Judson Santos Santiago

Autômatos Finitos

Compiladores

Introdução

- As expressões regulares formam a base de um reconhecedor
  - Permitem especificar padrões em textos
  - Podem ser transformadas em diagramas de transição
  - Diagramas de transição podem ser simulados em código

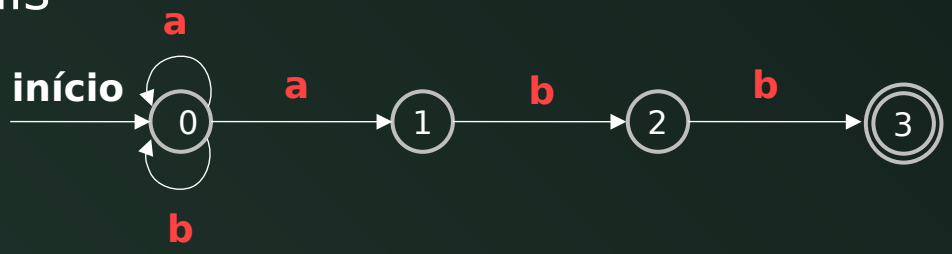
- Com expressões regulares é possível criar **analisadores léxicos**
  - Os diagramas de transição são autômatos finitos
  - Eles podem ser gerados automaticamente

Introdução

- O Flex transforma expressões regulares em um analisador léxico utilizando autômatos finitos
- Os autômatos finitos se classificam em:
  - Autômatos Finitos Não-deterministas (NFA)
    - Um símbolo pode rotular várias arestas saindo do mesmo estado
    - Aceita a cadeia vazia, ϵ, como o rótulo de uma aresta
  - Autômatos Finitos Deterministas (DFA)
    - Um estado não pode ter mais de uma aresta com o mesmo símbolo
  - Ambos reconhecem as mesmas linguagens regulares[†]

Autômatos Finitos

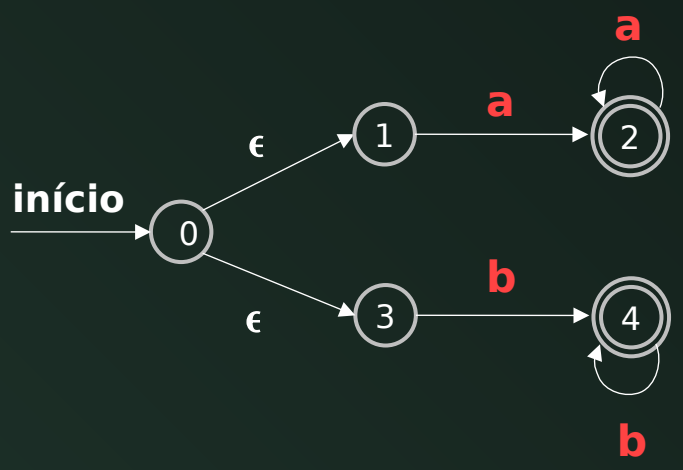
- Um **NFA** consiste em:
  1. Um conjunto finito de estados S
  2. Um conjunto de símbolos de entrada, o alfabeto ❮
  3. Uma função de transição que dá, para cada estado e para cada símbolo em ❮ $\{\epsilon\}$, um conjunto de estados seguintes
  4. Um estado s_0 de S é indicado como estado inicial
  5. Um conjunto de estados F, subconjunto de S, indicados como estados finais



NFA para a expressão regular (a|b)*abb

Autômatos Finitos

- O NFA aceita transições com a cadeia vazia ϵ

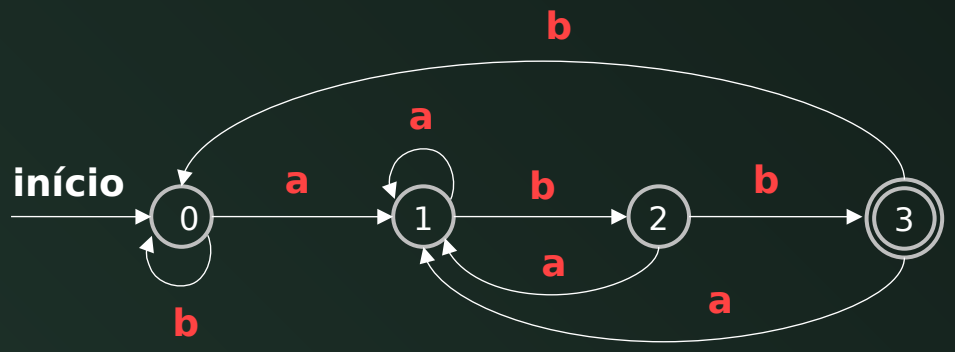


NFA para a expressão regular aa*|bb*

- A cadeia **aaa** é aceita pelo caminho: 0 —ϵ→ 1 —**a**→ 2 —**a**→ 2 —**a**→ 2

Autômatos Finitos

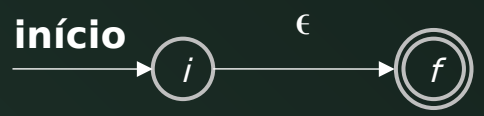
- Um **DFA** é um caso especial de um NFA, em que:
  1. Não existem arestas para a cadeia vazia ϵ
  2. Para cada estado *s* e símbolo ***a***, existe exatamente uma aresta saindo de *s* rotulada com ***a***



DFA para a expressão regular (a|b)*abb

Conversão RegExp em NFA

- Qualquer expressão regular pode ser convertida em um NFA
  - Algoritmo de McNaughton-Yamada-Thompson:
    - A expressão regular é desmembrada em expressões básicas
    - Existem regras para converter expressões básicas em NFAs

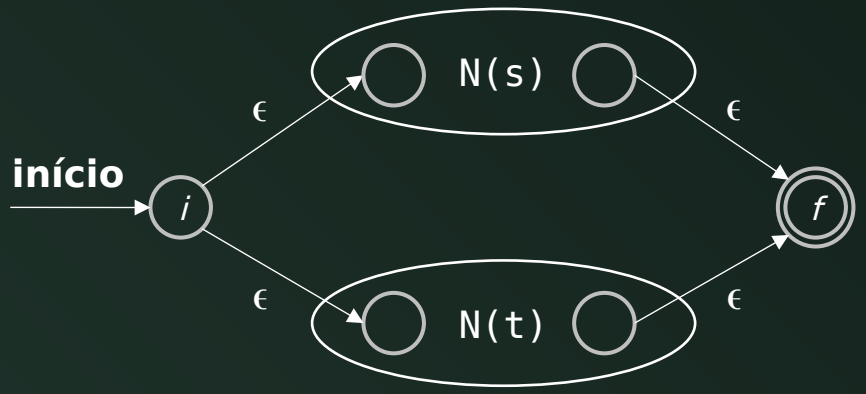


Expressão ϵ

Para qualquer *a* em ❮

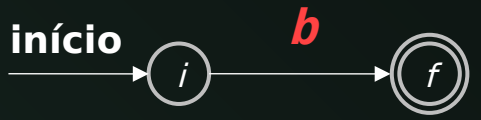
Conversão de RegExp em NFA

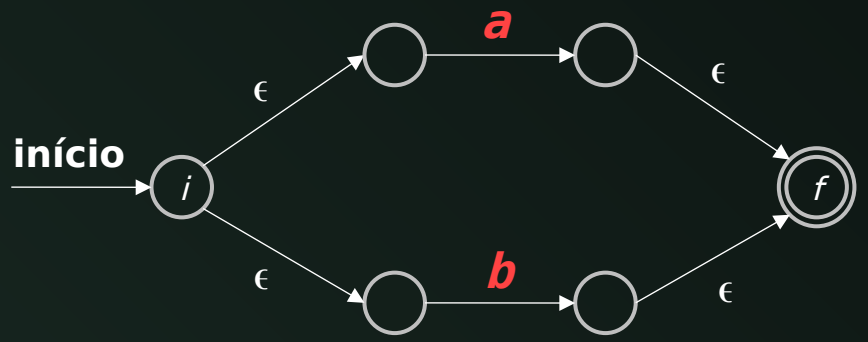
- Considerando que:
  - `N(s)` é o NFA para a expressão regular `s` e
  - `N(t)` é o NFA para a expressão regular `t`, então
  - `N(r)` é o NFA para a expressão regular `r = s|t`



NFA para a união

Ex.: ***a*|*b***

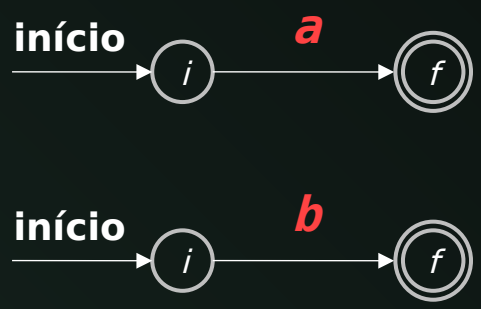


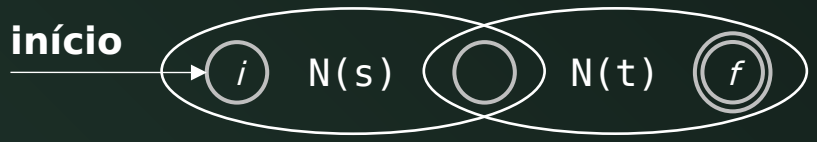


Conversão RegExp em NFA

- Considerando que:
  - `N(s)` é o NFA para a expressão regular `s` e
  - `N(t)` é o NFA para a expressão regular `t`, então
  - `N(r)` é o NFA para a expressão regular `r = st`

