
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE

Os tokens retornados pelo analisador léxico

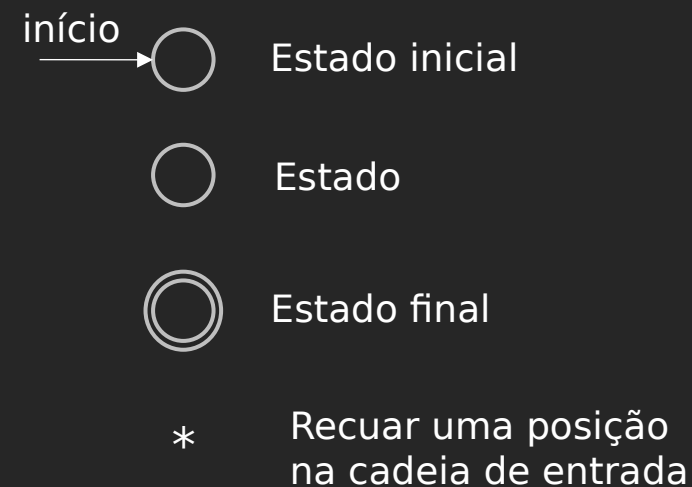
Diagramas de Transição

- O reconhecimento dos padrões descritos por expressões regulares pode ser implementado através de diagramas de transição
 - Vamos estudar a conversão manual
 - Existe uma forma automática de construir esses diagramas
- Os diagramas são caracterizados por:
 - Uma coleção de estados: cada estado descreve uma condição da entrada
 - Interligados por arestas: rotuladas por um símbolo ou conjunto de símbolos
 - Determinísticos: não existe duas arestas saindo do mesmo estado com o mesmo símbolo

Diagramas de Transição

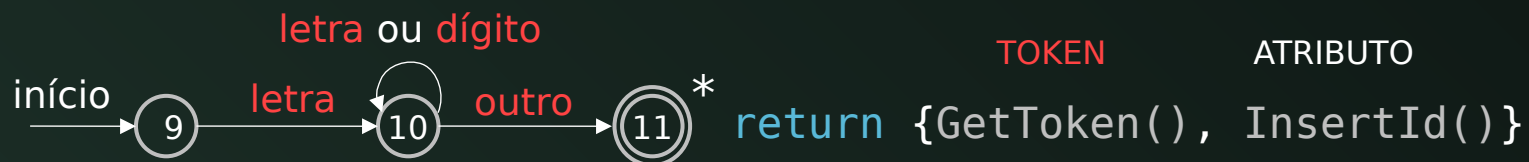


Legenda

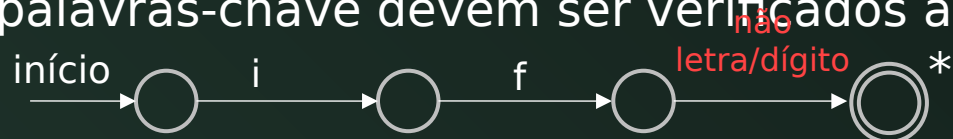


Diagramas de Transição

- Reconhecimento de palavras-chave e identificadores
 - Compartilham o mesmo padrão, um problema que pode ser resolvido:
 - Inserindo as palavras-chave na tabela de símbolos: qualquer nome que não estiver na tabela é um identificador e não uma palavra-chave



- Criando diagramas de transição para cada palavra-chave: os nomes das palavras-chave devem ser verificados antes dos identificadores

