Compiladores[1]

Análise Lexical

Capítulo 3 - "Compilers: Principles, Techniques and Tools"

Prof. Alberto Abad

IST - Universidade de Lisboa

2021/2022

[1] Slides adaptados a partir do material do Prof. Pedro T. Monteiro (2017/2018)

Outline

Reconhecimento de elementos da linguagem

Expressões Regulares

Autómatos Finitos

Conversão entre Representações

Reconhecimento multi-string

Objectivo: Processamento da parte regular de uma linguagem

Objectivo: Processamento da parte regular de uma linguagem

Etapas principais:

- Ler a sequência de caracteres do programa fonte
- Agrupar conjuntos de caracteres em lexemas
- Produzir uma sequência de tokens correspondente aos lexemas do programa fonte

Objectivo: Processamento da parte regular de uma linguagem

Etapas principais:

- Ler a sequência de caracteres do programa fonte
- Agrupar conjuntos de caracteres em lexemas
- Produzir uma sequência de tokens correspondente aos lexemas do programa fonte

Etapas adicionais:

- remoção de espaços
- remoção de comentários
- informação de número de linha
- gestão de erros

Token	Informal description	Lexeme(s)
IF	characters **i**, **f**	**if**
ELSE	characters **e**, **l**, **l**, **e**	**else**
COMPARISON	$<$ or $>$ or $<=$ or $>=$ or $==$ or $!=$	$<=$, $!=$
ID	letter followed by letters and digits	**pi**, **score**, **D2**
NUMBER	any numeric constant	**3.1415**, **0**, **6.02e23**

- Token: símbolo abstracto representando uma unidade lexical
  par consistindo num nome e num atributo opcional
- Padrão: representação dos lexemes associados a um token
  - e.g. dígito repetido uma ou mais vezes
- Lexema: sequência de characteres de um programa que faz *match* com o padrão

- Na maioria de linguagens as seguintes classes cobrem a maioria (ou todos) os tokens:
  1. Um token para cada keyword
  2. Um token para cada operador
  3. Um token para representar todos os identificadores
  4. Um token para cada tipo literal (números, strings)
  5. Um token para cada da símbolo de pontuação (parenteses, ponto vírgula, etc.)

  - Alguns tokens podem ter atributos (ex: identificadores e literais)

Análise lexical

Primeira fase do compilador

Análise lexical

Primeira fase do compilador

Gestão de Tokens:

- Como são representados?

- Como são reconhecidos?

Gestão de Tokens:

- Como são representados?
  - Expressões regulares (REs)
- Como são reconhecidos?

Análise lexical

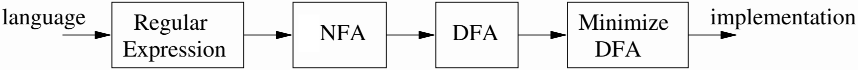
Primeira fase do compilador

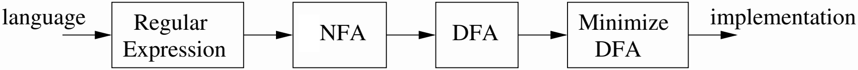
Gestão de Tokens:

- Como são representados?
  - Expressões regulares (REs)
- Como são reconhecidos?
  - Autómatos finitos determinísticos (DFAs)

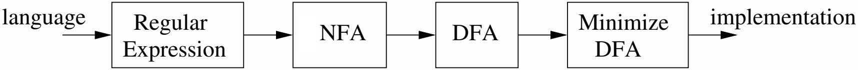
Análise lexical

Tópicos





- Expressões regulares (REs)
- Autómatos finitos (não) determinísticos (NFA/DFA)
- Conversão de expressões regulares para NFA
- Conversão de NFA para DFA
- Minimização de DFAs



- Expressões regulares (REs)
- Autómatos finitos (não) determinísticos (NFA/DFA)
- Conversão de expressões regulares para NFA
- Conversão de NFA para DFA
- Minimização de DFAs

Concretizado na ferramenta Flex

Outline

Reconhecimento de elementos da linguagem

Expressões Regulares

Autómatos Finitos

Conversão entre Representações

Reconhecimento multi-string

Expressões regulares

Exemplos:

- Números inteiros
- Números reais
- Identificadores em C/C++/...
- Matrículas de veículos ligeiros
- Endereços web
- Palavras começadas por 'v' e terminadas por 'ar' em ficheiros de texto
- Strings de 0s e 1s terminadas em 01
- ...

- Um alfabeto Σ é um conjunto de símbolos, não vazio e finito
  - Exemplo: $\Sigma = \{0, 1\}$
- Uma string (ou palavra) é uma sequência finita de símbolos de um alfabeto
  - Exemplo: 011001

- Um alfabeto Σ é um conjunto de símbolos, não vazio e finito
  - Exemplo: $\Sigma = \{0, 1\}$
- Uma string (ou palavra) é uma sequência finita de símbolos de um alfabeto
  - Exemplo: 011001
  - String vazia: ϵ

- Um alfabeto Σ é um conjunto de símbolos, não vazio e finito
  - Exemplo: $\Sigma = \{0, 1\}$
- Uma string (ou palavra) é uma sequência finita de símbolos de um alfabeto
  - Exemplo: 011001
  - String vazia: ϵ
  - Comprimento da string w, $|w|$: número de posições de símbolos na string

- Um alfabeto Σ é um conjunto de símbolos, não vazio e finito
  - Exemplo: $\Sigma = \{0, 1\}$
- Uma string (ou palavra) é uma sequência finita de símbolos de um alfabeto
  - Exemplo: 011001
  - String vazia: ϵ
  - Comprimento da string w, $|w|$: número de posições de símbolos na string
  - Σ^k: conjunto de strings com comprimento k com símbolos de Σ
    - $\Sigma^0 = \{\epsilon\}$
    - $\Sigma = \{0, 1\}$, $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$
  - Σ^*: conjunto de todas as strings sobre um alfabeto Σ
    - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots$
    - $\Sigma^* = \{\epsilon\} \cup \Sigma^+$