Ex.: ***ab***

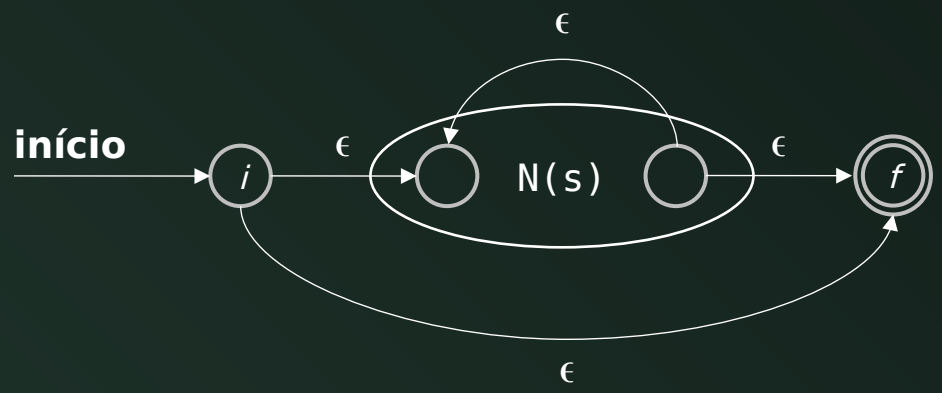




NFA para a concatenação

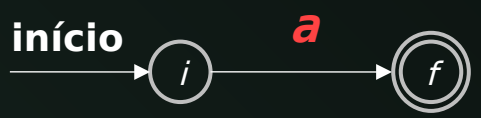
Conversão RegExp em NFA

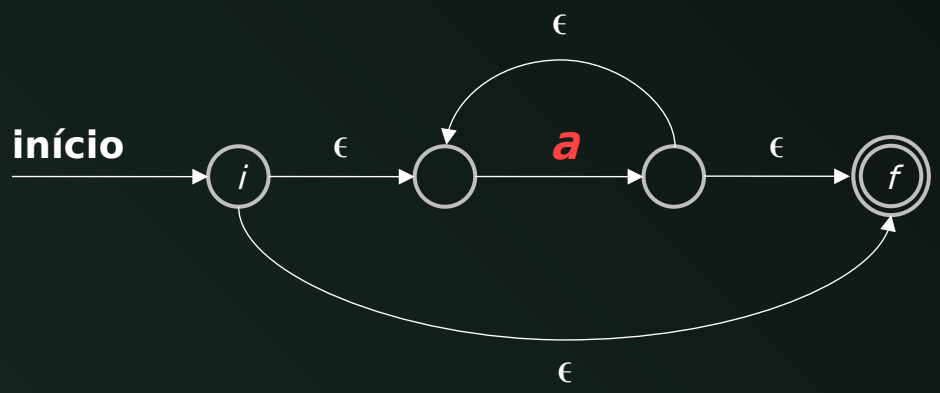
- Considerando que:
  - `N(s)` é o NFA para a expressão regular `s` então
  - `N(r)` é o NFA para a expressão regular `r = s*`



NFA para o fechamento

Ex.: ***a****

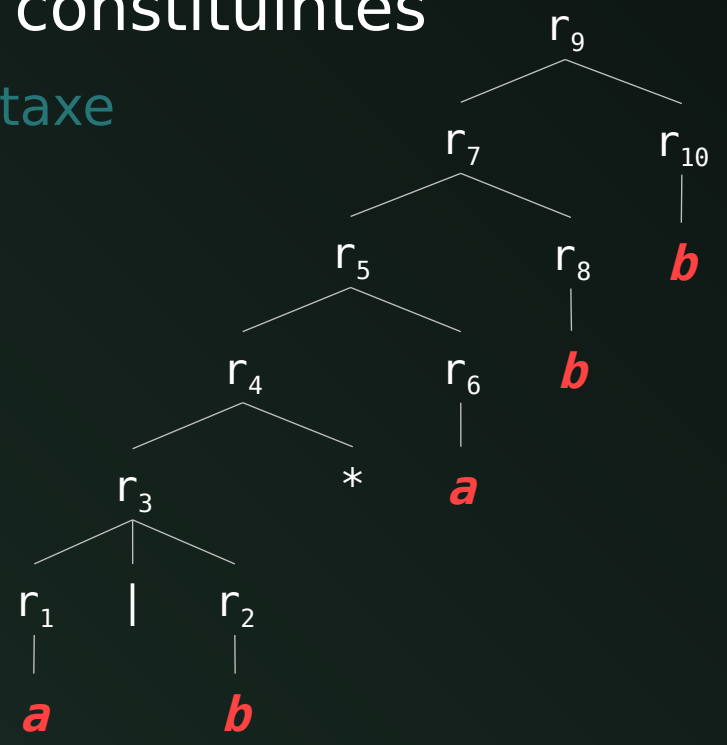




Conversão RegExp em NFA

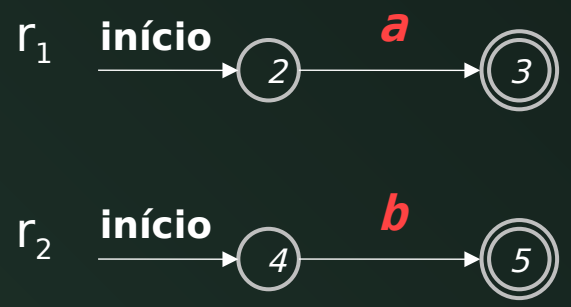
- Para utilizar as regras de conversão é preciso desmembrar a expressão regular nas suas partes constituintes
  - Isso é feito gerando sua árvore de sintaxe (o algoritmo é dirigido por sintaxe)

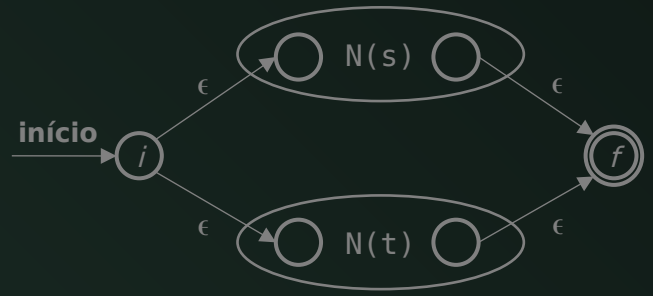
Árvore de sintaxe para
a
expressão regular (a|
b)*abb



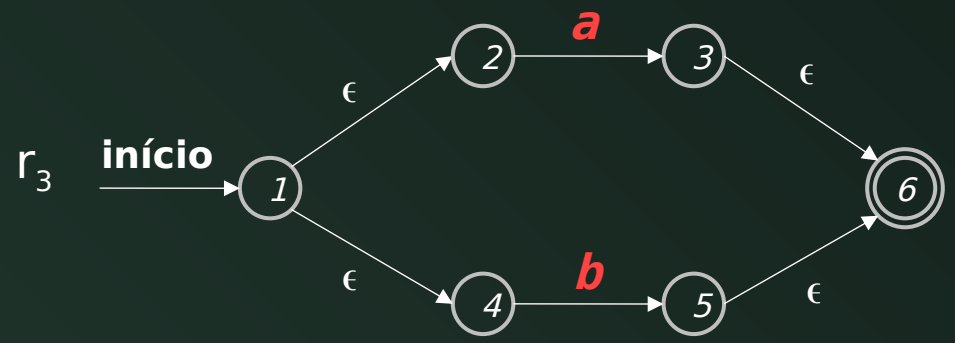
Conversão RegExp em NFA

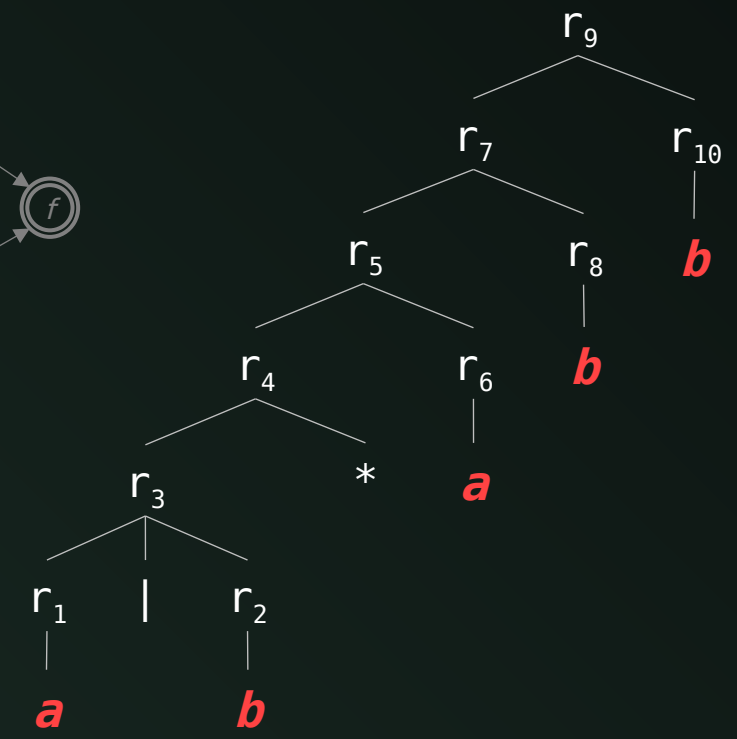
- Para as expressões r_1, r_2 e r_3, construímos:





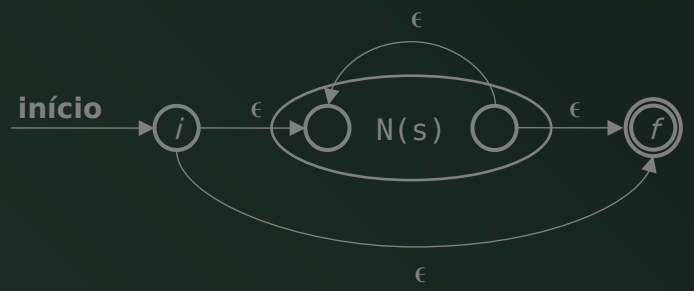
NFA para a união



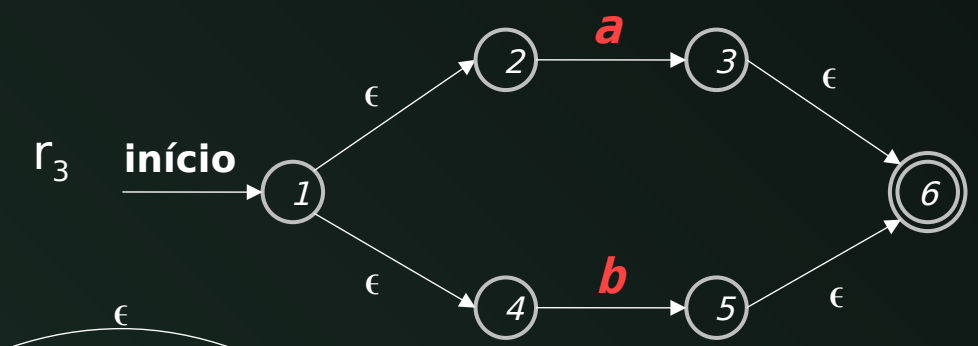


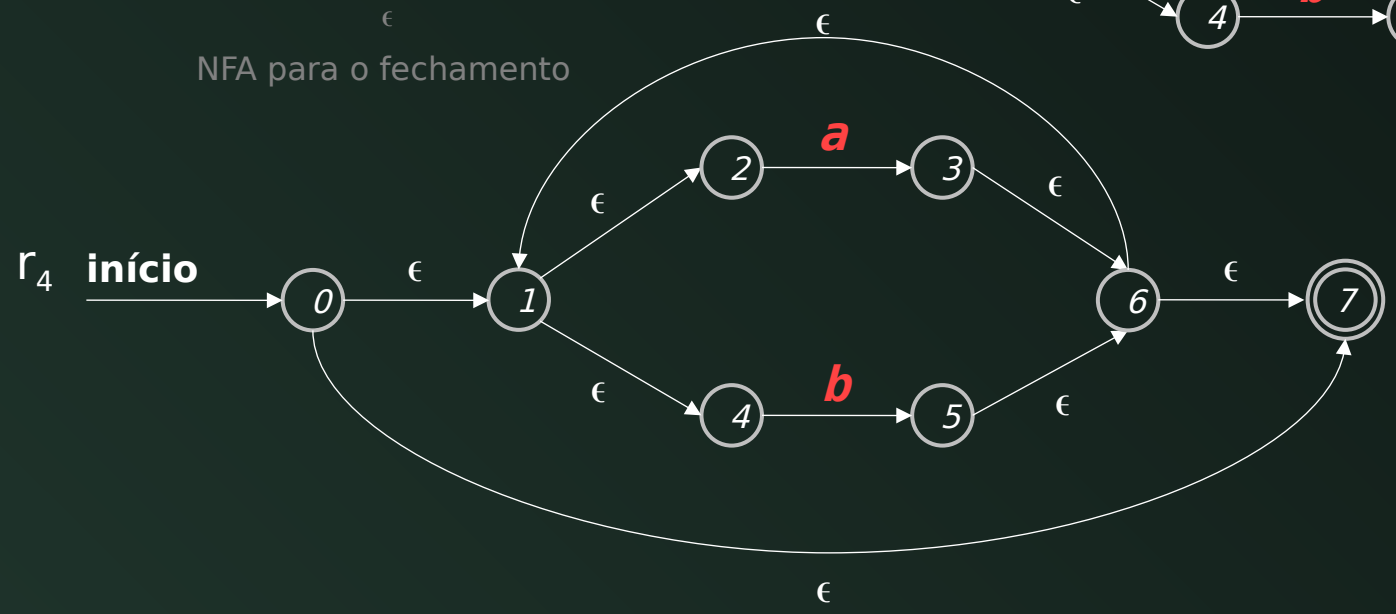
Conversão RegExp em NFA

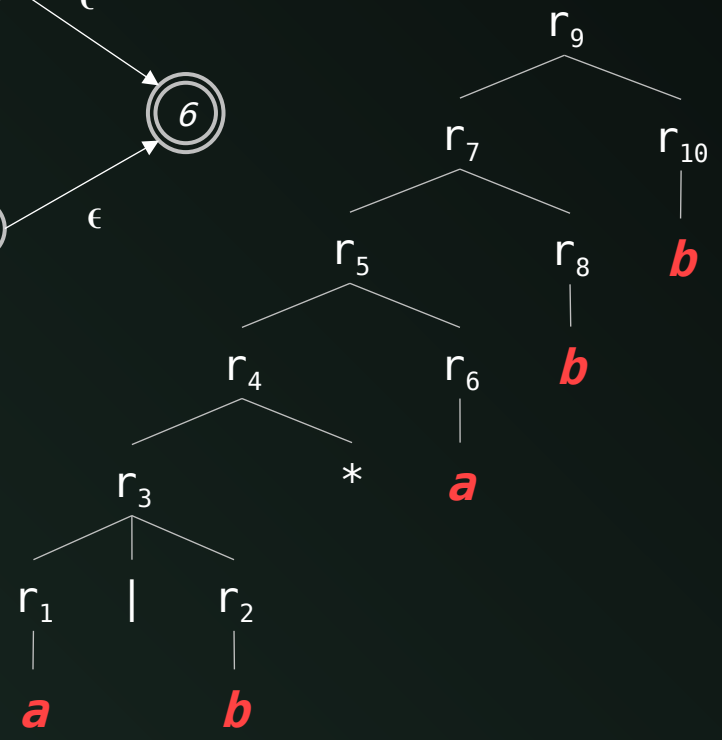
- Para a expressão r_4, temos:



NFA para o fechamento

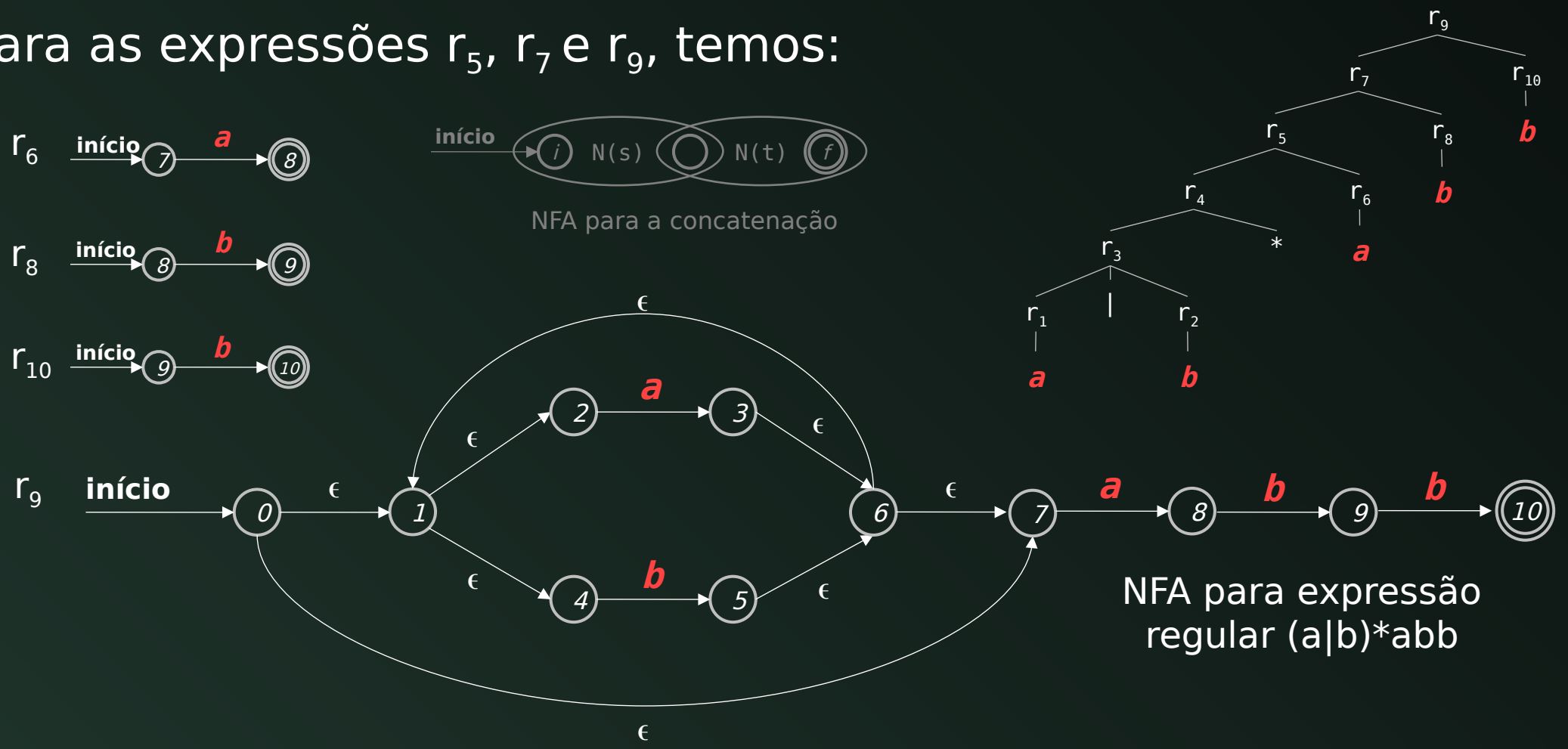






Conversão RegExp em NFA

- Para as expressões r_5, r_7 e r_9, temos:



