Judson Santos Santiago

Exercícios e Revisão

Compiladores

Exercício

1. Utilize expressões regulares para reconhecer:

   a) Uma constante string na linguagem C++

```
"Oi Mundo"
"Som\a Som\a Testando..."
"\tUm\n\t\"Hacker\"\n\tEsteve aqui!\n"
```

   b) Um comentário iniciando com //

```
// comentários
// na linguagem C++
// se estendem até o fim da linha
```

Solução

```
%{
// strings e comentários em C++
#include <iostream>
using std::cout;
%}

%option noyywrap

%%
[ \t\n]+                ; // ignora espaços em branco
\"(\\.|[^\"\\])*\"      cout << "string\n";
"//".*$                 cout << "comentário\n";
.+                      cout << "outro\n";
%%
```

Exercício

2. Crie um diagrama de transição para um analisador léxico que precisa reconhecer os operadores lógicos e os operadores bit a bit da linguagem C/C++.

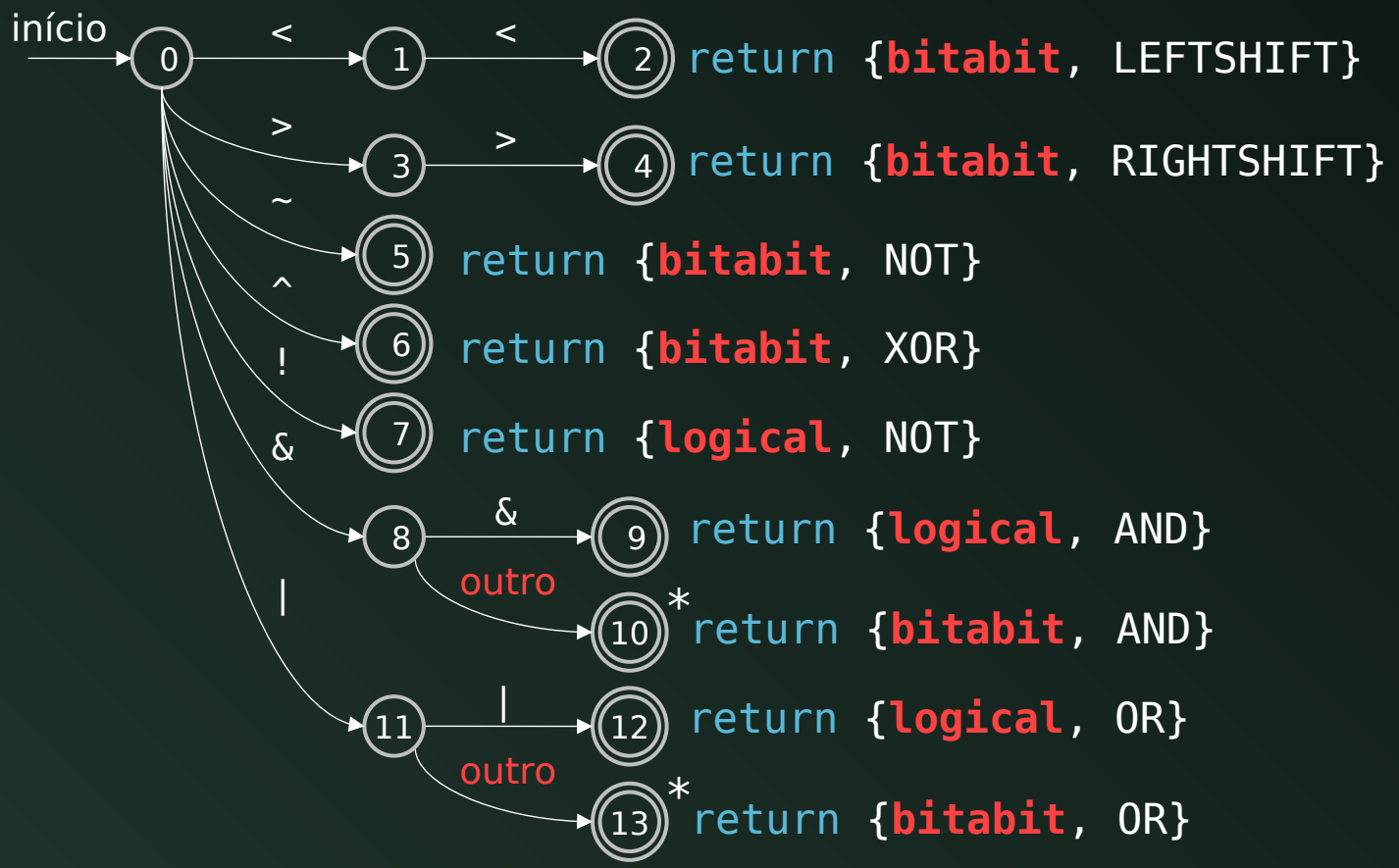
Lógicas

```
expr1 && expr2
expr1 || expr2
     !expr
```

Bit a Bit

```
     ~ expr
expr1 << expr2
expr1 >> expr2
 expr1 & expr2
 expr1 | expr2
 expr1 ^ expr2
```

