Judson Santos Santiago

Geração de Analisadores Léxicos

Compiladores

Introdução

- Um gerador de analisador léxico
  - Utiliza:
    - Expressões regulares
    - Autômatos Finitos (NFAs e DFAs)
  - Realiza:
    - Conversão de Expressão Regular para NFA
    - Conversão de NFA para DFA
    - Simulação do DFA (reconhecimento de cadeias)

Introdução

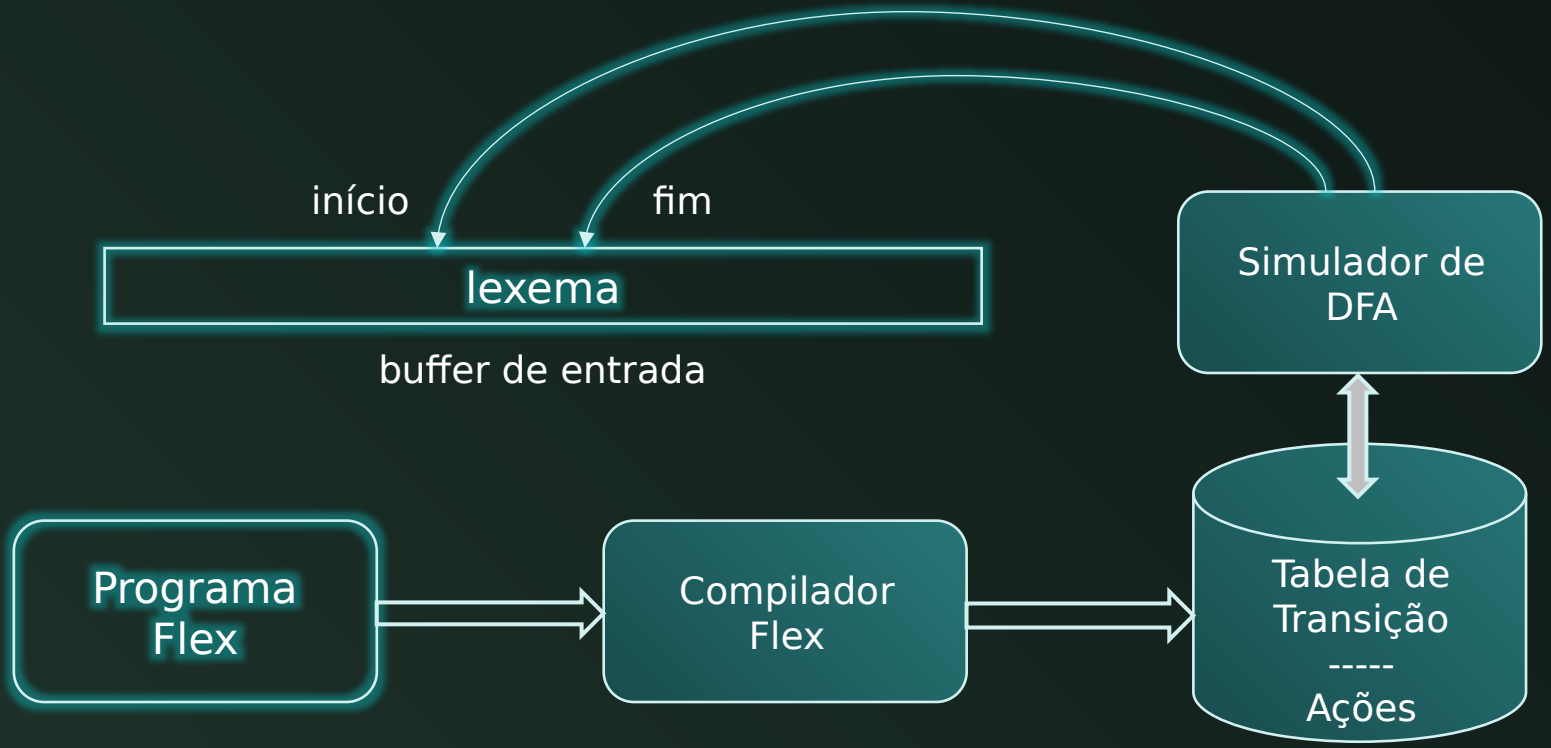
- A entrada para um gerador de analisador léxico
  - É composta por padrões e ações

```
// definições regulares
id          {letra}({letra}|{digito})*
num         {digito}+(\.{digito}+)?(E[+-]?{digito}+)?

%%
// padrões e ações
{brancos}           ;
if          return IF;
then        return THEN;
else        return ELSE;
{id}        return ID;
{num}       return NUM;
.           cout << YYText() << " é um token inválido!";
%%
// funções auxiliares
```

Introdução

- A arquitetura de um analisador léxico gerado pelo Flex



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 `'='`

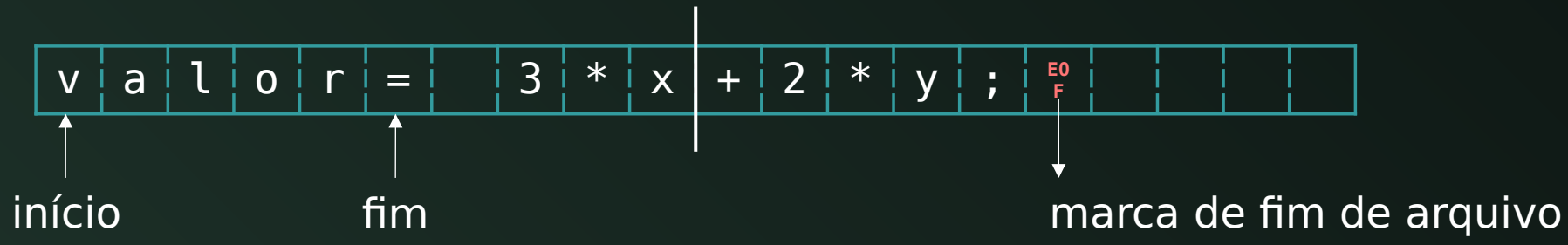
- Essa tarefa pode ser realizada com o uso de buffers
  - Acelera a leitura do código
  - Permite usar *lookaheads* grandes com segurança
- Uma técnica eficiente utiliza dois buffers
  - São carregados alternadamente