NFA para expressão regular (a|b)*abb

Expressões Regulares e Autômatos

- As expressões regulares são usadas para descrever:
  - Analisadores léxicos
  - Softwares de casamento de padrão (ex.: `grep`)
- A implementação requer a simulação de um NFA ou DFA
  - A simulação de um NFA não é tão simples:
    - Possui múltiplas escolhas para um símbolo da entrada
    - Pode usar a transição vazia (ϵ)
  - A simulação de um NFA é usada em aplicações como o `grep`
  - Para o analisador léxico é mais eficiente converter o NFA em um DFA

Conversão NFA em DFA

- A técnica é chamada construção de subconjuntos
  - Cada estado do DFA corresponde a vários estados no NFA
  - Lidar com as transições-ϵ é um problema
    - Isso é feito com as funções da tabela abaixo

Operação	Descrição
fecho-ϵ(s)	Conjunto de estados do NFA que podem ser alcançados a partir do estado s usando apenas transições-ϵ
fecho-ϵ(T)	Conjunto de estados do NFA que podem ser alcançados a partir de algum estado do conjunto T usando apenas transições-ϵ
move(T,a)	Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T

s representa um único estado e T representa um conjunto de estados do NFA

Conversão NFA em DFA

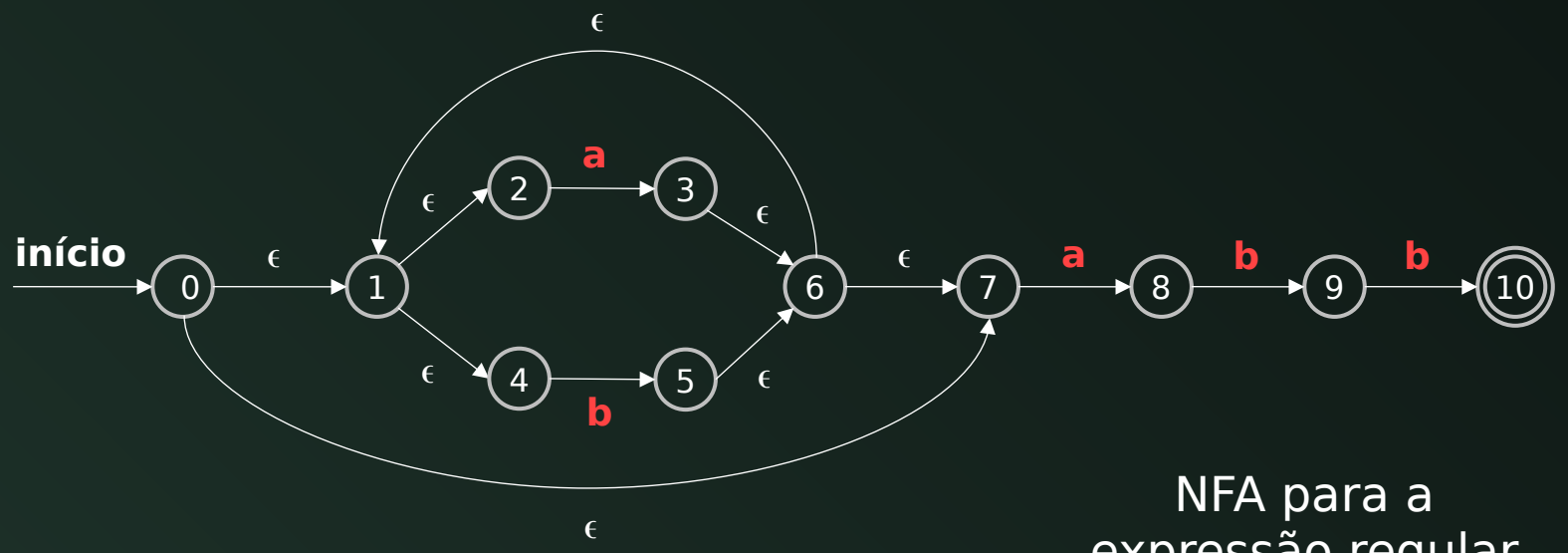
- A construção de um conjunto de estados D_{states} e sua função de transição D_{tran} são feitas pelo algoritmo:

Algoritmo: construção de subconjuntos

```
inicialmente, fecho-ϵ(s0) é o único estado em Dstates e não está marcado
while (existe um estado não marcado T em Dstates) {
    marcar T;
    for (cada símbolo de entrada a) {
        U = fecho-ϵ(move(T,a));
        if (U não está em Dstates)
            inclua U como um estado não marcado em Dstates;
        Dtran[T,a] = U;
    }
}
```

Conversão NFA em DFA

- Aplicando o algoritmo sobre o NFA:



NFA para a expressão regular (a|b)*abb

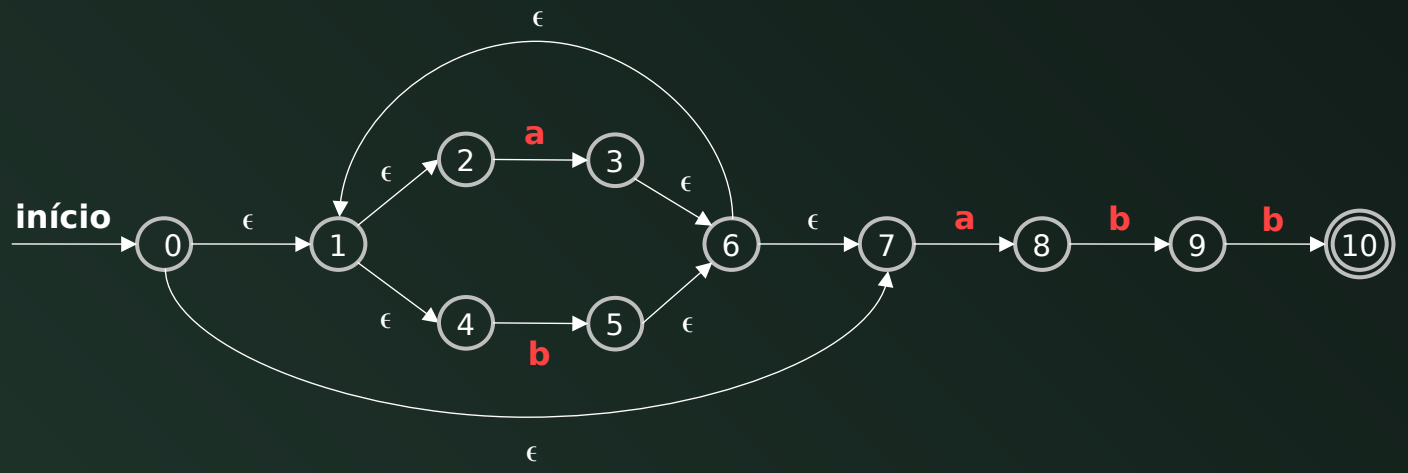
Conversão NFA em DFA

inicialmente, fecho-ϵ(s_0) é o único estado em D_{states} e não está marcado

fecho-ϵ(s)	Conjunto de estados do NFA que podem ser alcançados a partir do estado s usando apenas transições-ϵ

fecho-ϵ(0) = {0, 1, 2, 4, 7} = A

D_{states} = { A }



NFA para a expressão regular (a|b)*abb

Conversão NFA em DFA

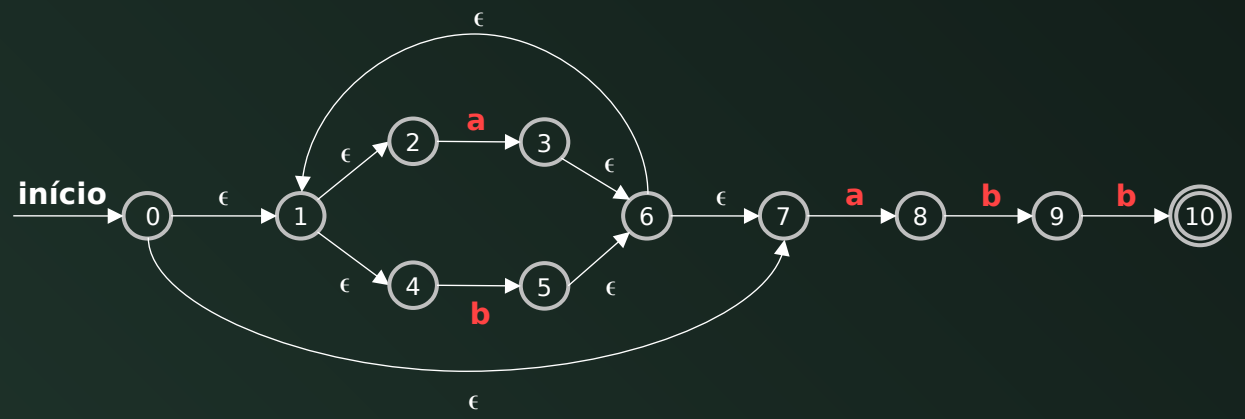
```
while (existe um estado não marcado T em D_states) {
       marcar T;
       for (cada símbolo de entrada a) {
           U = fecho-ϵ(move(T,a));
```

move(T,a)	Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T

```
fecho-ϵ(move(A,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B
fecho-ϵ(move(A,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C
```