Solução



Lógicas

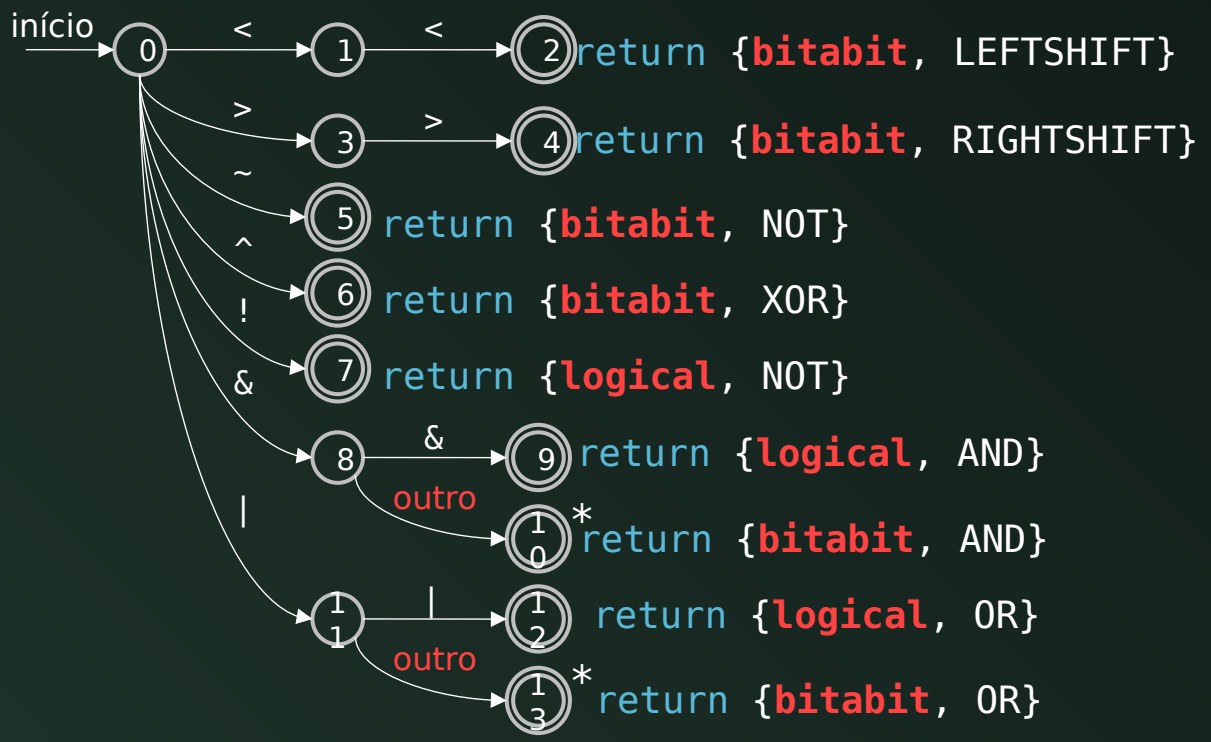
```
expr1 && expr2
expr1 || expr2
     !expr
```

Bit a Bit

```
     ~ expr
expr1 << expr2
expr1 >> expr2
 expr1 & expr2
 expr1 | expr2
 expr1 ^ expr2
```

Exercício

3. Transforme o diagrama obtido na questão anterior em código.



Solução

```
switch (peek)
    {
    case '<':
        peek = cin.get();
        if (peek == '<')
        {
            cout << "<bit,LEFT>
";
            peek = cin.get();
            return BIT;
        }
        break;
    case '>':
        peek = cin.get();
        if (peek == '>')
        {
            cout << "<bit,RIGHT>
 ";
            peek = cin.get();
            return BIT;
        }
        break;
```

```
case '~':
    cout << "<bit,NOT>
    peek = cin.get();
    return BIT;
case '^':
    cout << "<bit,XOR>
    peek = cin.get();
    return BIT;
case '!':
    cout << "<log,NOT>
    peek = cin.get();
    return LOG;
```

```
case '&':
    peek = cin.get();
    if (peek == '&')
    {
        cout << "<log,AND>
        peek = cin.get();
        return LOG;
    }
    else
    {
        cout << "<bit,AND>
        return BIT;
    }
```

```
case '|':
    peek = cin.get();
    if (peek == '|')
    {
        cout << "<log,OR>
        peek = cin.get();
        return LOG;
    }
    else
    {
        cout << "<bit,OR>
        return BIT;
    }
}
```

Exercício

4. Crie um programa `Flex` para reconhecer os operadores lógicos e os operadores bit a bit da linguagem C/C++.

Lógicas

```
expr1 && expr2
expr1 || expr2
     !expr
```

Bit a Bit

```
     ~ expr
expr1 << expr2
expr1 >> expr2
 expr1 & expr2
 expr1 | expr2
 expr1 ^ expr2
```

Solução

```
%%
"<<"            cout << "<bit,LEFT> ";
">>"            cout << "<bit,RIGHT> ";
"~"             cout << "<bit,NOT> ";
"^"             cout << "<bit,XOR> ";
"&"             cout << "<bit,AND> ";
"|"             cout << "<bit,OR> ";
"!"             cout << "<log,NOT> ";
"&&"            cout << "<log,AND> ";
"||"            cout << "<log,OR> ";

{white}         ;
{num}           cout << "<num> ";
{id}            cout << "<id> ";
.               cout << "<" << YYText() << ">
";
%%
```

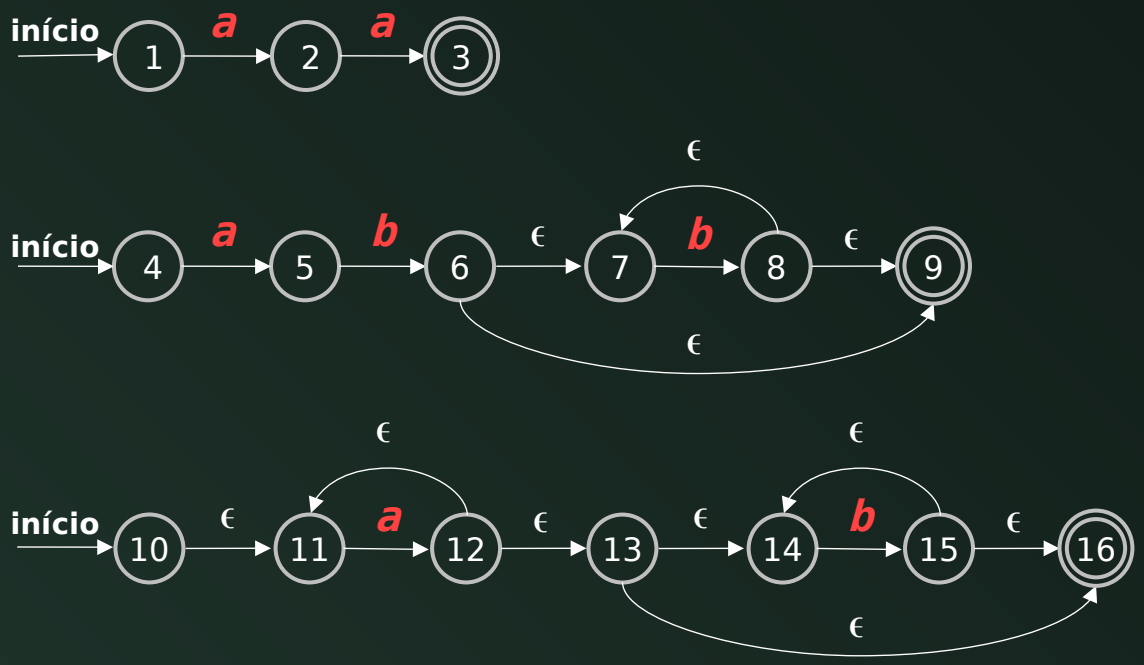
Exercício

5. Considerando o trecho de programa Flex abaixo, crie um DFA mínimo que possa ser usado para reconhecer os tokens descritos pelas expressões regulares.

```
aa     { return PRIMEIRO; }
abb*   { return SEGUNDO;  }
a+b*   { return TERCEIRO; }
```