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

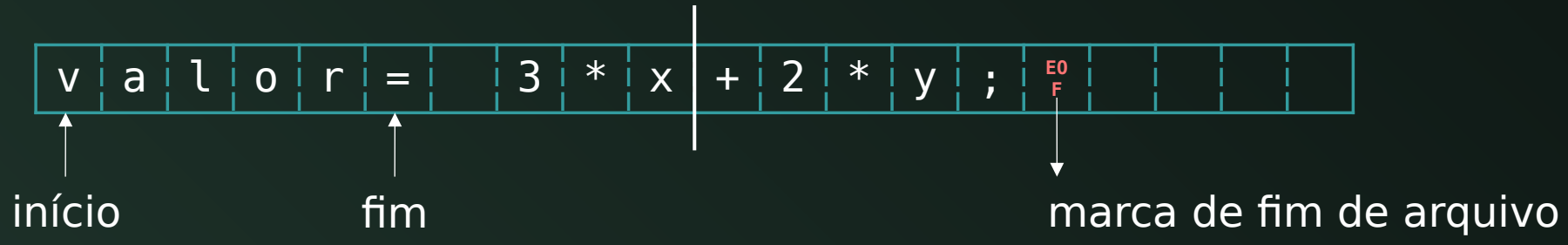
Buffer de Entrada

- 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

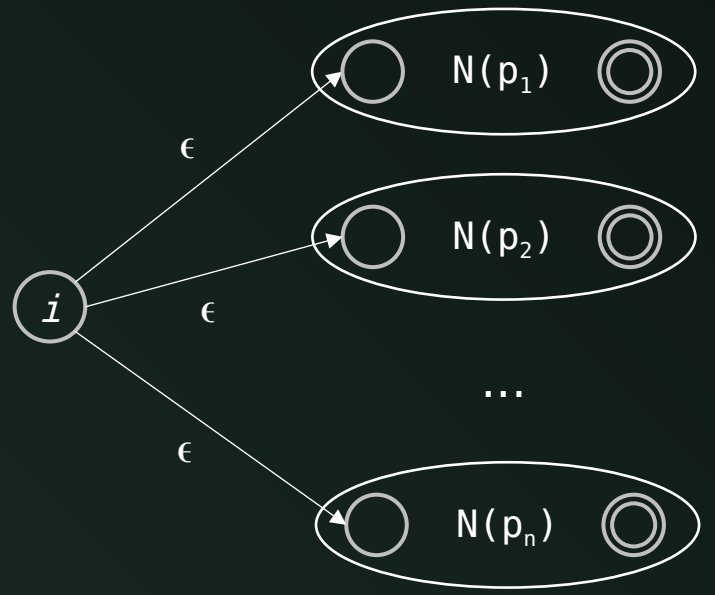
- Avançar o apontador de fim requer
  - Testar se o fim do buffer foi atingido e em caso positivo
    - Carregar o outro buffer
    - Mover o ponteiro fim para o início do buffer recém carregado
- Espaço em buffer pode se esgotar
  - Tamanho do lexema + avanço adiante for maior que N



Construção do Autômato

- Para construir o autômato:
  - Cada padrão *p* de expressão regular é convertido para um NFA
  - Os NFAs individuais são combinados
    - Usando um novo estado inicial
    - Transições-ϵ para cada NFA

Autômato que casa com qualquer um dos padrões no programa Flex



Construção do Autômato

- O exemplo abaixo ilustra o processo:

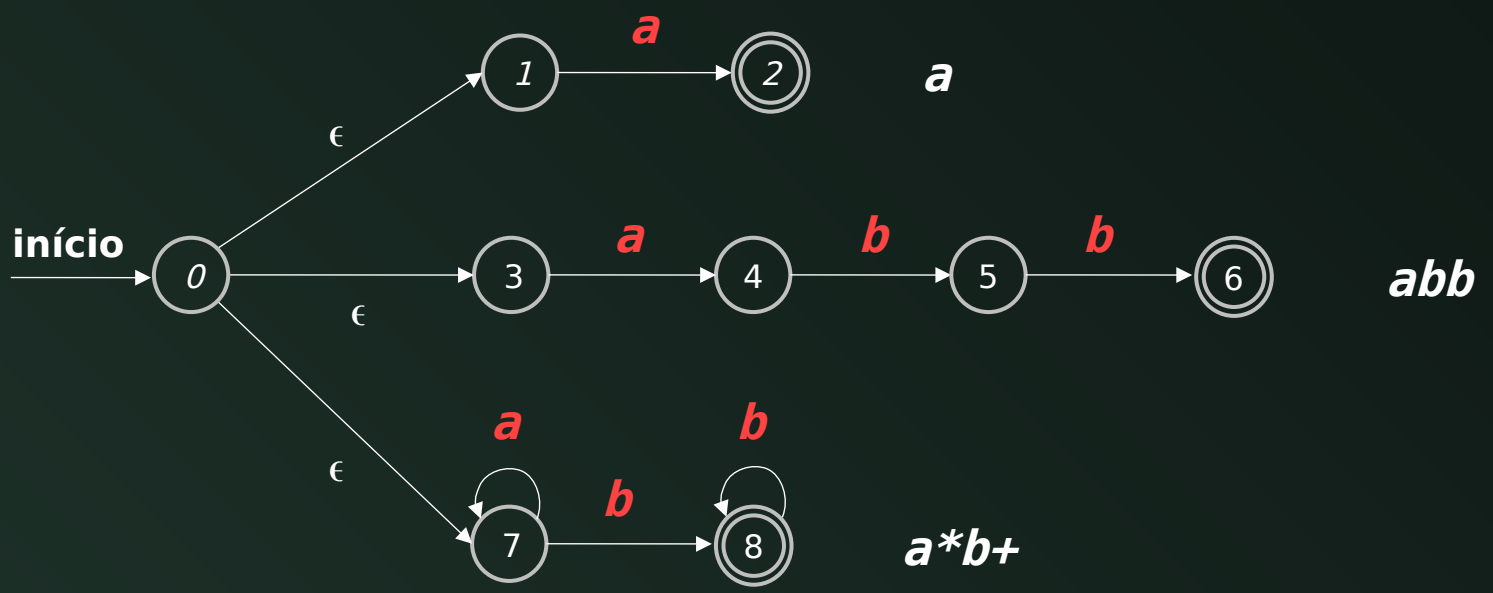
```
a       { ação A1 para padrão p1 }
abb     { ação A2 para padrão p2 }
a*b+    { ação A3 para padrão p3 }
```

Expressões regulares e ações de um programa Flex

  - Os padrões apresentam alguns conflitos:
    - A cadeia abb casa tanto com **abb** quanto com **a*b+**
      - A regra do Flex é casar com o que é listado primeiro
    - A cadeia aabbbb... possui muitos prefixos que casam com a*b+
      - A regra do Flex é reconhecer a cadeia mais longa, de modo que se deve continuar lendo b’s até encontrar um outro caractere diferente

Construção do Autômato

- Os NFAs de cada expressão combinados:



Construção do Autômato

- A conversão do NFA em DFA
  - Requer a execução da construção de subconjuntos
  - Mas como se implementa o cálculo do fecho-ϵ?