D_{states} = { A }

A = {0,1,2,4,7}



NFA para a expressão regular (a|b)*abb

Conversão NFA em DFA

```
if (U não está em Dstates)
    inclua U como um estado não marcado em Dstates;
Dtran[T,a] = U;
```

fecho-ϵ(move(A,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

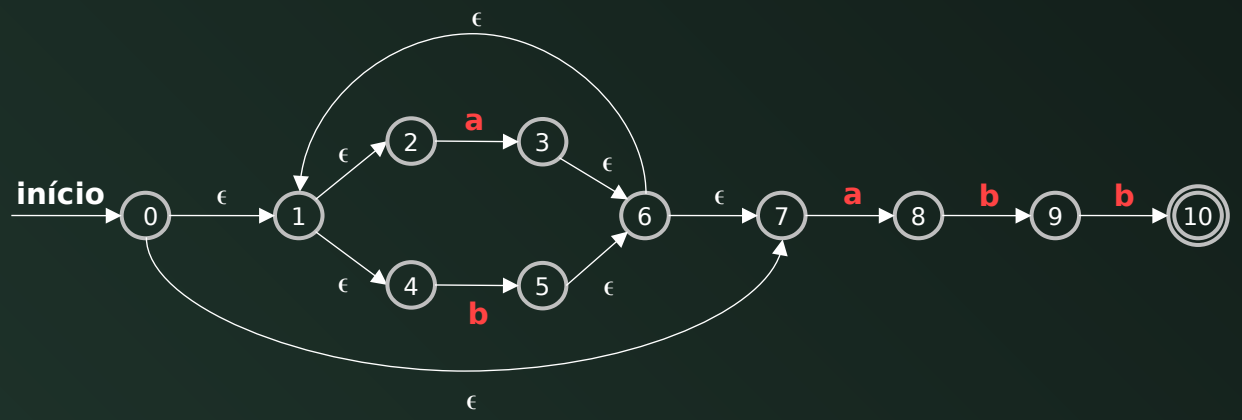
fecho-ϵ(move(A,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C

D_{states} = { A,B,C }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}



D_{tran}

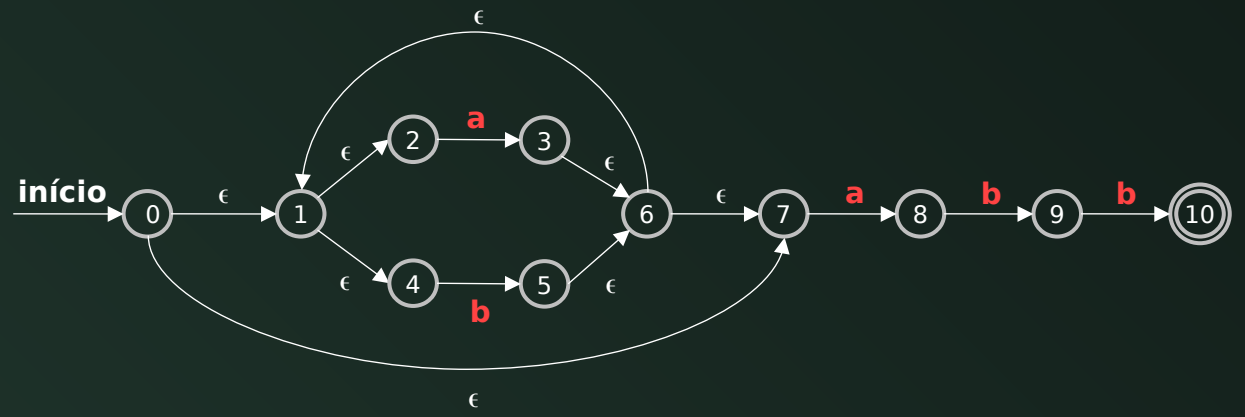
Estado	Símb.	Próx.
A	a	B
A	b	C

Conversão NFA em DFA

```
while (existe um estado não marcado T em D_states) {
        marcar T;
        for (cada símbolo de entrada a) {
            U = fecho-ϵ(move(T,a));
```

move(T,a) — **Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T**

```
fecho-ϵ(move(B,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B
fecho-ϵ(move(B,b)) = fecho-ϵ({5,9}) = {1,2,4,5,6,7,9} = D
```



D_{states} = { A, B, C }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C

Conversão NFA em DFA

```
if (U não está em Dstates)
    inclua U como um estado não marcado em Dstates;
Dtran[T,a] = U;
```

fecho-ϵ(move(B,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

fecho-ϵ(move(B,b)) = fecho-ϵ({5,9}) = {1,2,4,5,6,7,9} = D

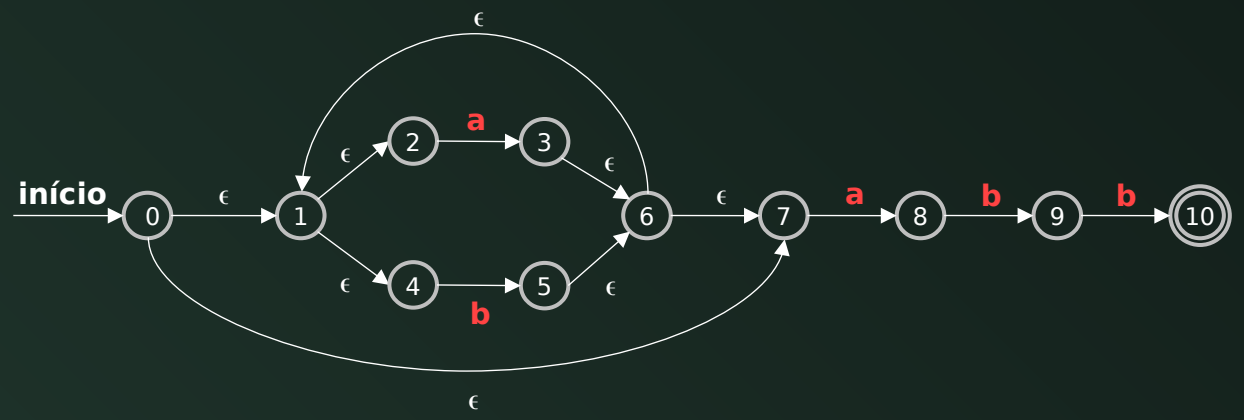
D_{states} = { A,B,C,D }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}



D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D

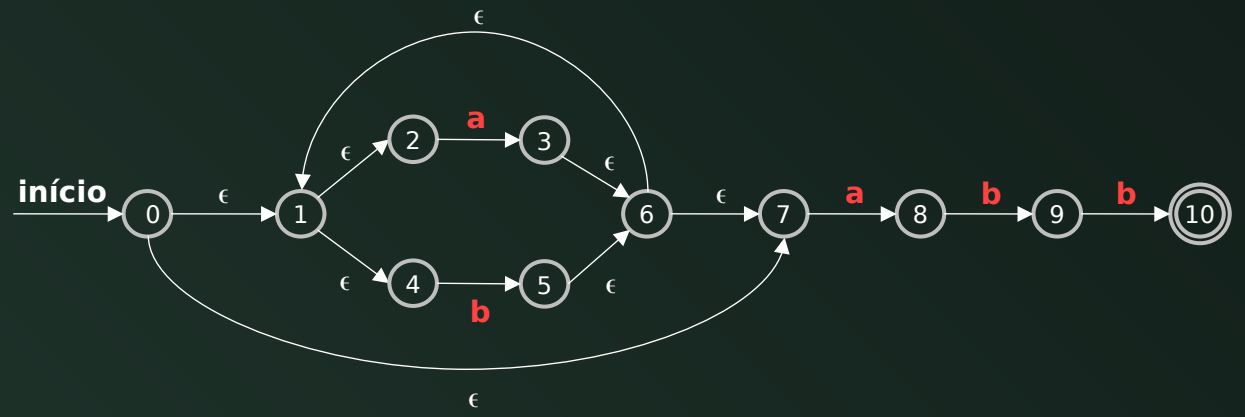
Conversão NFA em DFA

```
while (existe um estado não marcado T em D_states) {
       marcar T;
       for (cada símbolo de entrada a) {
           U = fecho-ϵ(move(T,a));
```

move(T,a) **Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T**

fecho-ϵ(move(C,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

fecho-ϵ(move(C,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C



D_{states} = { A,B,C,D }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}

D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D

Conversão NFA em DFA

> if (U não está em D_{states})
> inclua U como um estado não marcado em D_{states};
> $D_{tran}[T,a] = U;$