Solução

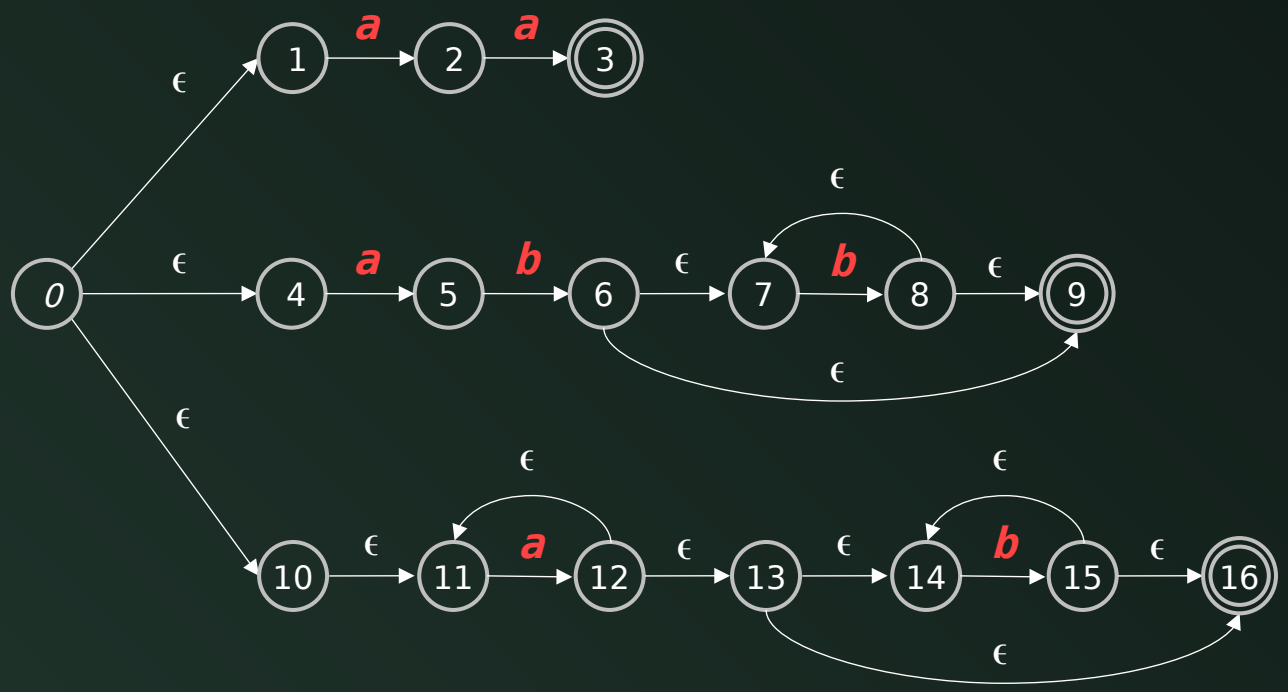
- Montando um NFA para cada padrão



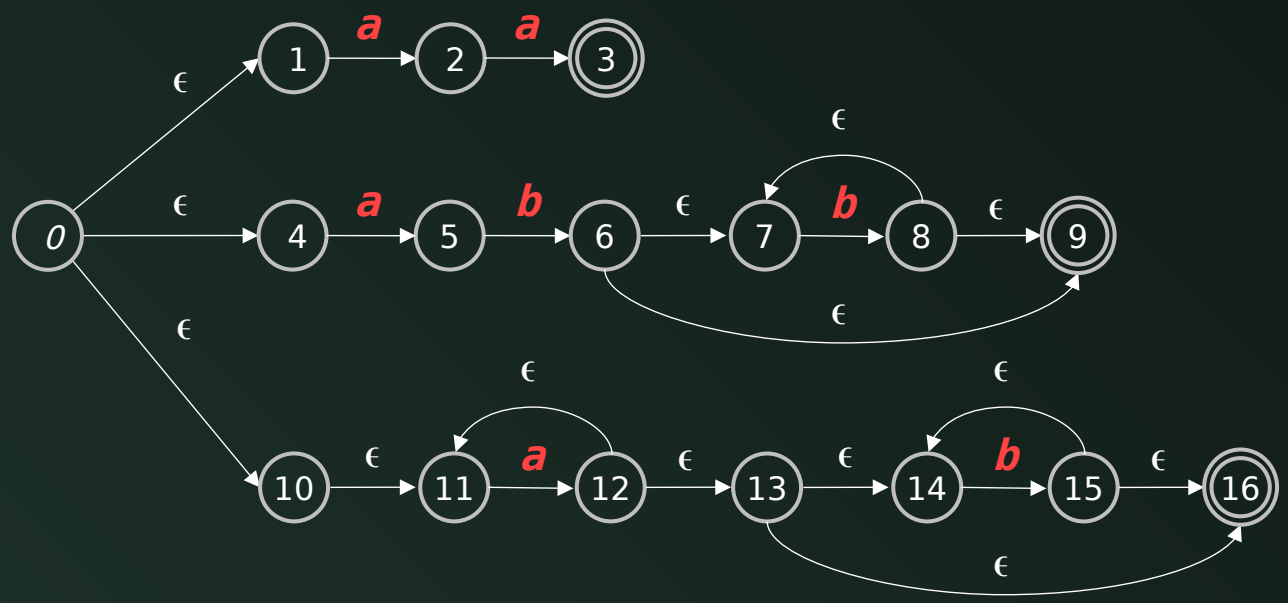

```
aa     { return
PRIMEIRO; }
abb*   { return
SEGUNDO;  }
a+b*   { return
TERCEIRO; }
```