- Um alfabeto Σ é um conjunto de símbolos, não vazio e finito
  - Exemplo: $\Sigma = \{0, 1\}$
- Uma string (ou palavra) é uma sequência finita de símbolos de um alfabeto
  - Exemplo: 011001
  - String vazia: ϵ
  - Comprimento da string w, $|w|$: número de posições de símbolos na string
  - Σ^k: conjunto de strings com comprimento k com símbolos de Σ
    - $\Sigma^0 = \{\epsilon\}$
    - $\Sigma = \{0, 1\}$, $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$
  - Σ^*: conjunto de todas as strings sobre um alfabeto Σ
    - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots$
    - $\Sigma^* = \{\epsilon\} \cup \Sigma^+$
  - Se x, y são strings, xy é a concatenação de x e y
    - Cópia de x seguida de cópia de y

- Dado um alfabeto Σ, uma linguagem é um qualquer subconjunto das strings em Σ^*
  - e.g. a linguagem de todas as strings sobre $\{0, 1\}$ que contém a string 01

- Problema: decidir se uma string w pertence a uma linguagem L
  - Seja Σ um alfabeto e seja L uma linguagem sobre Σ
  - Problema L: Dada $w \in \Sigma^*$, decidir se w pertence a L

- **União** de linguagens L e M ($L \cup M$)
  - Conjunto de strings em L, em M ou em ambas
    - $L \cup M = \{s \mid s \in L \text{ or } s \in M\}$

- **União** de linguagens L e M ($L \cup M$)
  - Conjunto de strings em L, em M ou em ambas
    - $L \cup M = \{s \mid s \in L \text{ or } s \in M\}$
- **Concatenação** de linguagens L e M ($L \cdot M$ ou LM)
  - Conjunto de strings formadas pela concatenação de qualquer string de L com qualquer string de M
    - $LM = \{st \mid s \in L \text{ and } t \in M\}$
    - $L^0 = \{\epsilon\}$; $L^2 = \{s_1 s_2 \mid s_1, s_2 \in L\}$; $L^k = \{s_1 \dots s_k \mid s_1, \dots s_k \in L\}$

- **União** de linguagens L e M ($L \cup M$)
  - Conjunto de strings em L, em M ou em ambas
    - $L \cup M = \{s \,|\, s \in L \text{ or } s \in M\}$
- **Concatenação** de linguagens L e M ($L \cdot M$ ou LM)
  - Conjunto de strings formadas pela concatenação de qualquer string de L com qualquer string de M
    - $LM = \{st \,|\, s \in L \text{ and } t \in M\}$
    - $L^0 = \{\epsilon\}$; $L^2 = \{s_1 s_2 \,|\, s_1, s_2 \in L\}$; $L^k = \{s_1 \ldots s_k \,|\, s_1, \ldots s_k \in L\}$
- **Fecho** (ou **fecho de Kleene**) de uma linguagem L (L^*)
  - Conjunto de strings formadas por concatenação de qualquer string de L **0 ou mais vezes**
    - $L^* = \cup_{i=0}^{\infty} L^i$

- **União** de linguagens L e M ($L \cup M$)
  - Conjunto de strings em L, em M ou em ambas
    - $L \cup M = \{s \mid s \in L \text{ or } s \in M\}$
- **Concatenação** de linguagens L e M ($L \cdot M$ ou LM)
  - Conjunto de strings formadas pela concatenação de qualquer string de L com qualquer string de M
    - $LM = \{st \mid s \in L \text{ and } t \in M\}$
    - $L^0 = \{\epsilon\}$; $L^2 = \{s_1 s_2 \mid s_1, s_2 \in L\}$; $L^k = \{s_1 \dots s_k \mid s_1, \dots s_k \in L\}$
- **Fecho** (ou **fecho de Kleene**) de uma linguagem L (L^*)
  - Conjunto de strings formadas por concatenação de qualquer string de L **0 ou mais vezes**
    - $L^* = \cup_{i=0}^{\infty} L^i$
- **Fecho Positivo** de uma linguagem L (L^+)
  - Conjunto de strings formadas por concatenação de qualquer string de L **1 ou mais vezes**
    - $L^+ = \cup_{i=1}^{\infty} L^i$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* = \{\epsilon, cc, ccddd, cccc, dddc, dddcccc, \ldots\}$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* = \{\epsilon, cc, ccddd, cccc, dddc, dddcccc, \ldots\}$
- $L^+ =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* = \{\epsilon, cc, ccddd, cccc, dddc, dddcccc, \ldots\}$
- $L^+ = \{a, abb, aa, aabb, aaabb, bba, \ldots\}$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* = \{\epsilon, cc, ccddd, cccc, dddc, dddcccc, \ldots\}$
- $L^+ = \{a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^+ =$

$L = \{a, bb\}$ and $M = \{cc, ddd\}$

- $L \cup M = \{a, cc, bb, ddd\}$
- $LM = \{acc, addd, bbcc, bbddd\}$
- $L^* = \{\epsilon, a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^* = \{\epsilon, cc, ccddd, cccc, dddc, dddcccc, \ldots\}$
- $L^+ = \{a, abb, aa, aabb, aaabb, bba, \ldots\}$
- $M^+ = \{cc, ccddd, cccc, dddc, dddcccc, \ldots\}$

- **Caso base**:
  - A constante ϵ é uma expressão regular, $L(\epsilon) = \{\epsilon\}$
  - Se a é um símbolo, então **a** é uma expressão regular, $L(\mathbf{a}) = \{a\}$

- **Caso base**:
  - A constante ϵ é uma expressão regular, $L(\epsilon) = \{\epsilon\}$
  - Se a é um símbolo, então **a** é uma expressão regular, $L(\mathbf{a}) = \{a\}$

- **Indução**:
  Se as variáveis R e S são expressões regulares que denotam as lingugens $L(R)$ e $L(S)$, então:
  - $(\mathbf{R})|(\mathbf{S})$ é uma expressão regular, com linguagen $L(R) \cup L(S)$
  - $(\mathbf{R})(\mathbf{S})$ é uma expressão regular, com linguagen $L(R)L(S)$
  - $(\mathbf{R})^*$ é uma expressão regular, com linguagen $(L(R))^*$
  - $(\mathbf{R})$ é uma expressão regular, com linguagen $L(R)$

Os parêntesis permitem alterar a prioridade dos operadores.