fecho-ϵ(move(C,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

fecho-ϵ(move(C,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C

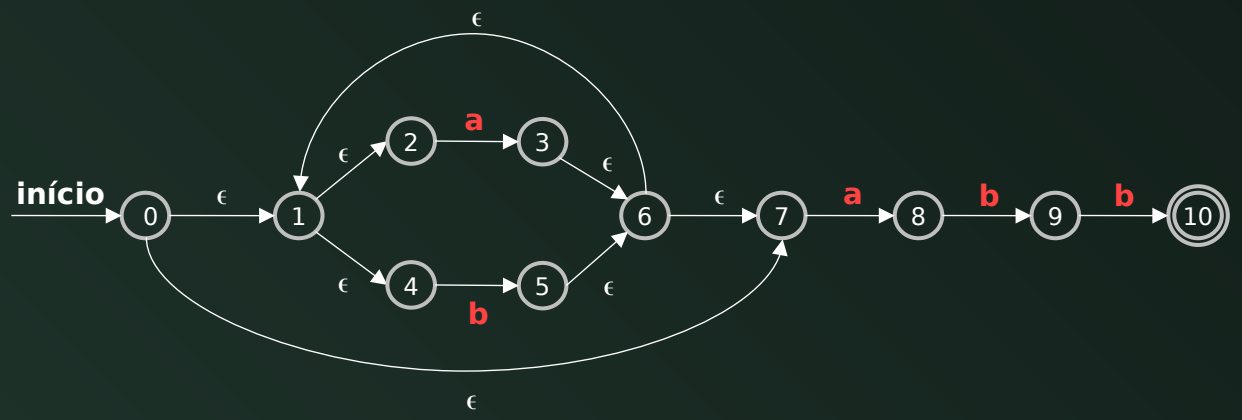
D_{states} = { A,B,C,D }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}



D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	b	D
C	a	B
C	b	C

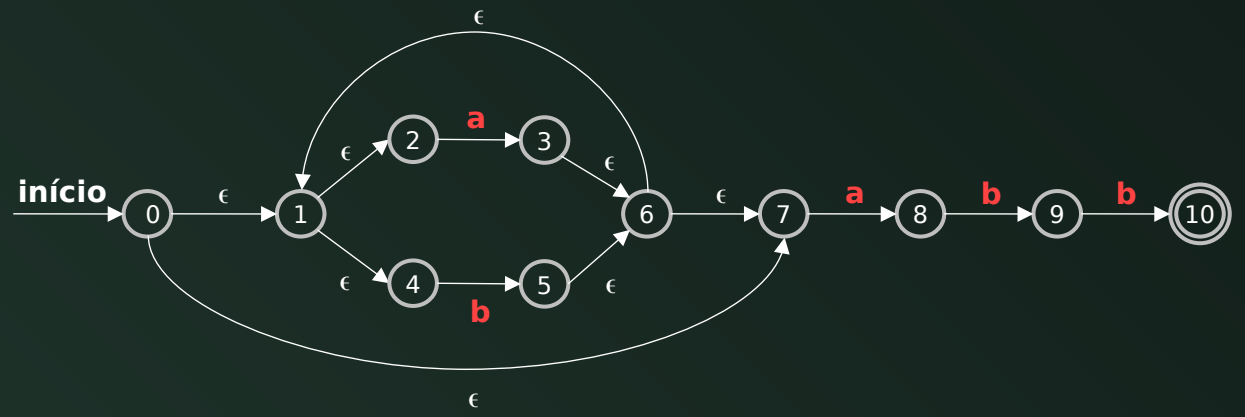
Conversão NFA em DFA

```
while (existe um estado não marcado T em D_states) {
       marcar T;
       for (cada símbolo de entrada a) {
           U = fecho-ϵ(move(T,a));
```

move(T,a)	Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T

fecho-ϵ(move(D,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

fecho-ϵ(move(D,b)) = fecho-ϵ({5,10}) = {1,2,4,5,6,7,10} = E



D_{states} = { A,B,C,D }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}

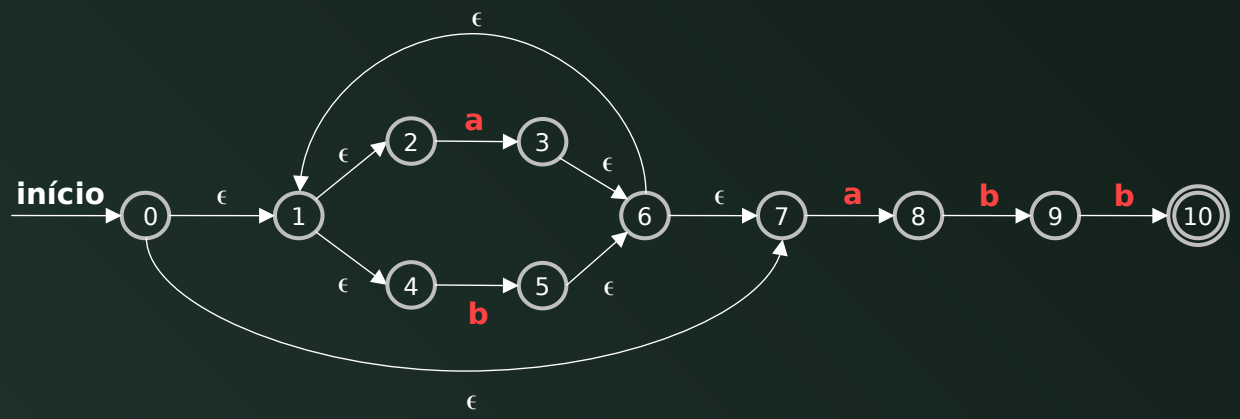
D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D
C	a	B
C	b	C

Conversão NFA em DFA

```
if (U não está em D_states)
    inclua U como um estado não marcado em D_states;
D_tran[T,a] = U;
```

```
fecho-ϵ(move(D,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B
fecho-ϵ(move(D,b)) = fecho-ϵ({5,10}) = {1,2,4,5,6,7,10} = E
```



D_{states} = { A,B,C,D,E }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}

E = {1,2,4,5,6,7,10}

D_{tran}

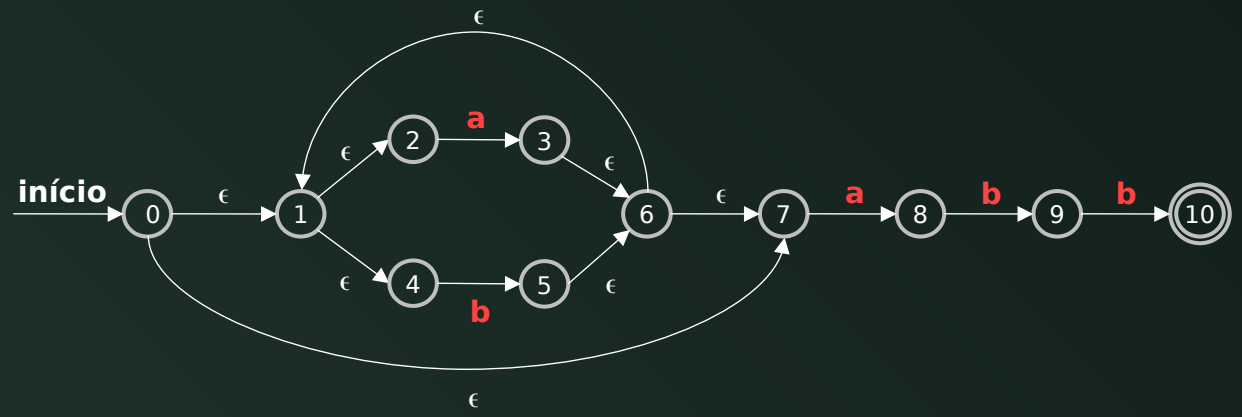
Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D
C	a	B
C	b	C
D	a	B
D	b	E

Conversão NFA em DFA

```
while (existe um estado não marcado T em D_states) {
       marcar T;
       for (cada símbolo de entrada a) {
           U = fecho-ϵ(move(T,a));
```

move(T,a) **Conjunto de estados do NFA para os quais existe uma transição sob o símbolo de entrada a, a partir de algum estado em T**

```
fecho-ϵ(move(E,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B
fecho-ϵ(move(E,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C
```



D_{states} = { A,B,C,D,E }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}

E = {1,2,4,5,6,7,10}

D_{tran}

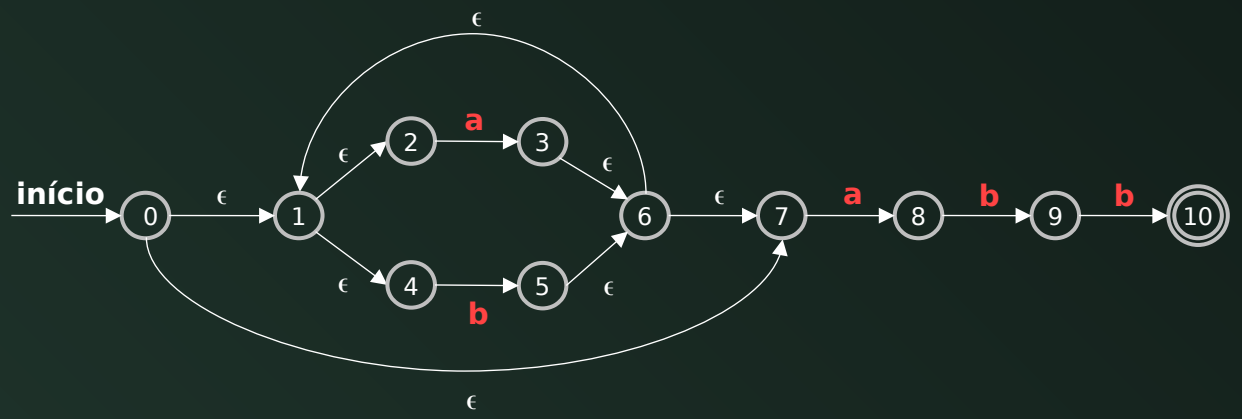
Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D
C	a	B
C	b	C
D	a	B
D	b	E