Solução

- Combinando os NFAs em um só



Solução



D_{states} = { A,B }

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

D_{tran}

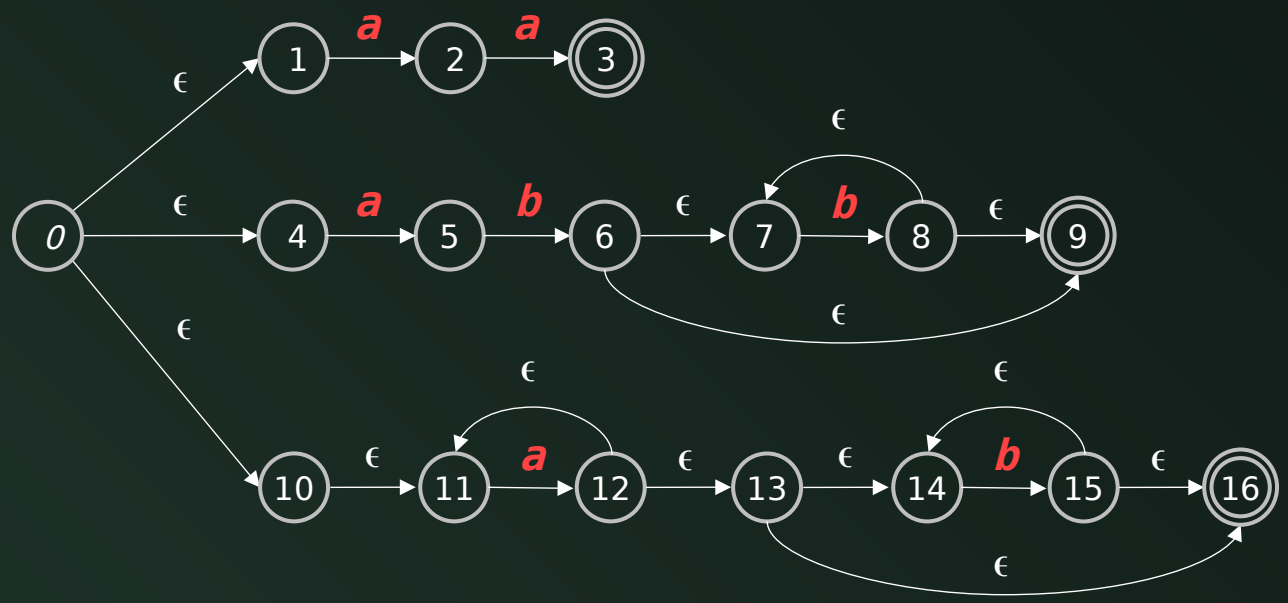
Estado	Símb.	Próx.
A	a	B

fecho-ϵ(0) = {0,1,4,10,11} = A

fecho-ϵ(move(A,a)) = fecho-ϵ({2,5,12}) = {2,5,11,12,13,14,16} = B

fecho-ϵ(move(A,b)) = fecho-ϵ({}) = {}

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

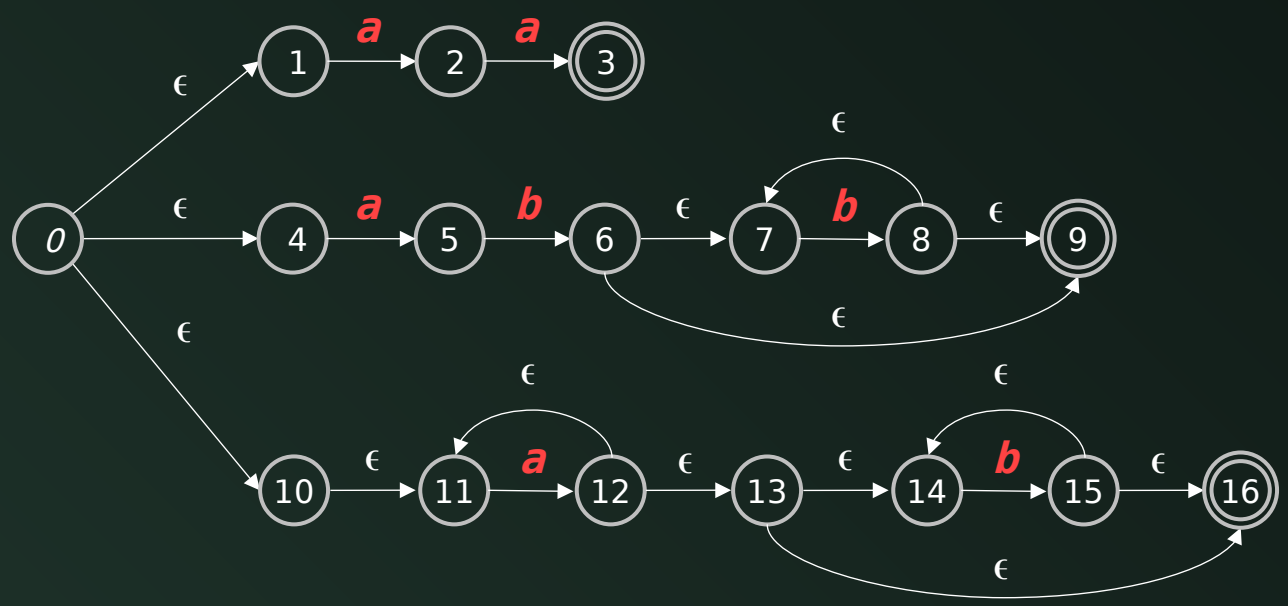
D_{tran}

Estado	Símb.	Próx.
A	a	B
B	a	C
B	b	D

fecho-ϵ(move(B,a)) = fecho-ϵ({3,12}) = {3,11,12,13,14,16} = C

fecho-ϵ(move(B,b)) = fecho-ϵ({6,15}) = {6,7,9,14,15,16} = D

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

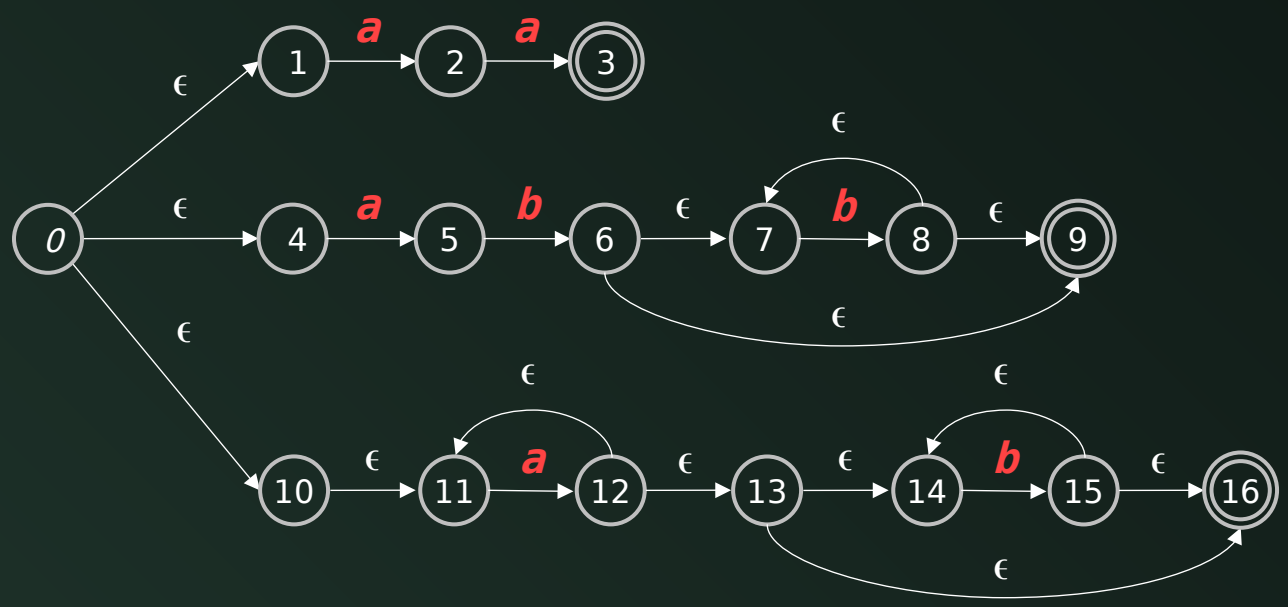
fecho-ϵ(move(C,a)) = fecho-ϵ({12}) = {11,12,13,14,16} = E