Algoritmo: construção de subconjuntos

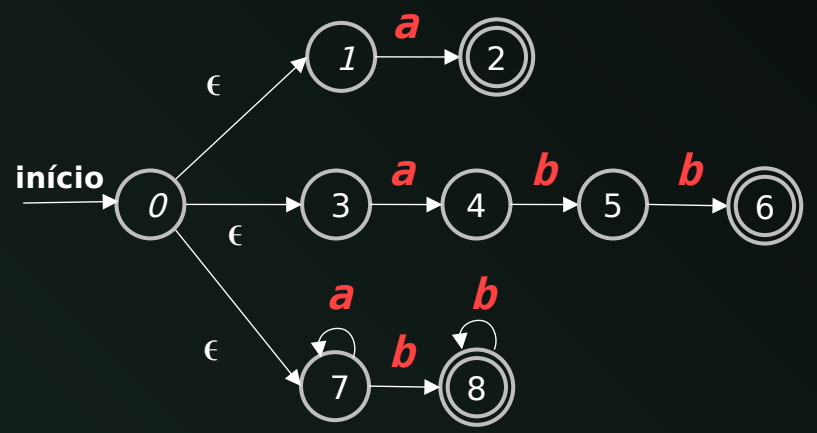
```
inicialmente, fecho-ϵ(s0) é o único estado em Dstates e não está marcado
while (existe um estado não marcado T em Dstates) {
    marcar T;
    for (cada símbolo de entrada a) {
        U = fecho-ϵ(move(T,a));
        if (U não está em Dstates)
            inclua U como um estado não marcado em Dstates;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

Construção do Autômato

- Calculando fecho-ϵ de um conjunto T

Algoritmo: fecho-ϵ(T)

```
insira todos os estados de T em uma pilha;
inicialize fecho-ϵ(T) para T;
while (pilha não está vazia) {
    desempilhe t, o elemento do topo da pilha;
    for (cada estado u com uma aresta-ϵ de t para u) {
        if (u não está em fecho-ϵ(T)) {
            inclua u em fecho-ϵ(T);
            insira u na pilha;
        }
    }
}
```

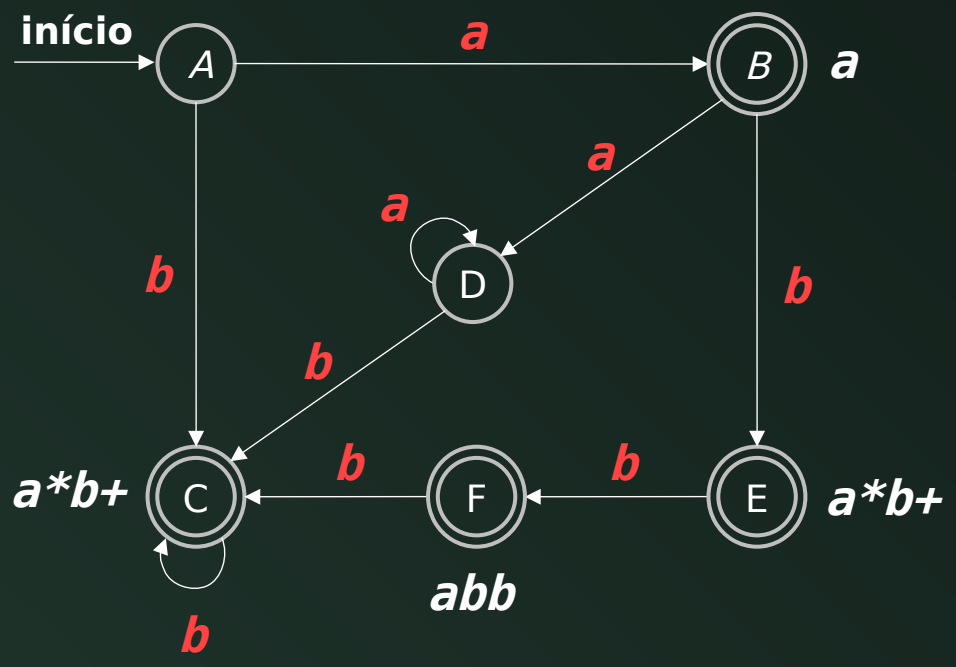


fecho-ϵ(0) = {0,1,3,7}

PILHA

Construção do Autômato

- A conversão do NFA em DFA
  - Os estados de aceitação são associados a uma expressão regular



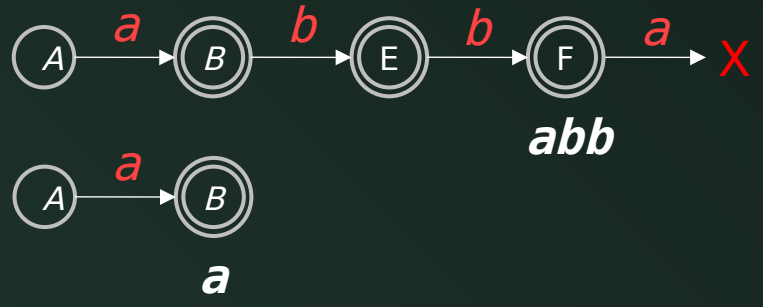