Conversão NFA em DFA

> if (U não está em D_{states})
> inclua U como um estado não marcado em D_{states};
> $D_{tran}[T,a] = U;$

fecho-ϵ(move(E,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8} = B

fecho-ϵ(move(E,b)) = fecho-ϵ({5}) = {1,2,4,5,6,7} = C



D_{states} = { A,B,C,D,E }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

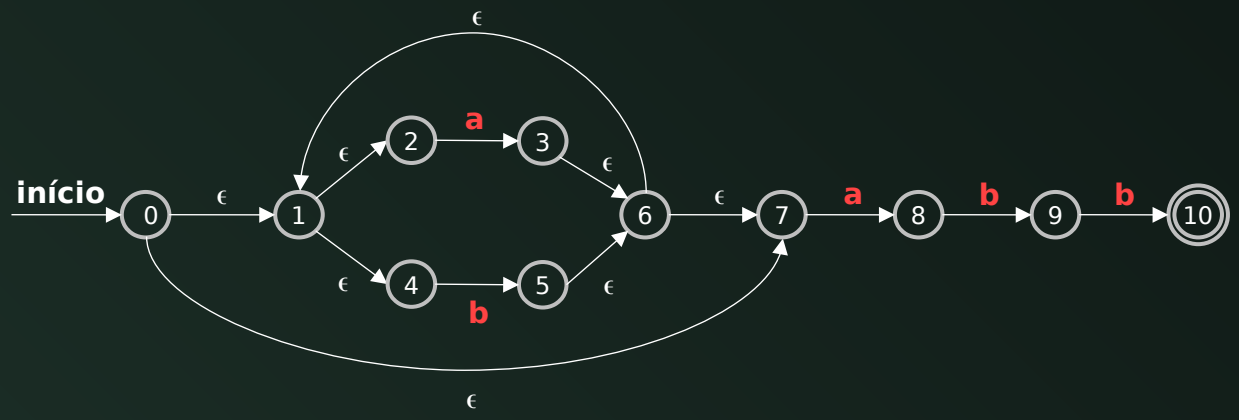
D = {1,2,4,5,6,7,9}

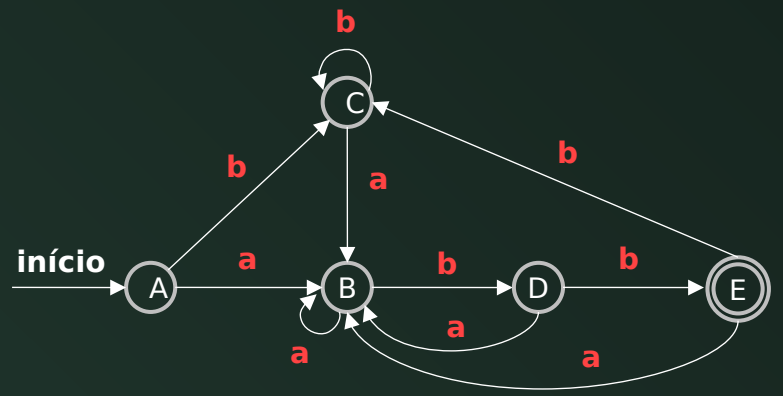
E = {1,2,4,5,6,7,10}

D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D
C	a	B
C	b	C
D	a	B
D	b	E
E	a	B
E	b	C

Conversão NFA em DFA





DFA para a expressão regular (a|b)*abb

D_{states} = { A,B,C,D,E }

A = {0,1,2,4,7}

B = {1,2,3,4,6,7,8}

C = {1,2,4,5,6,7}

D = {1,2,4,5,6,7,9}

E = {1,2,4,5,6,7,10}

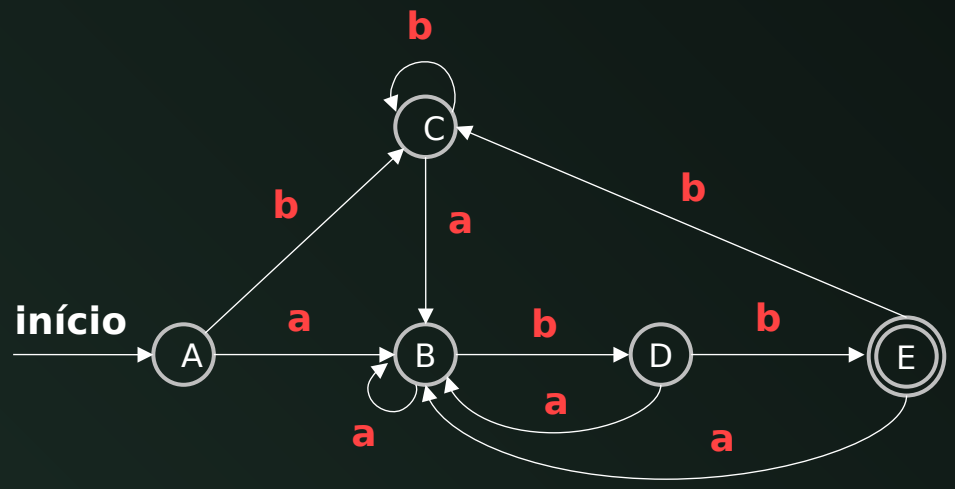
D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	B
B	b	D
C	a	B
C	b	C
D	a	B
D	b	E
E	a	B
E	b	C

Conversão NFA em DFA

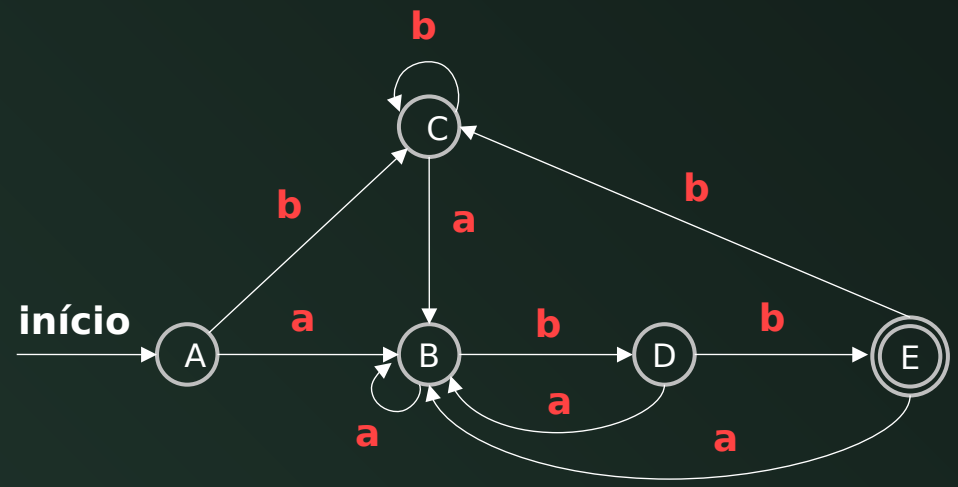
- A conclusão do processo de conversão é garantida
  - O número de estados do DFA é aproximadamente o mesmo do NFA
    - O estado A é o estado inicial
    - O estado E, que contém o estado 10 do NFA, é o único estado final

Estados NFA	Estado DFA	a	b
{0,1,2,4,7}	A	B	C
{1,2,3,4,6,7,8}	B	B	D
{1,2,4,5,6,7}	C	B	C
{1,2,4,5,6,7,9}	D	B	E
{1,2,4,5,6,7,10}	E	B	C

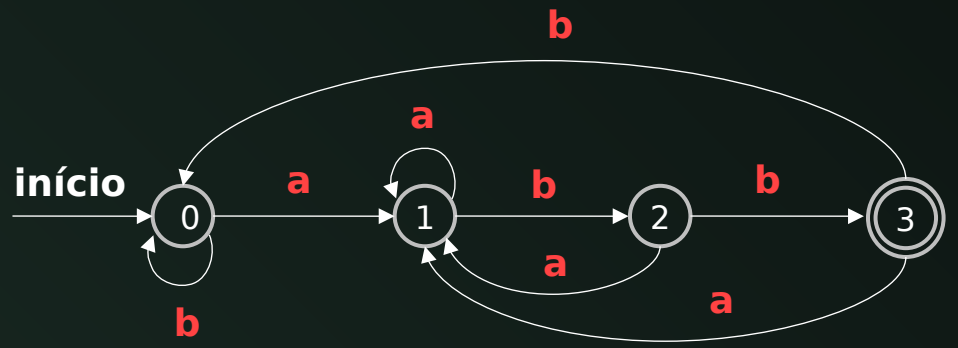


Conversão NFA em DFA

- O DFA resultante possui um estado a mais que o ideal
  - Os estados A e C podem ser combinados
  - Processo de minimização será discutido mais tarde