fecho-ϵ(move(C,b)) = fecho-ϵ({15}) = {14,15,16} = F

D_{tran}

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

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

fecho-ϵ(move(D,**a**)) = fecho-ϵ({}) = {}

fecho-ϵ(move(D,**b**)) = fecho-ϵ({8,15}) = {7,8,9,14,15,16} = G

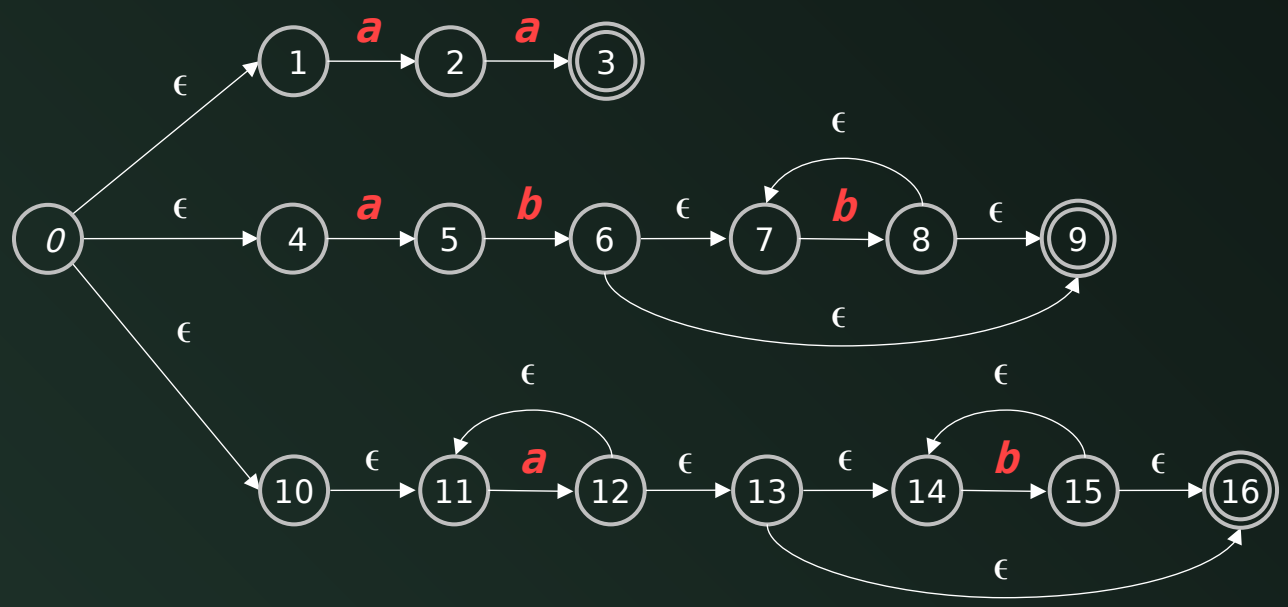
D_{tran}

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

D_{tran}

Estado	Símb.	Próx.
D	b	G

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

fecho-ϵ(move(E,**a**)) = fecho-ϵ({12}) = E

fecho-ϵ(move(E,**b**)) = fecho-ϵ({15}) = F

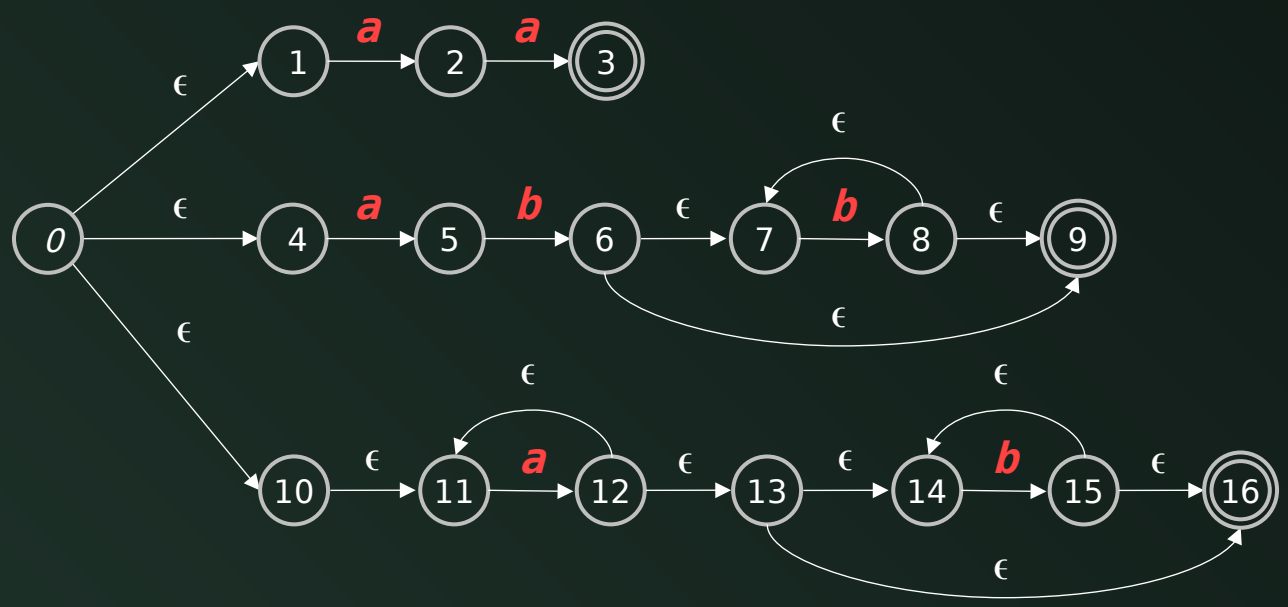
D_{tran}