```
A = {0,1,3,7}
B = {2,4,7}      => a
C = {8}          => a*b+
D = {7}
E = {5,8}        => a*b+
F = {6,8}        => abb ou a*b+?
```

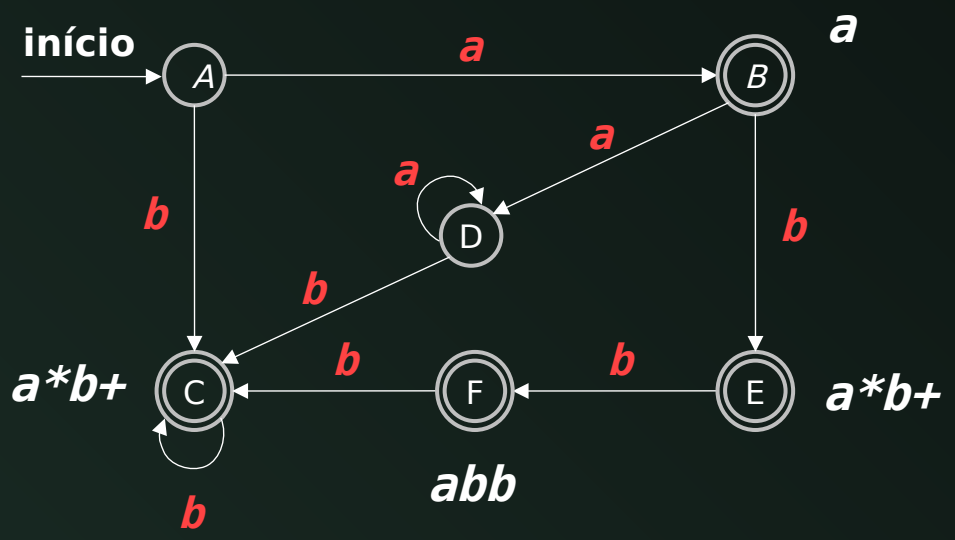
```
a      { ação A1 para padrão p1 }
abb    { ação A2 para padrão p2 }
a*b+   { ação A3 para padrão p3 }
```

Simulação do DFA

- A simulação do DFA segue os passos:
  - O DFA é percorrido até um ponto em que não exista estado seguinte
  - Recua-se na sequência de estados até encontrar um estado final
  - Realiza-se a ação associada ao padrão desse estado

Ex.: *abba*

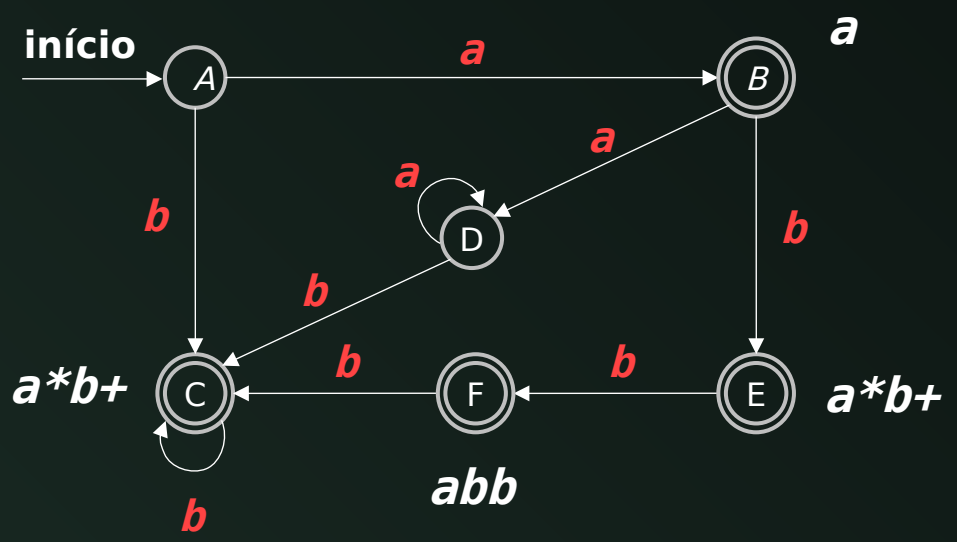




Minimização do DFA

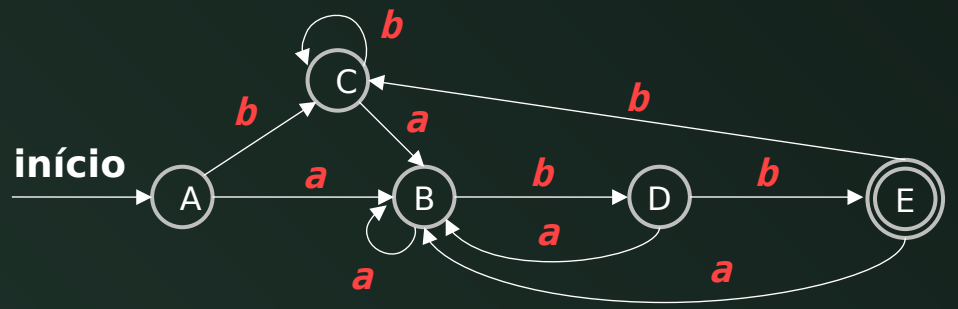
- Quanto menor o número de estados no DFA, melhor
  - DFA é armazenado em uma tabela
  - Cada estado requer uma entrada para cada símbolo do alfabeto

Estado	*a*	*b*
A	B	C
B	D	E
C	-	C
D	D	C
E	-	F
F	-	C



Minimização do DFA

- Sempre existe um único DFA com número mínimo de estados
- A minimização utiliza o conceito de estados distinguíveis
  - O estado *s* é distinguível de *t* se houver alguma cadeia que os diferencie:
    - Uma cadeia que leve apenas um deles a um estado final
      - A cadeia vazia distingue os estados finais dos não-finais
      - A cadeia ***bb*** dintingue o estado A de B

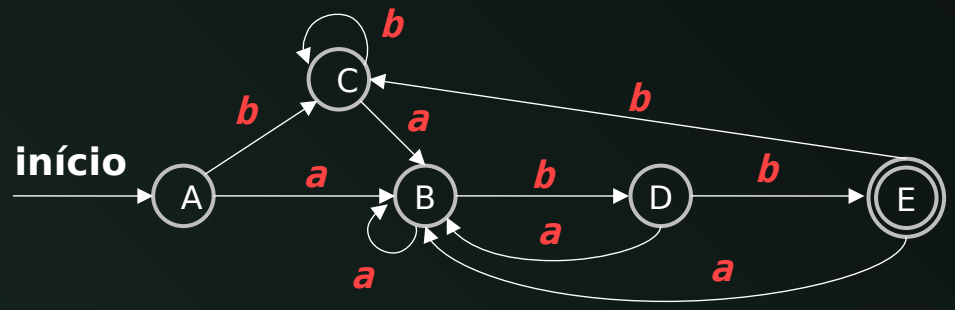


DFA para $(a|b)*abb$

Minimização do DFA

- O algoritmo particiona os estados do DFA em grupos
  - Inicialmente tem-se dois grupos
    - Estados finais