DFA obtido da conversão



DFA para (a|b)*abb

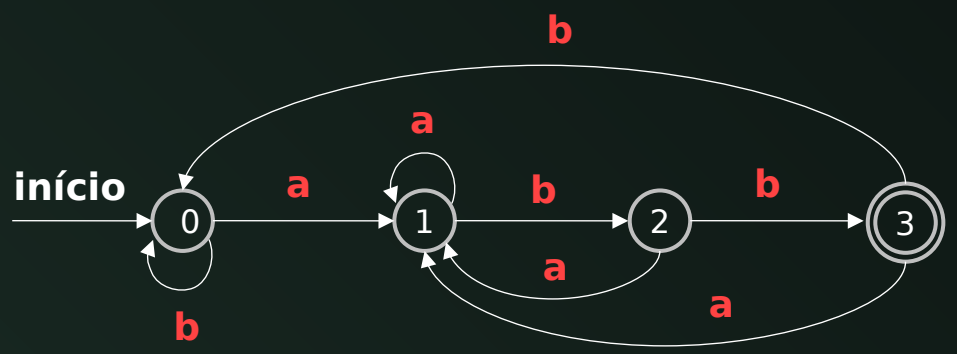
Reconhecimento da Cadeia

- Um NFA é uma representação abstrata de um reconhecedor, enquanto que um DFA é um algoritmo concreto para reconhecer uma cadeia

Algoritmo: Simulando um DFA

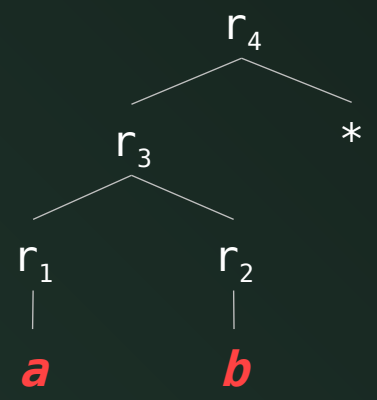
```
s = s0
c = next_char()
while (c != EOF) {
    s = move(s,c);
    c = next_char();
}
if (s está em F) return "sim";
else return "não";
```

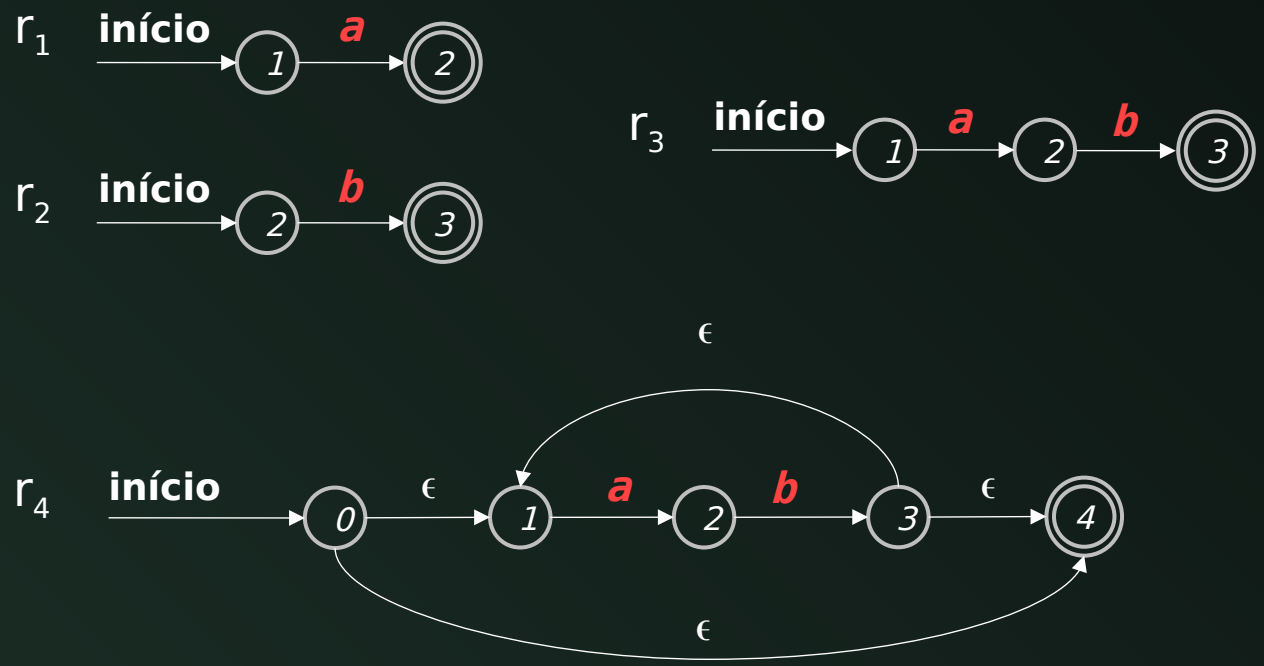
Retorna "sim" para a cadeia **ababb**, passando pelos estados 0, 1, 2, 1, 2, 3



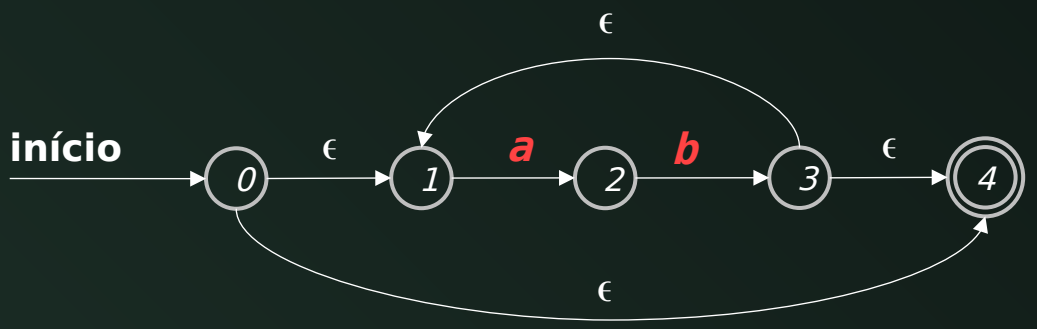
Exercício

1. Converta a expressão regular (ab)* em DFA





Exercício



fecho-ϵ(0) = {0, 1, 4 } = A

fecho-ϵ(move(A,a)) = fecho-ϵ({2}) = {2} = B

fecho-ϵ(move(A,b)) = fecho-ϵ({}) = {}

fecho-ϵ(move(B,a)) = fecho-ϵ({}) = { }

fecho-ϵ(move(B,b)) = fecho-ϵ({3}) = {1,3,4} = C

fecho-ϵ(move(C,a)) = fecho-ϵ({2}) = {2} = B

fecho-ϵ(move(C,b)) = fecho-ϵ({}) = {}

D_{states} = { A, B, C }

A = {0,1,4}

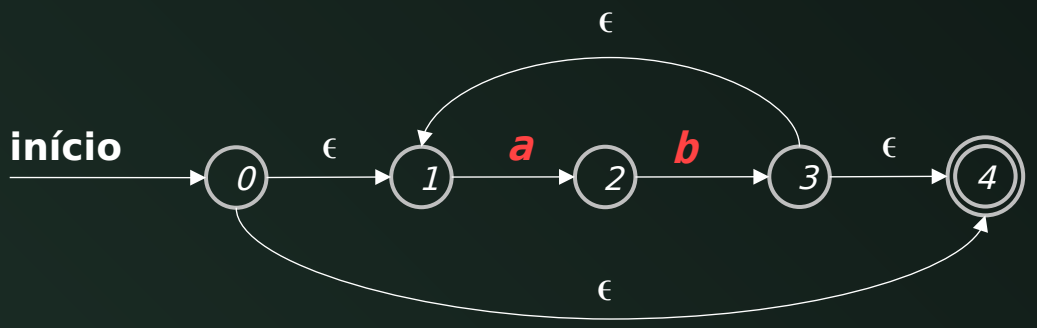
B = {2}

C = {1,3,4}

D_{tran}

Estado	Símb.	Próx.
A	a	B
A	b	-
B	b	C
C	a	B
C	b	-

Exercício

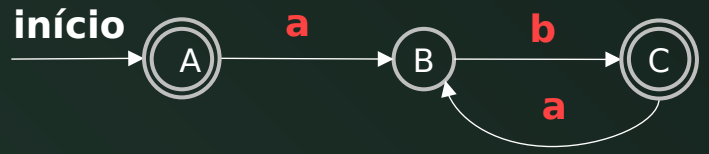


D_{states} = { A, ,C
B }

A = {0,1,4}

B = {2}

C = {1,3,4}



DFA para a expressão regular (ab)*

D_{tran}

Estado	Símb.	Próx.
A	a	B
B	b	-
B	b	C
C	a	B
C	b	-

Resumo

- A ferramenta `Flex` é um gerador de analisador léxico
  - Ela se baseia na simulação de Autômatos Finitos Determinísticos (DFA)
- Os geradores de analisadores léxicos utilizam:
  - Expressões regulares para descrever os padrões dos tokens
    - Expressões regulares são convertidas em NFAs
    - NFAs são convertidos em DFAs
    - DFAs são armazenados
    - O DFA é executado em cima de uma cadeia de entrada