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

D_{tran}

Estado	Símb.	Próx.
D	b	G
E	a	E
E	b	F

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

fecho-ϵ(move(F,**a**)) = fecho-ϵ({}) = {}

fecho-ϵ(move(F,**b**)) = fecho-ϵ({15}) = F

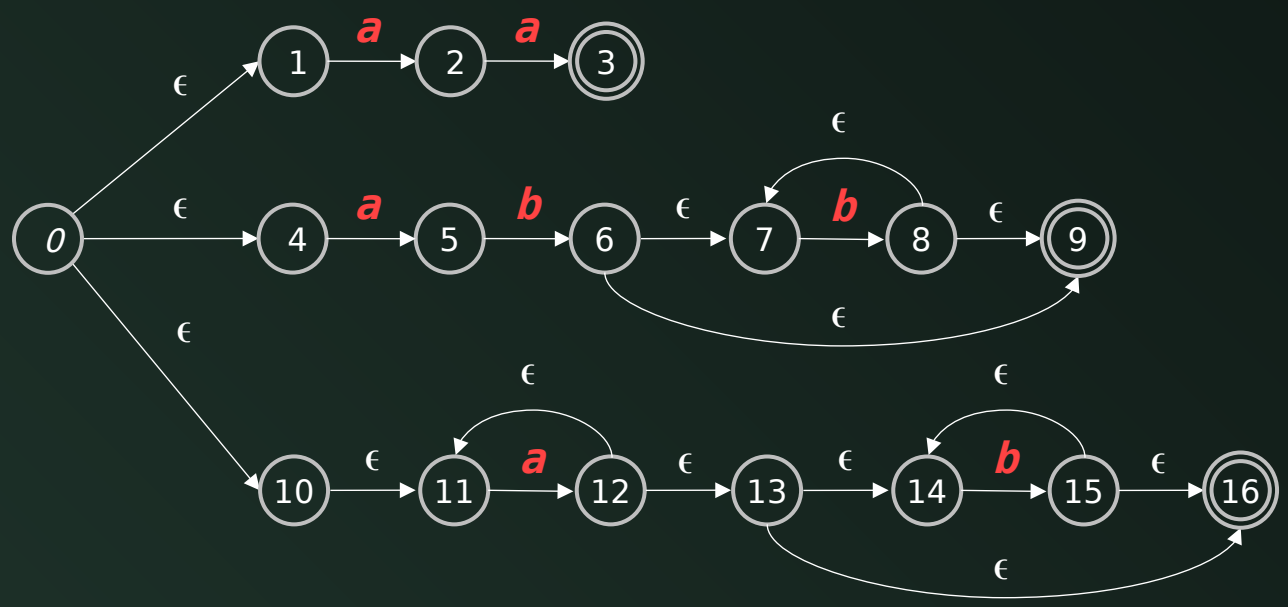
D_{tran}

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

D_{tran}

Estado	Símb.	Próx.
D	b	G
E	a	E
E	b	F
F	b	F

Solução



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

fecho-ϵ(move(G,**a**)) = fecho-ϵ({}) = {}

fecho-ϵ(move(G,**b**)) = fecho-ϵ({8,15}) = G

D_{tran}

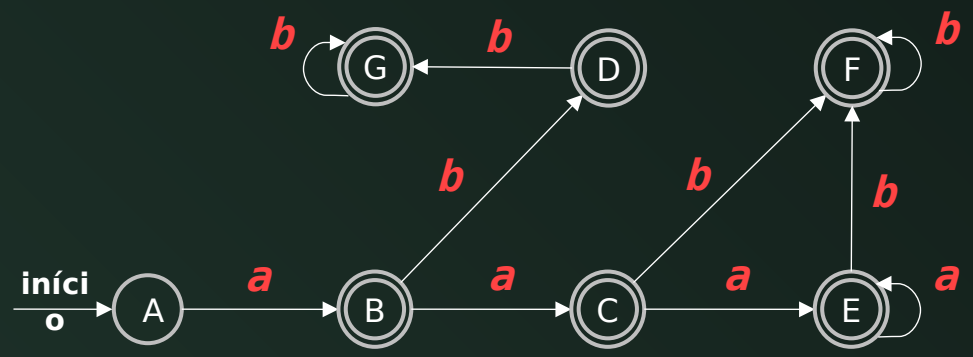
Estado	Símb.	Próx.
A	a	B
B	a	C
B	b	D
C	a	E
C	b	F