    - Estados não-finais

    Π = {A,B,C,D}{E}



  - Cada grupo é testado sobre cada símbolo *a* do alfabeto
    - Se os estados alcançados estiverem em dois ou mais grupos, dividimos o grupo em subgrupos de forma que:
      - s e t estejam no mesmo subgrupo se, e somente se, a transição para o símbolo *a* os levar para estados dentro do mesmo grupo

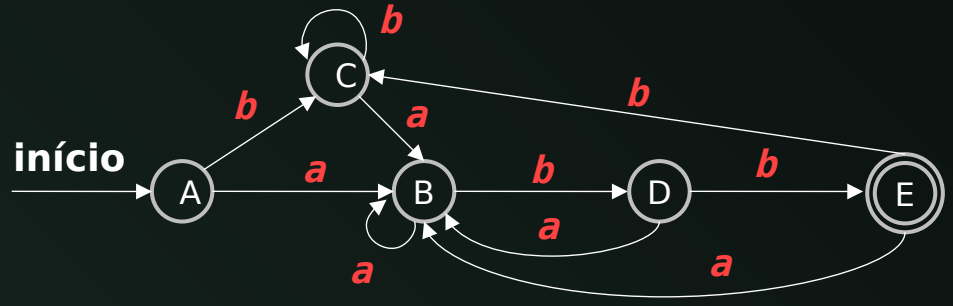
Minimização do DFA

- Considerando a partição

  Π = {A,B,C,D}{E}

  - Analisando o grupo {A,B,C,D}
    - O símbolo ***a*** leva todos os estados para B, um membro do grupo {A,B,C,D}
    - O símbolo ***b***:
      - Leva A, B e C para membros do grupo {A,B,C,D}
      - Leva D para o estado E, um membro de outro grupo

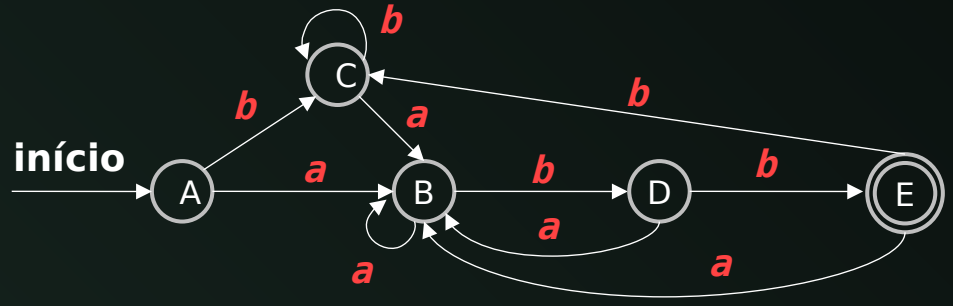
      Π_{nova} = {A,B,C}{D}{E}



Minimização do DFA

- Considerando a partição

  Π = {A,B,C}{D}{E}



  - Analisando o grupo {A,B,C}
    - O símbolo ***a*** leva todos os estados para B, um membro do grupo {A,B,C}
    - O símbolo ***b***:
      - Leva A e C para membros do grupo {A,B,C}
      - Leva B para o estado D, um membro de outro grupo

Π_{nova} = {A,C}{B}{D}{E}

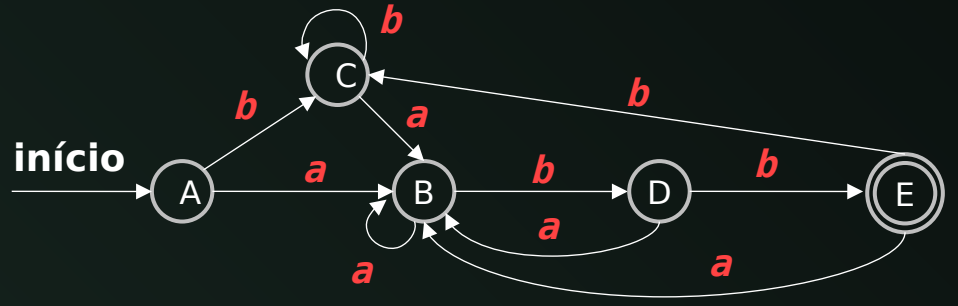
Minimização do DFA

- Considerando a partição

  Π = {A,C}{B}{D}{E}

  - Analisando o grupo {A,C}
    - O símbolo ***a***:
      - Leva A e C para membros do grupo {B}
    - O símbolo ***b***:
      - Leva A e C para membros do grupo {A,C}
    - Como não houve mudança na nova partição, então