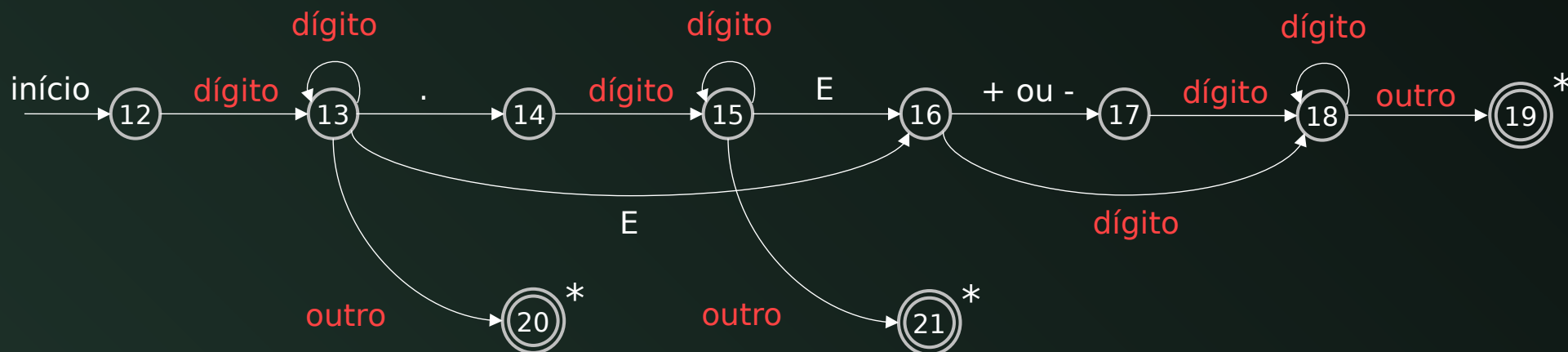


Diagramas de Transição

- Reconhecimento de **números**

nu
m

 *dígitos*(. *dígitos*)?(E[+-]? *dígitos*)?



Diagramas de Transição

- Reconhecimento de **espaços em branco**
 - O **token delim** representa os espaços em branco
 - Tipicamente são espaços, tabulações e quebras de linha
 - Podem ser outros caracteres que a linguagem deseja ignorar
Ex.: comentários, documentação, etc.

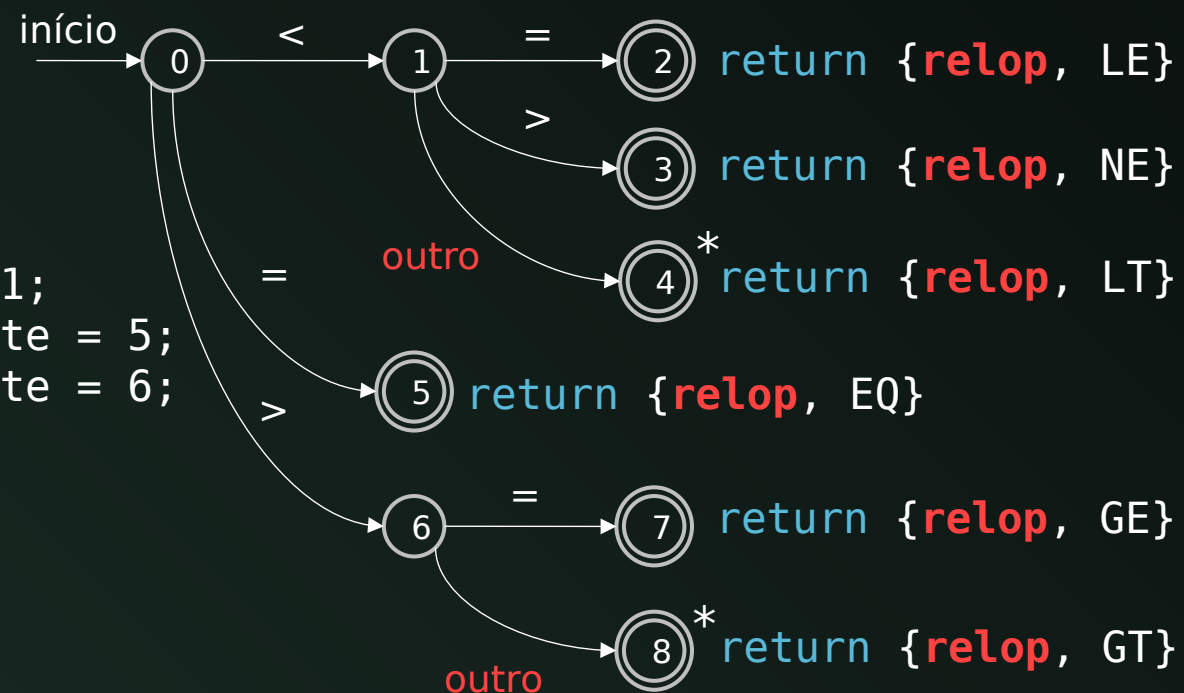


Analizador Léxico

- Um **diagrama de transição** pode ser **traduzido em código** e implementado em uma linguagem de programação
 - A estratégia geral é a seguinte:
 - Uma variável indica o **estado** corrente
 - Um **switch** seleciona um caminho com base no estado corrente
 - Cada estado é **transformado em código** dentro de um case do switch
- Normalmente o código de um estado é também uma **instrução de desvio** que determina o **próximo estado** examinando o **próximo caractere** da entrada

Analizador Léxico

```
Token GetRelop( )
{
    Token t = Token(relop);
    while (true) {
        switch(state) {
            case 0: c = GetChar();
                    if (c == '<') state = 1;
                    else if (c == '=') state = 5;
                    else if (c == '>') state = 6;
                    else fail();
                    break;
            ...
            case 8: UngetChar();
                    t.attribute = GT;
                    return t;
        }
    }
}
```