D_{tran}

Estado	Símb.	Próx.
D	b	G
E	a	E
E	b	F
F	b	F
G	b	G

Solução

- Montando o DFA



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

D_{tran}

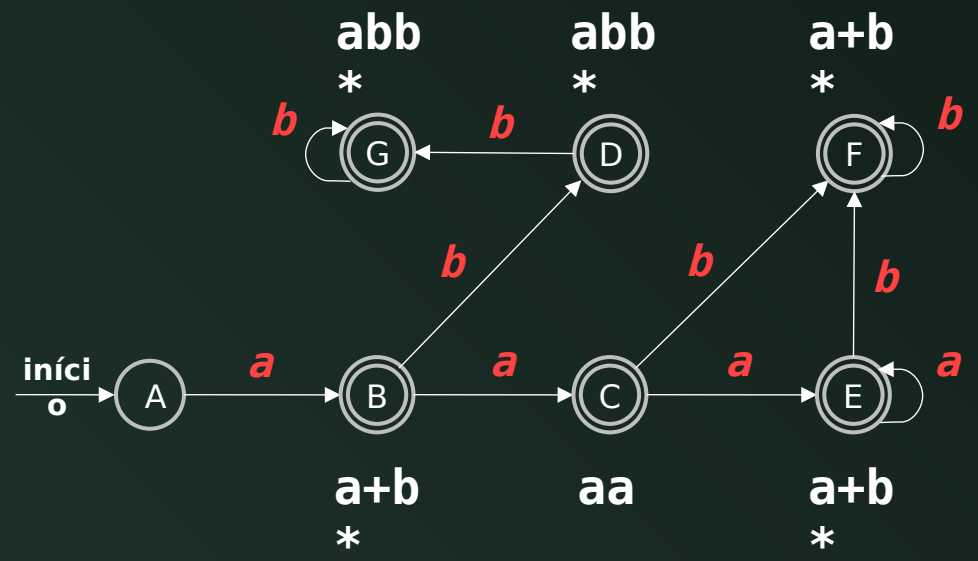
Estado	Símb.	Próx.
A	a	B
B	a	C
B	b	D
C	a	E
C	b	F

D_{tran}

Estado	Símb.	Próx.
D	b	G
E	a	E
E	b	F
F	b	F
G	b	G

Solução

- Atribuindo padrões



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

A = {0,1,4,10,11}

B = {2,5,11,12,13,14,16}

C = {3,11,12,13,14,16}

D = {6,7,9,14,15,16}

E = {11,12,13,14,16}

F = {14,15,16}

G = {7,8,9,14,15,16}

```
3   aa     { return
9   PRIMEIRO; }
16  abb*   { return
        SEGUNDO;  }
        a+b*   { return
```

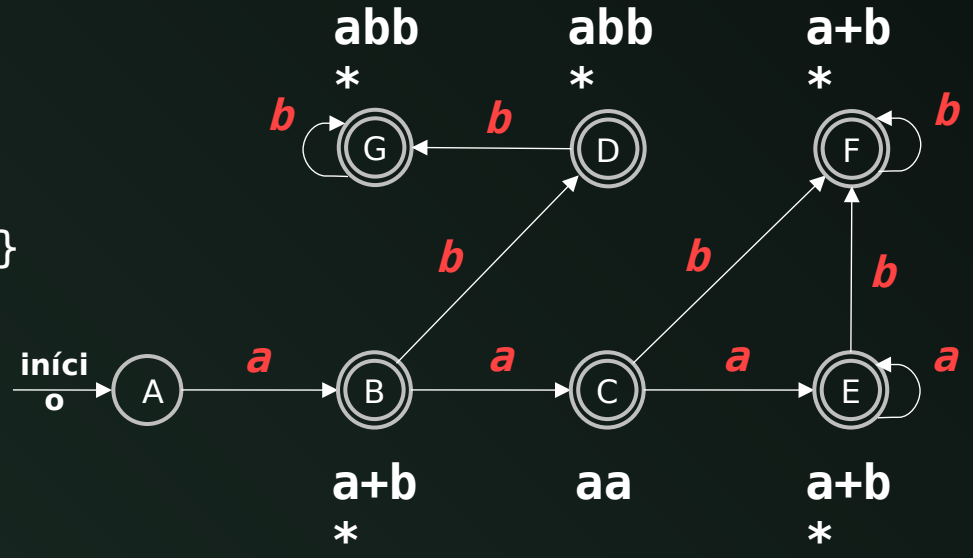
Solução

- Considerando a partição

  Π = {A}{B,E,F}{C}{D,G}

  - Analisando o grupo {B,E,F}
    - O símbolo ***a*** :
      - Leva B para C, membro do grupo {C}
      - Leva E para E, membro do grupo {B,E,F}
      - Leva F para o estado morto

Π_{nova} = {A}{B}{C}{E}{F}{D,G}



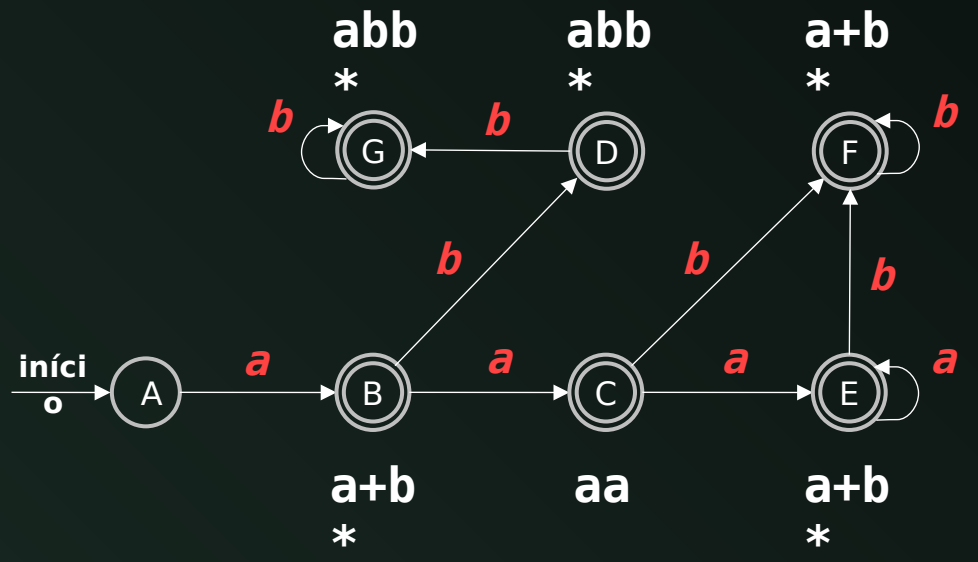
Solução

- Considerando a partição

  Π = {A}{B}{C}{E}{F}{D,G}

  - Analisando o grupo {D,G}
    - O símbolo ***a*** :
      - Leva D e G para o estado morto
    - O símbolo ***b*** :
      - Leva D e G para G