      $\Pi_{nova} = \Pi_{final}$ = {A,C}{B}{D}{E}

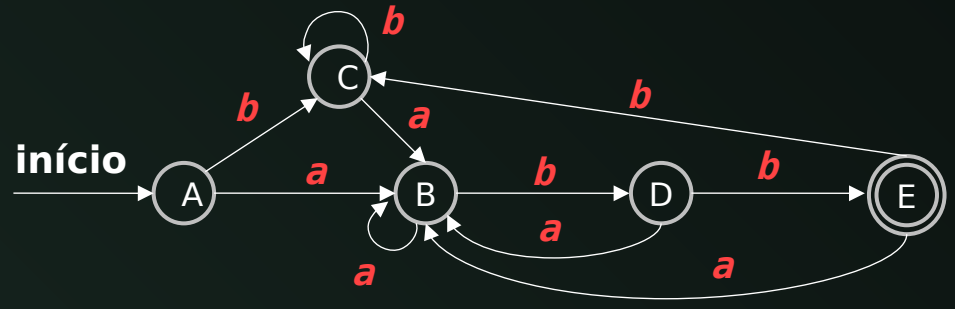


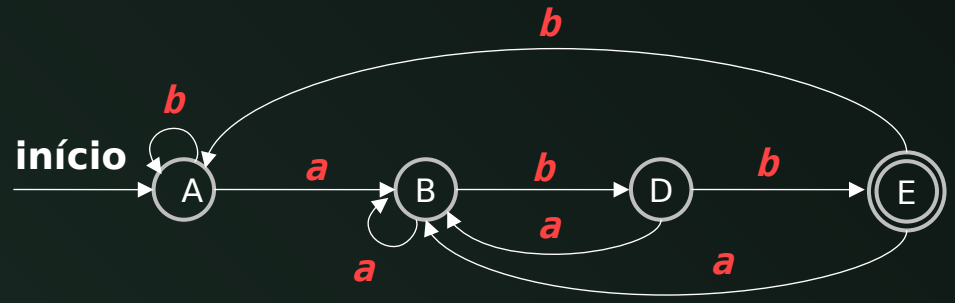
Minimização do DFA

- Construindo o DFA com número mínimo de estados
  - Seleciona-se um representante para cada grupo
  - Monta-se a tabela de transições

Π_{final} = {A,C}{B}{D}{E}

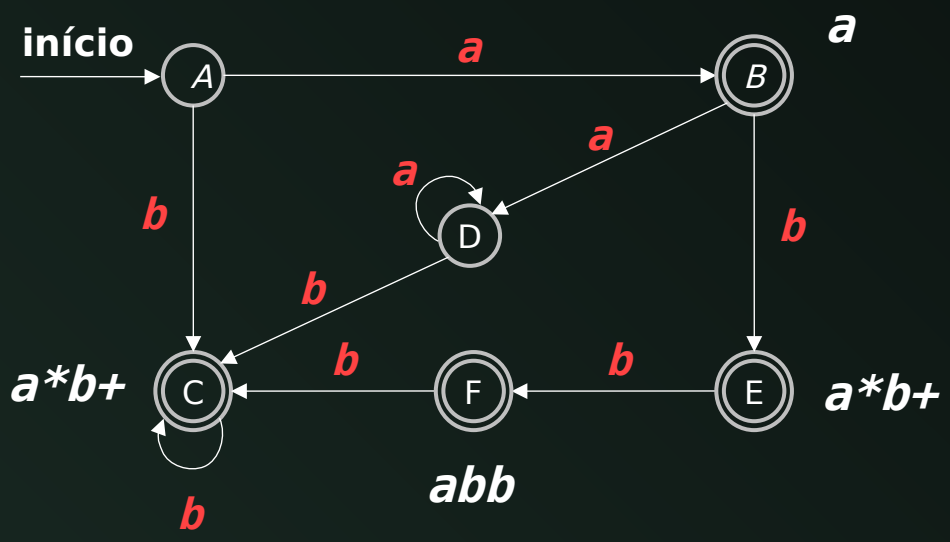
Estado	*a*	*b*
A	B	A
B	B	D
D	B	E
E	B	A





Minimização em Analisadores

- Para minimizar o DFA quando este possui padrões associados, uma otimização pode ser utilizada
  - A partição inicial contém:
    - Grupos para os estados que reconhecem um token em particular
      Ex.: {B}{C,E}{F}
    - Um grupo para os estados que não reconhecem tokens
      Ex.: {A,D}

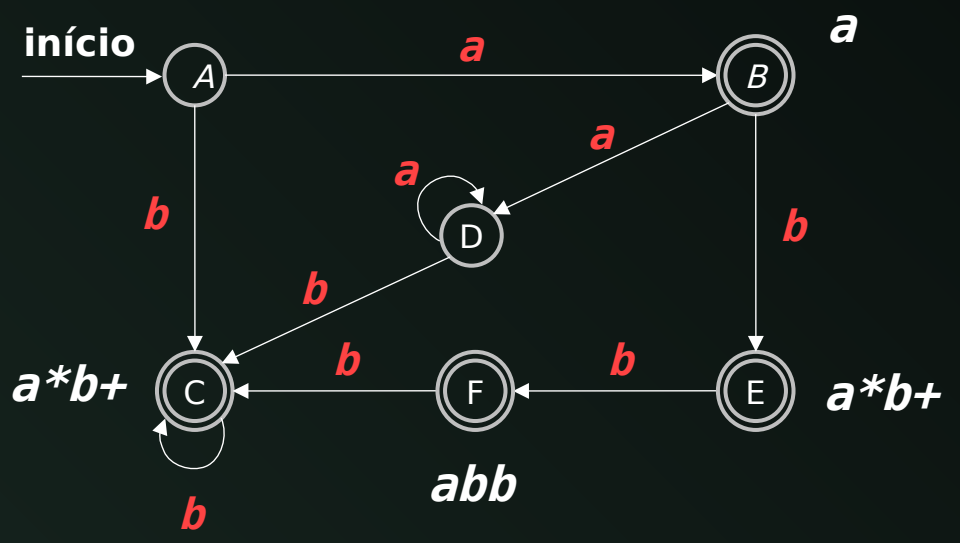


Minimização em Analisadores

- Considerando a partição

  Π = {A,D}{B}{C,E}{F}

  - Analisando o grupo {A,D}
    - O símbolo ***a***:
      - Leva A para membros do grupo {B}
      - Leva D para membros do grupo {A,D}
  - Analisando o grupo {C,E}
    - O símbolo ***b***:
      - Leva C para membros do grupo {A,C}
      - Leva E para membros do grupo {F}

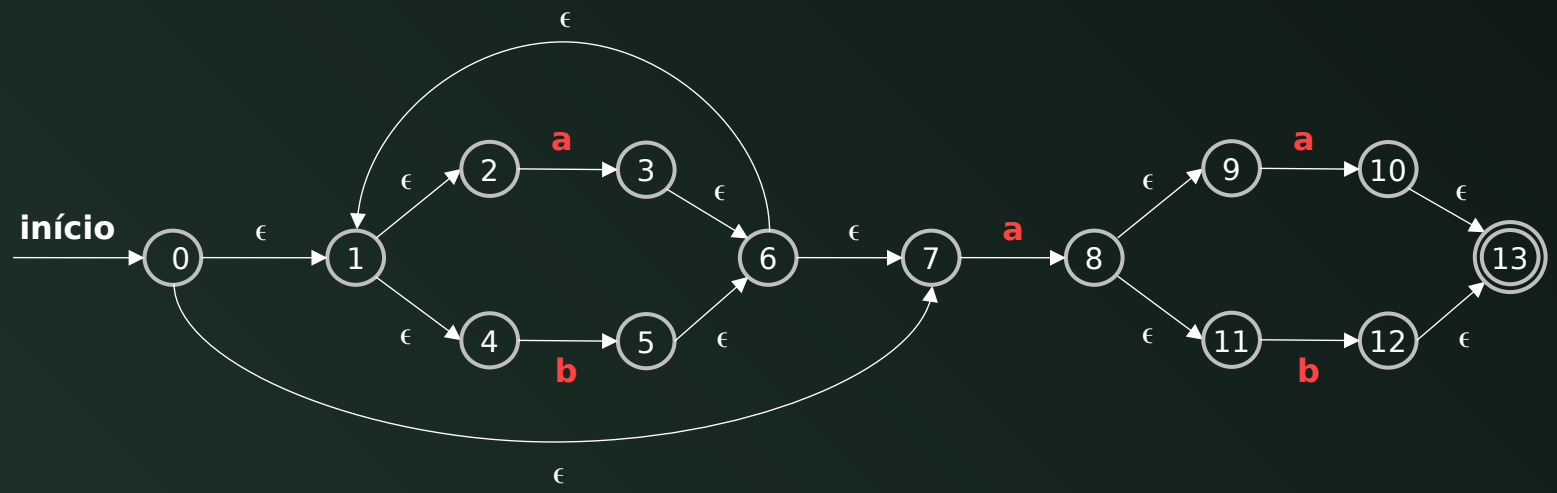


O DFA já é mínimo:

Π_{final} = {A}{B}{C}{D}{E}{F}

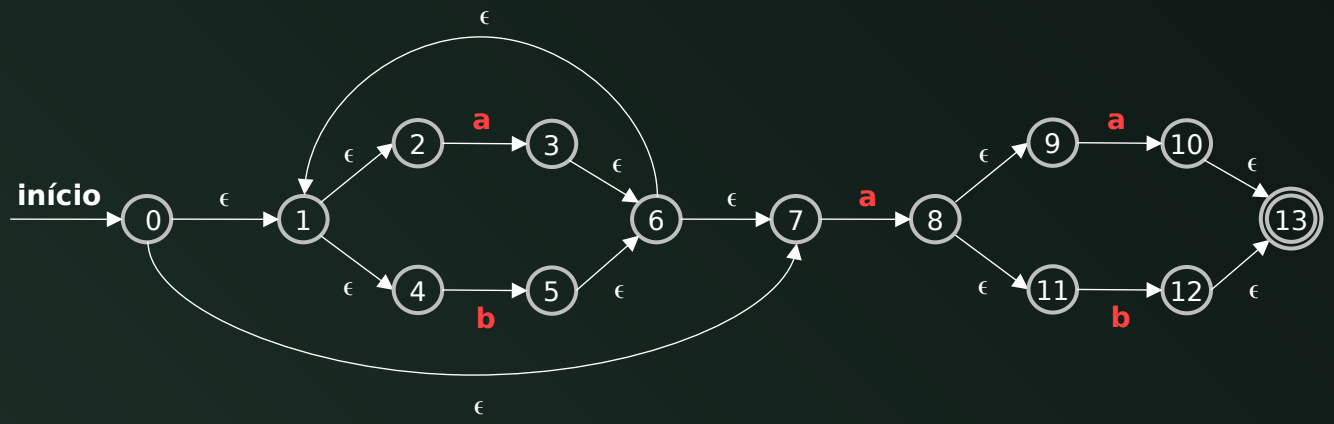
Exercício

1. Construa o DFA com o número mínimo de estados para a expressão regular `(a|b)*a(a|b)`.



NFA para `(a|b)*a(a|b)`

Exercício



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