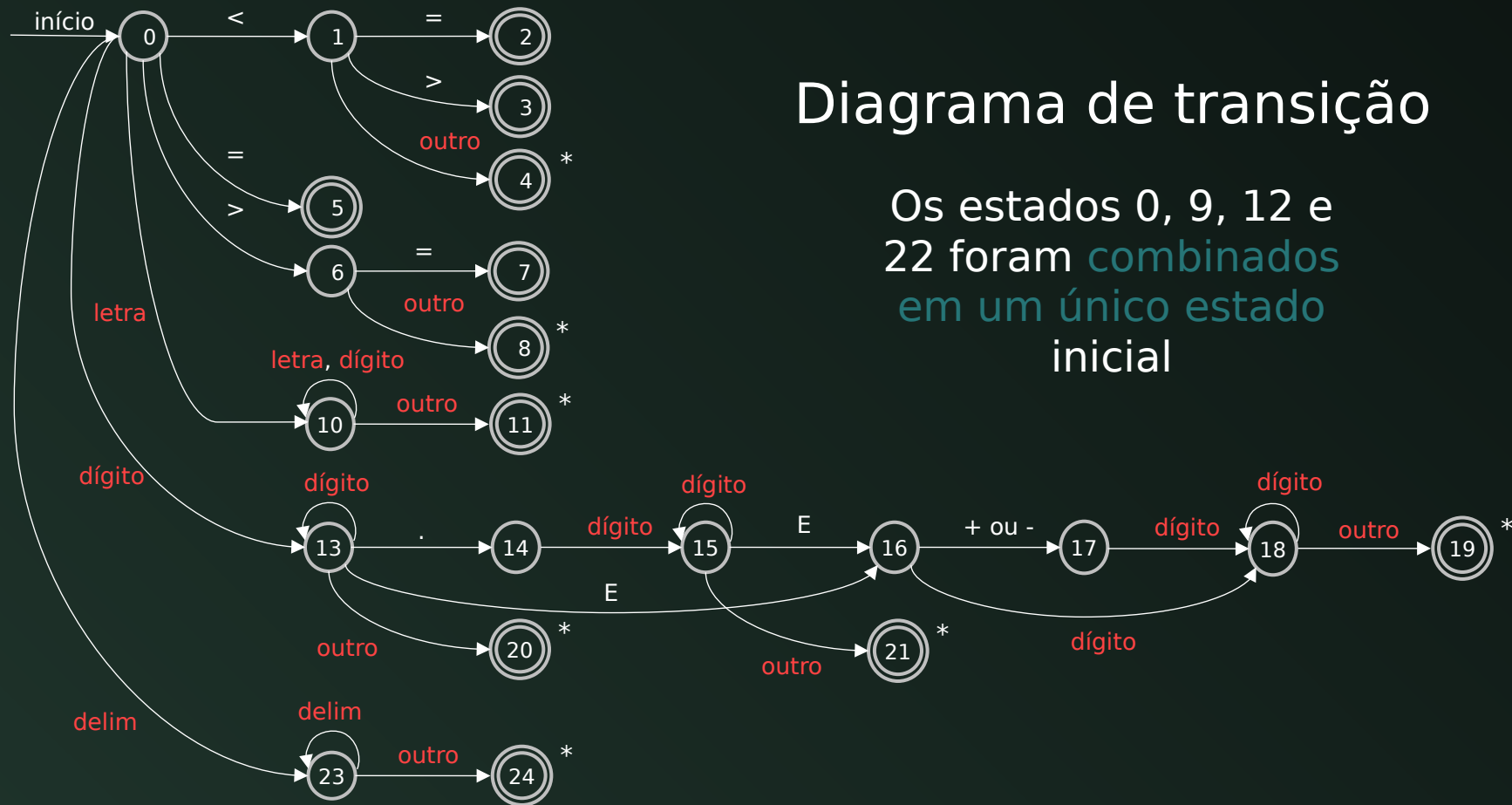
Analizador Léxico

- O que `fail()` faz depende da **estratégia de recuperação de erro**
 - Ele deve trazer o apontador do "caractere corrente" **de volta ao início** do lexema não reconhecido
 - Ele deve permitir que **outro diagrama de transição** seja aplicado
 - Mudar para o estado inicial de outro diagrama
 - Realizar a pesquisa de um outro token
 - Se não houver outro diagrama para usar, ele pode:
 - Mostrar uma **mensagem de erro**
 - **Registrar o erro** e tentar continuar para o próximo lexema