Π_{final} = {A}{B}{C}{E}{F}{D,G}

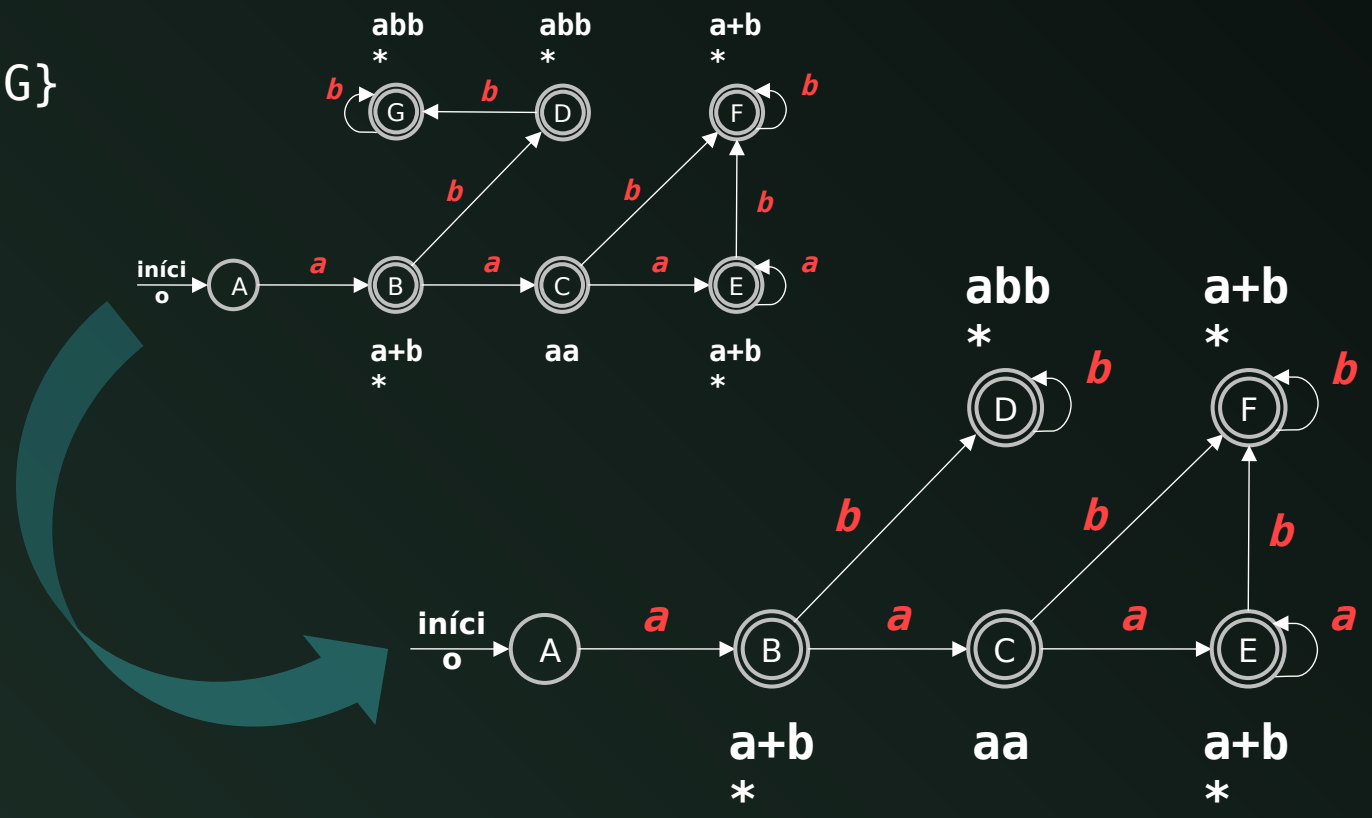


Solução

- Montando tabela do DFA mínimo

Π_{final} = {A}{B}{C}{E}{F}{D,G}

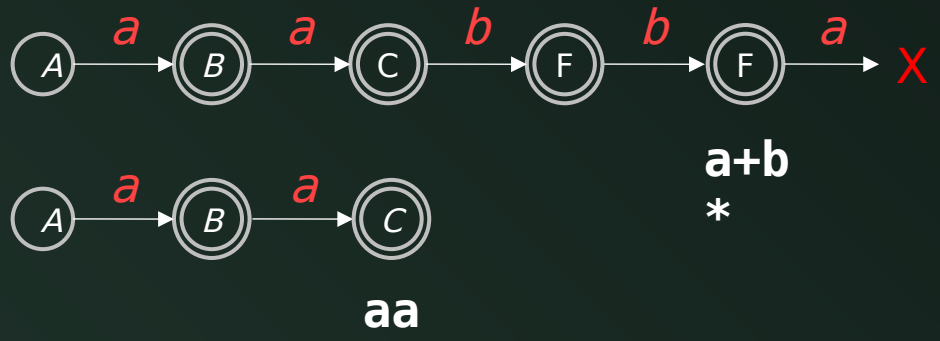
Estado	*a*	*b*
A	B	-
B	C	D
C	E	F
D	-	D
E	E	F
F	-	F



Solução

- Simulação do DFA

Ex.: *aabbaa*




```
aa     { return
PRIMEIRO; }
abb*   { return
SEGUNDO;  }
a+b*   { return
TERCEIRO; }
```

Resumo

- Os geradores de analisadores léxicos:
  - Aceleram o desenvolvimento de compiladores
  - Permitem a criação de outras ferramentas
  - `Flex` é um gerador para C/C++

- Existem também geradores de analisadores sintáticos
  - `Bison` é um gerador para a linguagem C/C++
  - Trabalha em sintonia com o `Flex`