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

B = {1,2,3,4,6,7,8,9,11}

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

D = {1,2,3,4,6,7,8,9,10,11,13}

E = {1,2,4,5,6,7,12,13}

fecho-ϵ(0) = {0,1,2,4,7} = A

fecho-ϵ(move(A,a)) = fecho-ϵ({3,8}) = {1,2,3,4,6,7,8,9,11} = B

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

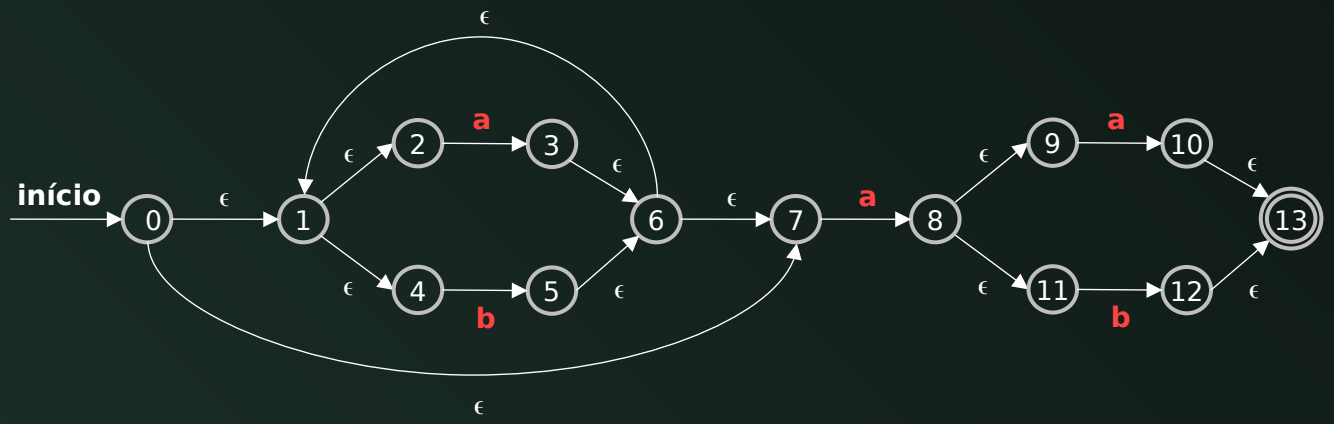
fecho-ϵ(move(B,a)) = fecho-ϵ({3,8,10}) = {1,2,3,4,6,7,8,9,10,11,13} = D

fecho-ϵ(move(B,b)) = fecho-ϵ({5,12}) = {1,2,4,5,6,7,12,13} = E

D_{tran}

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

Exercício



NFA para (a|b)*a(a|b)

fecho-ϵ(move(C,a)) = fecho-ϵ({3,8}) = B

fecho-ϵ(move(C,b)) = fecho-ϵ({5}) = C

fecho-ϵ(move(D,a)) = fecho-ϵ({3,8,10}) = D

fecho-ϵ(move(D,b)) = fecho-ϵ({5,12}) = E

fecho-ϵ(move(E,a)) = fecho-ϵ({3,8}) = B

fecho-ϵ(move(E,b)) = fecho-ϵ({5}) = C

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

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

B = {1,2,3,4,6,7,8,9,11}

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

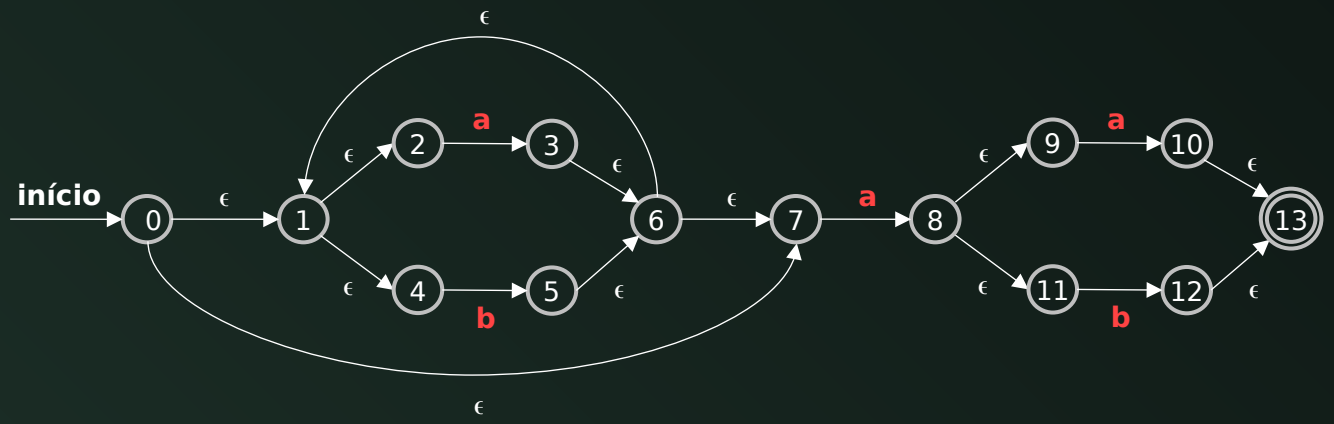
D = {1,2,3,4,6,7,8,9,10,11,13}

E = {1,2,4,5,6,7,12,13}

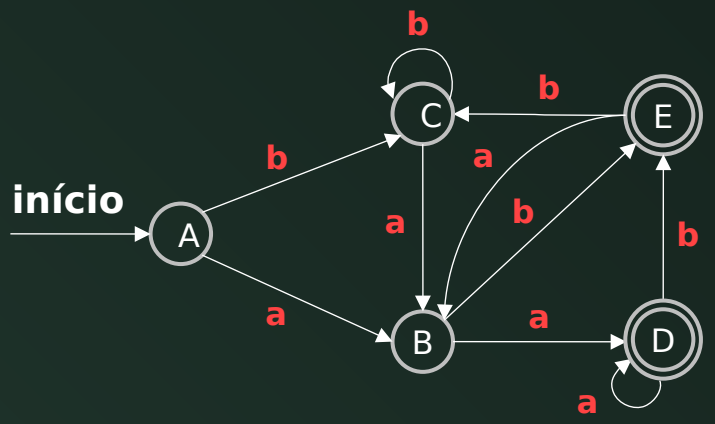
D_{tran}

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

Exercício



NFA para (a|b)*a(a|b)



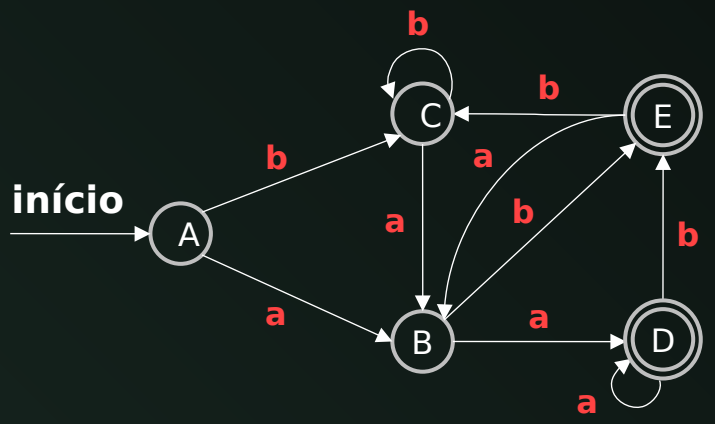
DFA para (a|b)*a(a|b)

D_{tran}		
Estado	Símb.	Próx.
A	a	B
A	b	C
B	a	D
B	b	E
C	a	B
C	b	C
D	a	D
D	b	E
E	a	B
E	b	C

Exercício

- Considerando a partição