Analizador Léxico

- Um analisador léxico pode ser **construído** a partir de **uma coleção de diagramas** de transição
 - Existem **várias estratégias** possíveis:
 - Os diagramas de cada **token** podem ser **testados sequencialmente**
 - Método permite usar um diagrama para cada palavra-chave
 - Executar os diversos diagramas **em paralelo**
 - Alguns podem terminar antes mas é preciso ir até o fim em todos
 - Pega-se a cadeia mais longa (**then**ext é um id)
 - **Combinar todos os diagramas** em um único (preferível)
 - Combinar os estados 0, 9, 12 e 22 em um único estado inicial

Analizador Léxico



Exercício

1. Construa **diagramas de transição** para reconhecer os padrões das expressões regulares abaixo:

a) $a(a|b)^*a$

b) $((\epsilon|a)b^*)^*$

c) $a^*ba^*ba^*ba^*$

d) $(a|b)^*a(a|b)(a|b)$

Exercício

1. Construa **diagramas de transição** para reconhecer os padrões das expressões regulares abaixo:

a) $a(a|b)^*a$

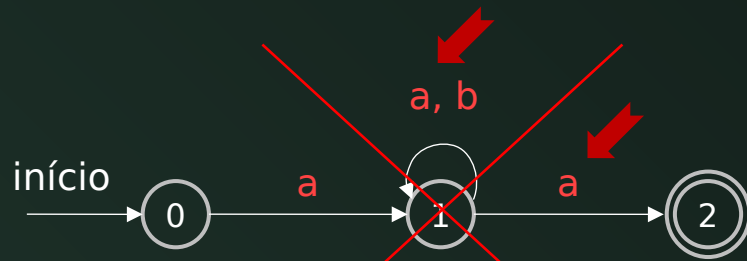
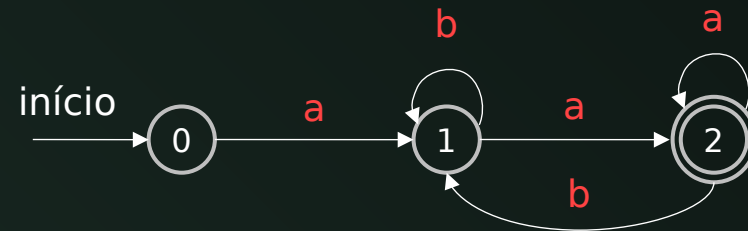


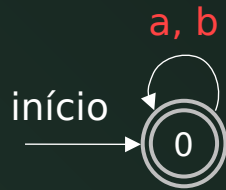
diagrama tem que ser
determinístico



Exercício

1. Construa **diagramas de transição** para reconhecer os padrões das expressões regulares abaixo:

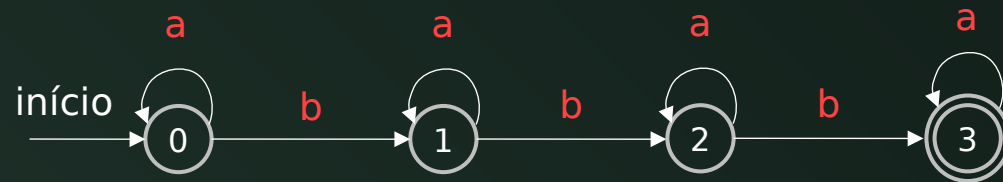
b) $((\epsilon|a)b^*)^*$



Exercício

1. Construa **diagramas de transição** para reconhecer os padrões das expressões regulares abaixo:

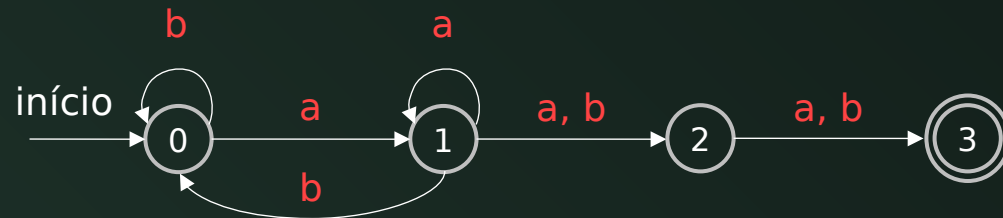
c) $a^*ba^*ba^*ba^*$



Exercício

1. Construa **diagramas de transição** para reconhecer os padrões das expressões regulares abaixo:

d) $(a|b)^*a(a|b)(a|b)$



Resumo

- As principais **tarefas de um analisador léxico** são:
 - Ler os caracteres da entrada
 - Agrupá-los em lexemas
 - Produzir uma sequência de **tokens**
- As **expressões regulares** formam a base de um reconhecedor
 - Permitem **especificar os padrões** dos tokens
 - Podem ser transformadas em **diagramas de transição**
 - Diagramas de transição podem ser implementados em **código**