Convenções para precedência:

- Fecho ($*$) tem a maior precedência
- Concatenação ($\cdot$) é o próximo na ordem de precedência
- União ($|$) tem a menor precedência

Exemplos:

- $\mathbf{a}^*|\mathbf{b}^*$ corresponde a $(\mathbf{a}^*)|(\mathbf{b}^*)$
- $\mathbf{a}^*\mathbf{c}|\mathbf{b}^*$ corresponde a $(\mathbf{a}^*)(\mathbf{c})|(\mathbf{b}^*)$

Considerando o alfabeto $\Sigma = \{a, b\}$:

- $\mathbf{a}|\mathbf{b}$ representa a linguagem $\{a, b\}$
- $(\mathbf{a}|\mathbf{b})(\mathbf{a}|\mathbf{b})$ representa a linguagem $\{aa, ab, ba, bb\}$
- $\mathbf{a}^*$ representa a linguagem $\{\epsilon, a, aa, aaa, \ldots\}$
- $(\mathbf{a}|\mathbf{b})^*$ representa a linguagem $\{\epsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$
- $\mathbf{a}|\mathbf{a}^*\mathbf{b}$ representa a linguagem $\{a, b, ab, aab, aaab, \ldots\}$

- Operador **?** representa zero ou um de
  - $R?$ corresponde a $\epsilon|R$
- Operador **+** representa um ou mais de
  - R^+ corresponde a RR^*
- Operador **{n}** representa *n* cópias de
- Símbolo **.** representa qualquer caracter
- Sequência $[\mathbf{a_1} \cdots \mathbf{a_k}]$ representa a expressão regular $a_1|\cdots|a_k$
- Intervalos *x*-*y* podem ser utilizados dentro de parêntesis rectos
  - e.g. todos os caracteres de *x* a *y* na sequência ASCII
- Caracteres especiais do UNIX/Linux: precedidos por \

Expressões regulares

Notação adicional

Exemplos de expressões regulares (baseadas no código ASCII):

- Inteiros:
  - $([+\,-]?[1\text{-}9][0\text{-}9]^*)|0$
- Identificadores:
  - $[a\text{-}zA\text{-}Z][a\text{-}zA\text{-}Z0\text{-}9]^*$
  - $[a\text{-}zA\text{-}Z\_][a\text{-}zA\text{-}Z0\text{-}9\_]^*$
- Reais:
  - $[+-]?(([0\text{-}9]^+\backslash.[0\text{-}9]^*)|([0\text{-}9]^*\backslash.[0\text{-}9]^+))$

Exemplos de expressões regulares (baseadas no código ASCII):

- Inteiros:
  - $([+\ \text{-}]?[1\text{-}9][0\text{-}9]^*)|0$
- Identificadores:
  - $[a\text{-}zA\text{-}Z][a\text{-}zA\text{-}Z0\text{-}9]^*$
  - $[a\text{-}zA\text{-}Z\_][a\text{-}zA\text{-}Z0\text{-}9\_]^*$
- Reais:
  - $[+\text{-}]?(([0\text{-}9]^+\backslash.[0\text{-}9]^*)|([0\text{-}9]^*\backslash.[0\text{-}9]^+))$

Nota: notação adicional simplifica escrita de expressões regulares, mas permite representar as mesmas linguagens

Outline

Autómatos finitos

Um autómato finito é essencialmente um grafo que:

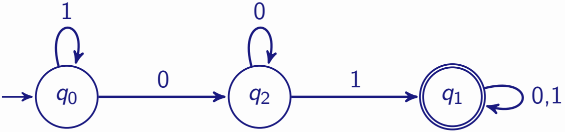
- actua como um reconhecedor de uma linguagem regular
  - responde "sim" ou "não" para uma string de entrada
- pode ser de dois tipos:
  - Autómato finito **determinístico** (DFA), em que para cada estado e símbolo do alfabeto, existe um único arco com esse símbolo partindo desse estado
  - Autómato finito **não determinístico** (NDA), em que não existem restrições nos símbolos dos seus arcos, pudendo existir vários arcos com o mesmo símbolo, assim como o símbolo ϵ, partindo de um mesmo estado

Um autómato finito determinístico (DFA) A é um tuplo $(Q, \Sigma, \delta, q_0, F)$ em que:

- Q é o conjunto finito de estados
- Σ é o conjunto finito de símbolos de entrada
- $\delta : \; Q \times \Sigma \rightarrow Q$ é uma função de transição
- $q_0 \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto de estados finais ou aceitadores

- DFA $A = (Q, \Sigma, \delta, q_0, F)$
- Diagrama de transição:
  - Um nó para cada estado em Q
  - Uma transição com etiqueta a do estado q para o estado p, com $p = \delta(q, a)$, e com $q, p \in Q$ e $a \in \Sigma$
  - Seta para o estado inicial q_0
  - Estados aceitadores (em F) marcados com dois círculos

- DFA $A = (Q, \Sigma, \delta, q_0, F)$
- Diagrama de transição:
  - Um nó para cada estado em Q
  - Uma transição com etiqueta a do estado q para o estado p, com $p = \delta(q, a)$, e com $q, p \in Q$ e $a \in \Sigma$
  - Seta para o estado inicial q_0
  - Estados aceitadores (em F) marcados com dois círculos
- Exemplo:



DFA para reconhecimento de linguagem

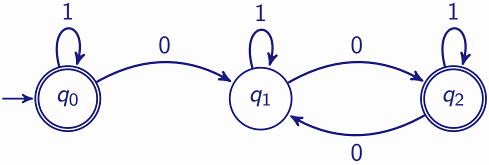
Exemplo 1

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings com um número par de 0's:

DFA para reconhecimento de linguagem

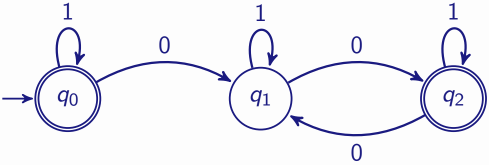
Exemplo 1

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings com um número par de 0's:

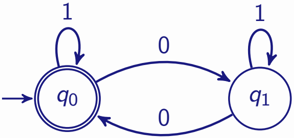


DFA para reconhecimento de linguagem
Exemplo 1

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings com um número par de 0's:



Uma solução com menos estados:



DFA para reconhecimento de linguagem

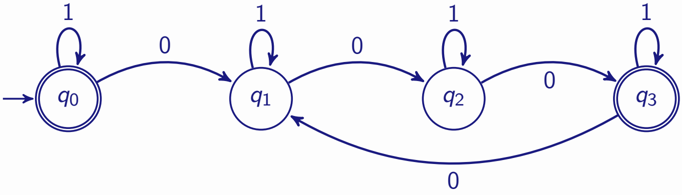
Exemplo 2

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings tal que número de 0's é um múltiplo de 3:

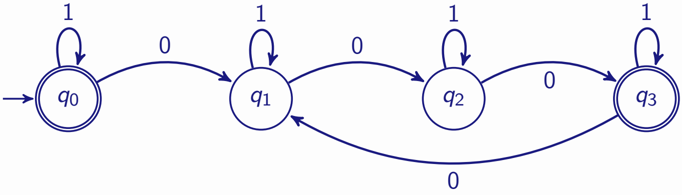
DFA para reconhecimento de linguagem

Exemplo 2

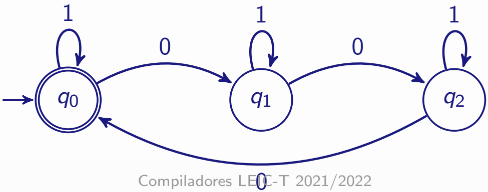
Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings tal que número de 0's é um múltiplo de 3:



Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o DFA para reconhecer strings tal que número de 0's é um múltiplo de 3:



Uma solução com menos estados:



Autómatos finitos **não determinísticos** (NFA)

Definição

Um autómato finito não determinístico (NFA) A é um tuplo $(Q, \Sigma, \delta, q_0, F)$ em que:

- Q é o conjunto finito de estados
- Σ é o conjunto finito de símbolos de entrada
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathbb{P}^Q$ é uma função de transição
  - δ mapeia um estado ou um símbolo de entrada (ou ϵ) num subconjunto de Q
- $q_0 \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto de estados finais ou aceitadores

Um autómato finito não determinístico (NFA) A é um tuplo $(Q, \Sigma, \delta, q_0, F)$ em que:

- Q é o conjunto finito de estados
- Σ é o conjunto finito de símbolos de entrada
- $\delta : \; Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathbb{P}^Q$ é uma função de transição
  - δ mapeia um estado ou um símbolo de entrada (ou ϵ) num subconjunto de Q
- $q_0 \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto de estados finais ou aceitadores

NOTA: Esta definição correponde a uma geralização, também conhecida como NFA com transições ϵ

Autómatos finitos **não determinísticos** (NFA)

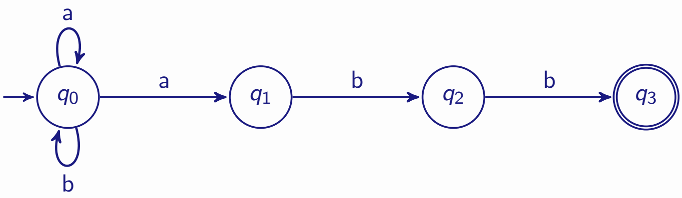
Características diferenciadoras:

- É possível existirem múltiplas mudanças de estado com o mesmo símbolo
- É possível existirem mudanças de estado com o símbolo ϵ

Características diferenciadoras:

- É possível existirem múltiplas mudanças de estado com o mesmo símbolo
- É possível existirem mudanças de estado com o símbolo ϵ

Exemplo - NFA aceitador de strings terminadas com abb:



NFA para reconhecimento de linguagem

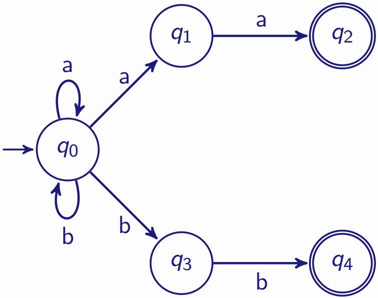
Exemplo 1

Considerando o alfabeto $\Sigma = \{a, b\}$, construa o NFA que aceita strings terminadas com aa ou com bb:

NFA para reconhecimento de linguagem

Exemplo 1

Considerando o alfabeto $\Sigma = \{a, b\}$, construa o NFA que aceita strings terminadas com aa ou com bb:



NFA para reconhecimento de linguagem

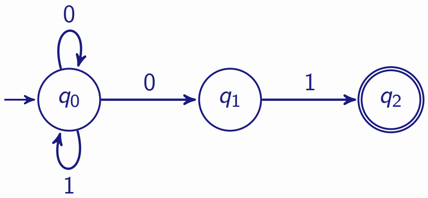
Exemplo 2

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o NFA que aceita strings terminadas com a sequência 01:

NFA para reconhecimento de linguagem

Exemplo 2

Considerando o alfabeto $\Sigma = \{0, 1\}$, construa o NFA que aceita strings terminadas com a sequência 01:



NFA para reconhecimento de linguagem

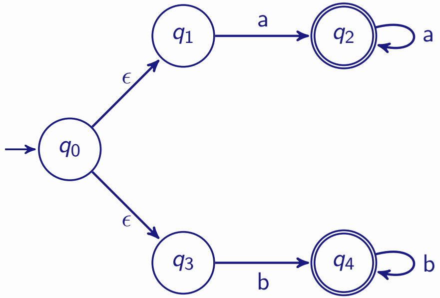
Exemplo 3 (com transições ϵ)

Considerando o alfabeto $\Sigma = \{a, b\}$, construa o NFA que aceita strings formadas por um ou mais símbolo a ou b:

NFA para reconhecimento de linguagem

Exemplo 3 (com transições ϵ)

Considerando o alfabeto $\Sigma = \{a, b\}$, construa o NFA que aceita strings formadas por um ou mais símbolo a ou b:



NFA para reconhecimento de linguagem

Exemplo 4 (números decimais)

NFA que aceita números decimais:

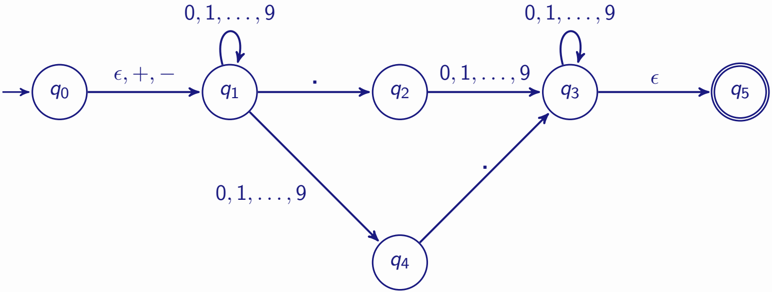
- Sinal $+$ ou $-$ opcional
- string de digitos
- ponto decimal
- outra string de digitos
- pelo menos uma string de digitos deve ser não vazia

NFA para reconhecimento de linguagem

Exemplo 4 (números decimais)

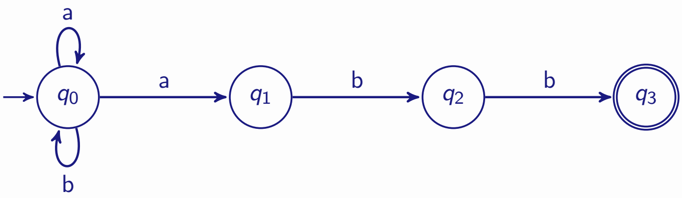
NFA que aceita números decimais:

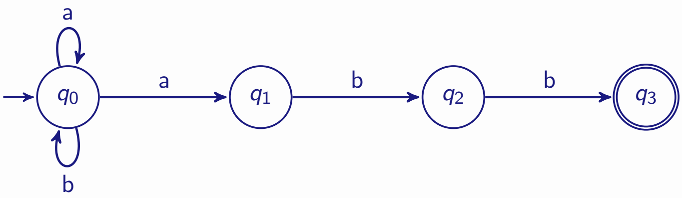
- Sinal $+$ ou $-$ opcional
- string de digitos
- ponto decimal
- outra string de digitos
- pelo menos uma string de digitos deve ser não vazia

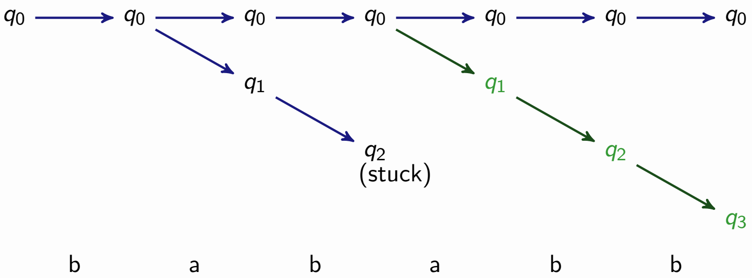


Autómatos finitos **não determinísticos** (NFA)

Não determinismo

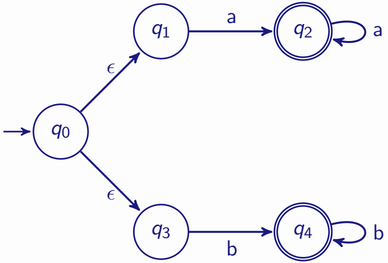



a
a
b
b
b
q_0
q_1
q_2
q_3

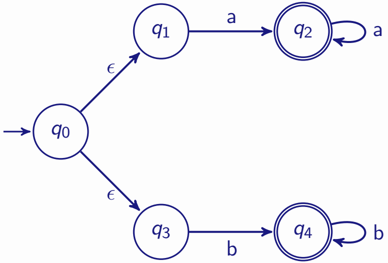

q_0
q_0
q_0
q_0
q_0
q_0
q_0
q_1
q_1
q_2
(stuck)
q_2
q_3
b
a
b
a
b
b

- Mecanismo para aceder aos estados atingíveis com transições ϵ
- Fecho-ϵ de q, ϵ-closure(q):
  - **Caso base:** Estado q está em ϵ-closure(q)
  - **Indução:** Se $p \in \epsilon$-closure(q), então ϵ-closure(q) contém os estados em $\delta(p, \epsilon)$

- Mecanismo para aceder aos estados atingíveis com transições ϵ
- Fecho-ϵ de q, ϵ-closure(q):
  - **Caso base:** Estado q está em ϵ-closure(q)
  - **Indução:** Se $p \in \epsilon$-closure(q), então ϵ-closure(q) contém os estados em $\delta(p, \epsilon)$