  Π = {A,B,C}{D,E}



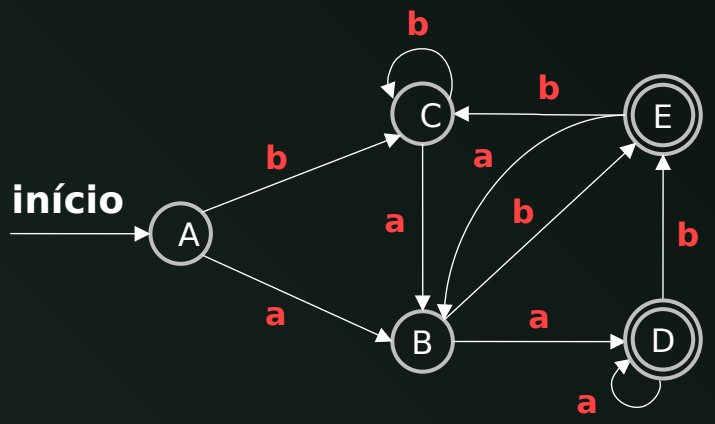
  - Analisando o grupo {A,B,C}
    - O símbolo ***a*** :
      - Leva A e C para B, membro do grupo {A,B,C}
      - Leva B para D, membro do grupo {D,E}

      Π_{nova} = {A,C}{B}{D,E}

Exercício

- Considerando a partição

  Π = {A,C}{B}{D,E}



  - Analisando o grupo {D,E}
    - O símbolo ***a*** :
      - Leva D para D, membro do grupo {D,E}
      - Leva E para B, membro do grupo {B}

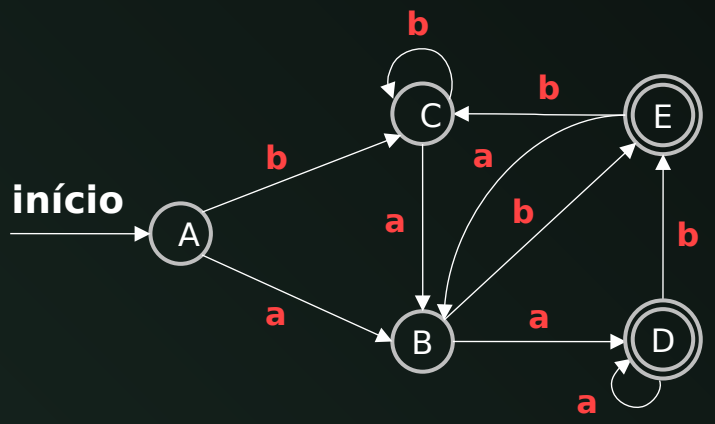
      Π_{nova} = {A,C}{B}{D}{E}

Exercício

- Considerando a partição

  Π = {A,C}{B}{D}{E}

  - Analisando o grupo {A,C}
    - O símbolo ***a*** :
      - Leva A e C para B, membro do grupo {B}
    - O símbolo ***b*** :
      - Leva A e C para C, membro do grupo {A,C}

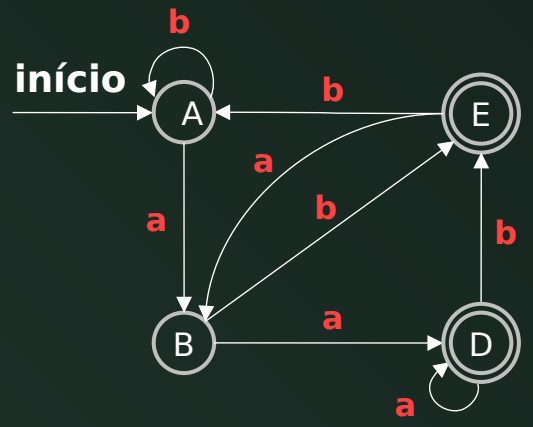


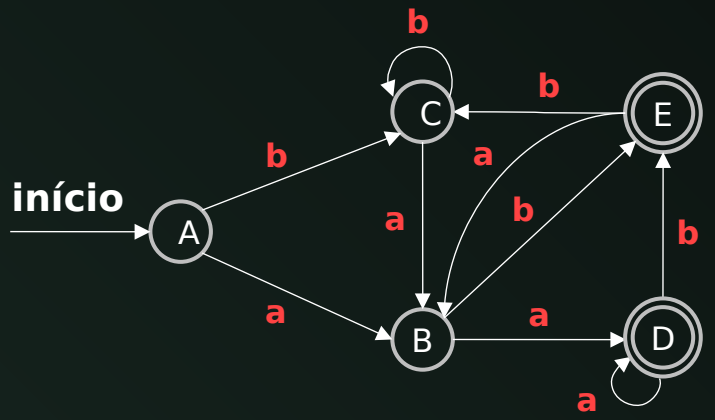
Π_{final} = {A,C}{B}{D}{E}

Exercício

- Considerando a partição

Π_{final} = {A,C}{B}{D}{E}





DFA
Mínimo para
(a|b)*a(a|
b)

Resumo

- Vimos como é construído um gerador de analisador léxico
  - A entrada é lida para um buffer duplo
  - Cada expressão regular é convertida em um NFA
  - Os NFAs são combinados com transições-ϵ em um único NFA
  - O NFA combinado é convertido em um DFA
  - É encontrado o DFA mínimo
  - Uma tabela de transições é armazenada
  - DFA mínimo é simulado sobre a entrada
    - Ao atingir um estado morto, recua-se até o último estado final
    - As ações são executadas