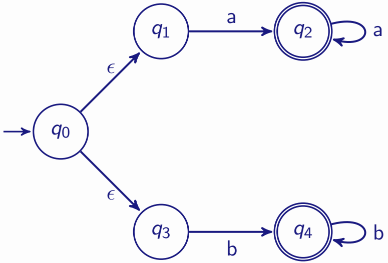


- Mecanismo para aceder aos estados atingíveis com transições ϵ
- Fecho-ϵ de q, ϵ-closure(q):
  - **Caso base:** Estado q está em ϵ-closure(q)
  - **Indução:** Se $p \in \epsilon$-closure(q), então ϵ-closure(q) contém os estados em $\delta(p, \epsilon)$



$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_3\}$$

- Mecanismo para aceder aos estados atingíveis com transições ϵ
- Fecho-ϵ de q, ϵ-closure(q):
  - **Caso base:** Estado q está em ϵ-closure(q)
  - **Indução:** Se $p \in \epsilon$-closure(q), então ϵ-closure(q) contém os estados em $\delta(p, \epsilon)$



$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_3\}$$
$$\epsilon\text{-closure}(q_1) = \{q_1\}$$

Outline

Conversão de REs para NFA

Qualquer linguagem descrita por uma expressão regular, é aceite por um autómato não determinista

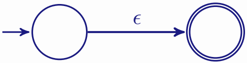
Conversão de REs para NFA

Qualquer linguagem descrita por uma expressão regular, é aceite por um autómato não determinista

O **Algoritmo de Thompson** permite converter uma expressão regular em um NFA

Caso base:

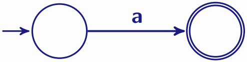
- Expressão ϵ:



Caso base:

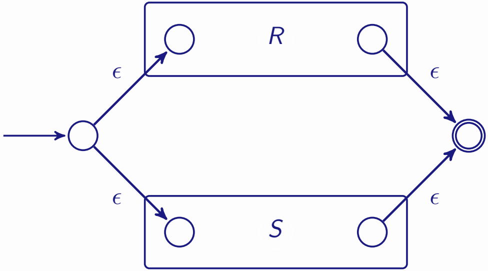
- Expressão ϵ:

- Expressão **a**:



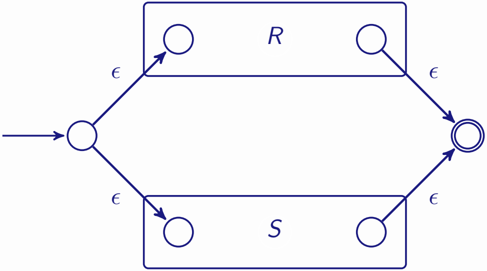
Indução:
Se as variáveis R e S são expressões regulares, então:

- Expressão $R|S$:



Indução:
Se as variáveis R e S são expressões regulares, então:

- Expressão $R|S$:

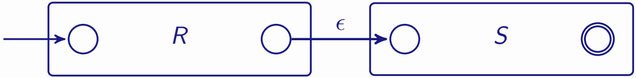


O autómato aceita $L(R) \cup L(S)$

Indução: (Cont.)

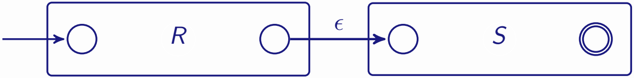
Se as variáveis R e S são expressões regulares, então:

- Expressão $R\,S$:



Indução: (Cont.)
Se as variáveis R e S são expressões regulares, então:

- Expressão $R\,S$:

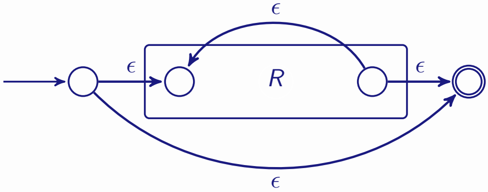


O autómato aceita $L(R)\,L(S)$

Indução: (Cont.)

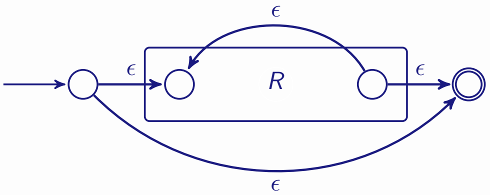
Se as variáveis R e S são expressões regulares, então:

- Expressão R^*:



Indução: (Cont.)
Se as variáveis R e S são expressões regulares, então:

- Expressão R^*:



O autómato aceita $L(\epsilon), L(R), L(R)L(R), \ldots$

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Primeiro, identificar componentes da expressão regular

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Primeiro, identificar componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Primeiro, identificar componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b
  - Um componente para $\mathbf{a}|\mathbf{b}$

Conversão de REs para NFA

Algoritmo de Thompson - Exemplo

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

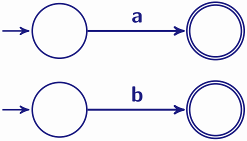
- Primeiro, identificar componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b
  - Um componente para $\mathbf{a}|\mathbf{b}$
  - Um componente para $(\mathbf{a}|\mathbf{b})^*$

Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Primeiro, identificar componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b
  - Um componente para $\mathbf{a}|\mathbf{b}$
  - Um componente para $(\mathbf{a}|\mathbf{b})^*$
  - Um componente final que agrega todos os outros componentes

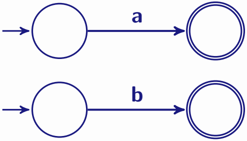
Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Expressões **a** e **b**:

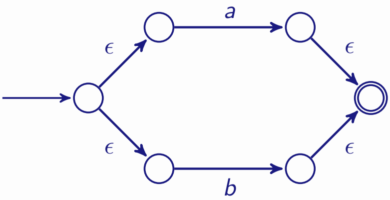


Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Expressões **a** e **b**:

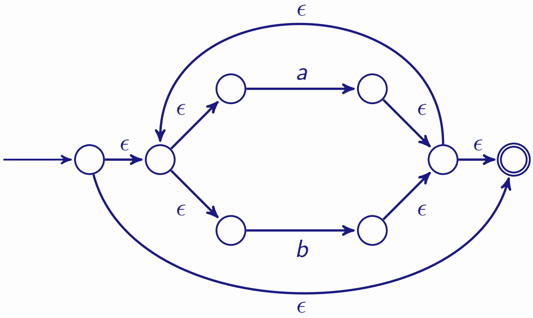


- Expressão $\mathbf{a}|\mathbf{b}$:



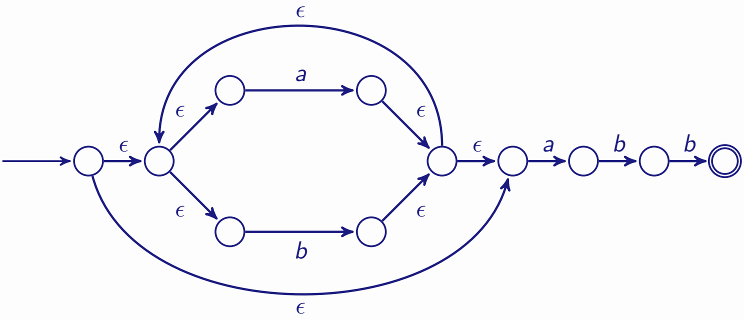
Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Expressão $(\mathbf{a}|\mathbf{b})^*$:



Construir NFA para $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

- Agregar todos os componentes:



Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b

Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada *a* e *b*
  - Um componente para $\mathbf{a}|\epsilon$

Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b
  - Um componente para $\mathbf{a}|\epsilon$
  - Um componente para $\mathbf{ab}$

Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada a e b
  - Um componente para $\mathbf{a}|\epsilon$
  - Um componente para $\mathbf{ab}$
  - Um componente para $(\mathbf{ab})^*$

Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada *a* e *b*
  - Um componente para $\mathbf{a}|\epsilon$
  - Um componente para **ab**
  - Um componente para $(\mathbf{ab})^*$
  - Um componente para $\mathbf{b}|\epsilon$

Conversão de REs para NFA

Outro exemplo

- Construir NFA para $(\mathbf{a}|\epsilon)(\mathbf{ab})^*(\mathbf{b}|\epsilon)$
- Primeiro, identificar os componentes da expressão regular
  - Um componente para cada símbolo de entrada *a* e *b*
  - Um componente para $\mathbf{a}|\epsilon$
  - Um componente para $\mathbf{ab}$
  - Um componente para $(\mathbf{ab})^*$
  - Um componente para $\mathbf{b}|\epsilon$
- Um componente final que agregue todos os outros componentes

Qualquer linguagem descrita por um NFA, é aceite por um DFA

Conversão de NFA para DFA

Qualquer linguagem descrita por um NFA, é aceite por um DFA

O algoritmo de **Construção de Subconjuntos** (*subset construction*) permite converter um NFA em um DFA

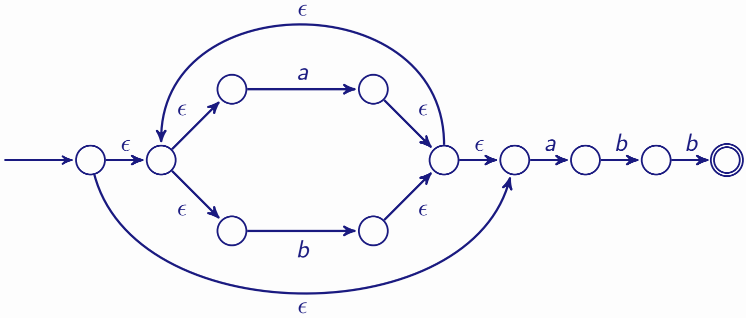
- A ideia geral consiste em que cada estado do DFA corresponde a um conjunto de estados do NFA, que são todos aqueis que são atingíveis após processar uma certa string de entrada

- **Caso base**: Conjunto singular consistindo do estado inicial do NFA é atingível
- **Indução**: Se um conjunto de estados S é atingível, então todos os estados alcançáveis a partir de S por transições vazias, também são atingíveis

Exemplo

Expressão regular: $(\mathbf{a}+\mathbf{b})^*\mathbf{abb}$

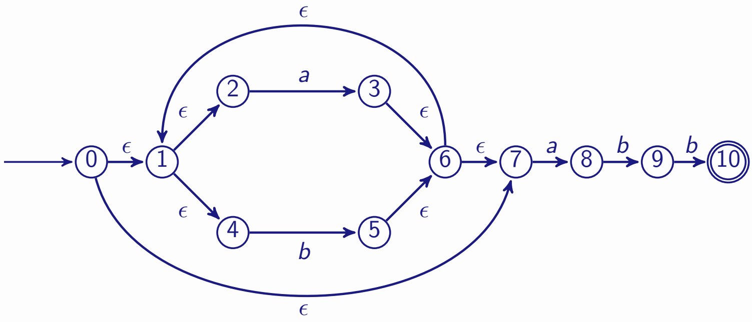
NFA:



Exemplo

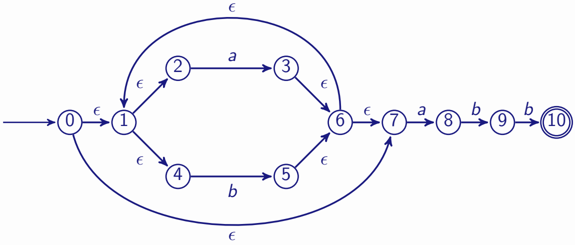
Expressão regular: $(\mathbf{a}+\mathbf{b})^{*}\mathbf{abb}$

NFA:



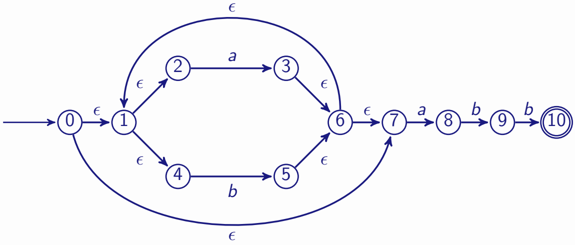
Conversão de NFA para DFA

Construção de Subconjuntos - Processo de determinização



Conversão de NFA para DFA

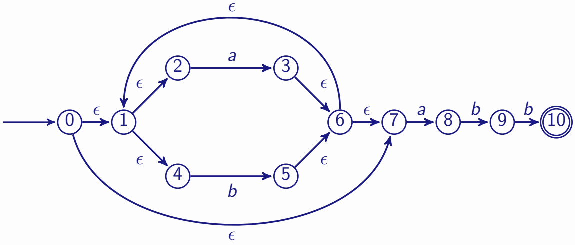
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}

Conversão de NFA para DFA

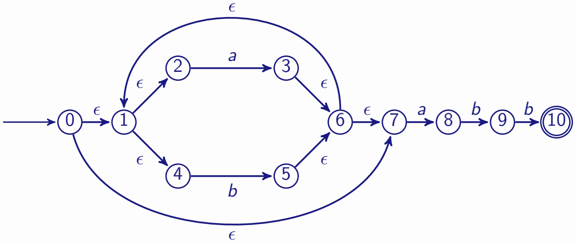
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**

Conversão de NFA para DFA

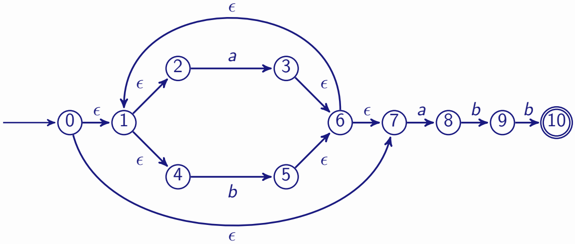
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**

Conversão de NFA para DFA

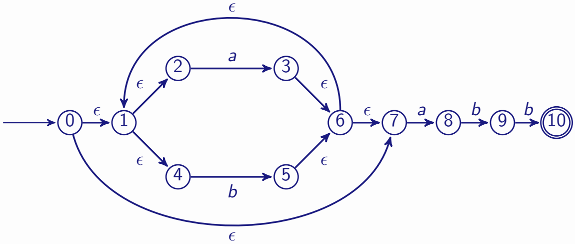
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**

Conversão de NFA para DFA

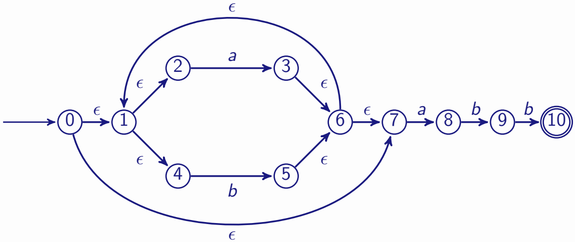
Construção de Subconjuntos - Processo de determinização



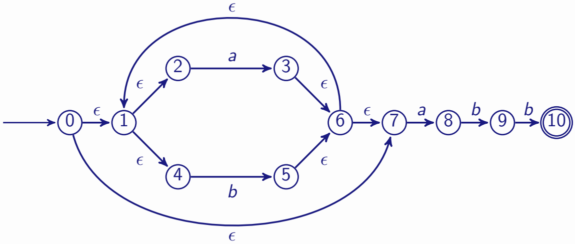
I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**

Conversão de NFA para DFA

Construção de Subconjuntos - Processo de determinização



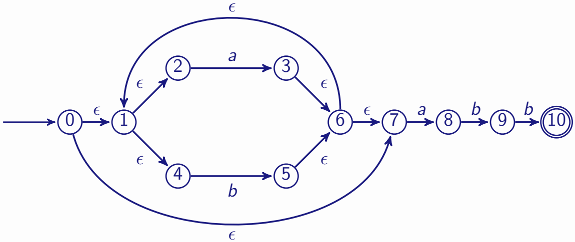
I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**



I_n	$\alpha \in \Sigma$	$\text{move}(I_n, \alpha)$	$\epsilon\text{-closure}(\text{move}(I_n, \alpha))$	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**

Conversão de NFA para DFA

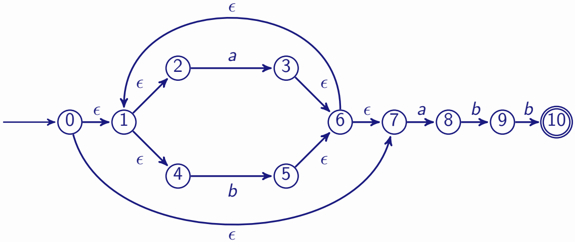
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**

Conversão de NFA para DFA

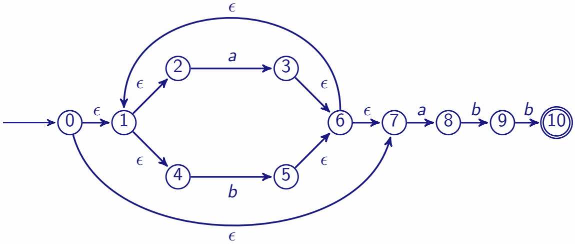
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**

Conversão de NFA para DFA

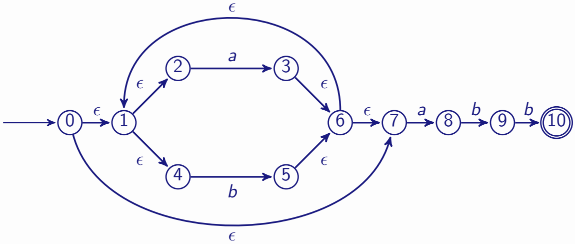
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
3	b	5, 10	1, 2, 4, 5, 6, 7, 10	**4**

Conversão de NFA para DFA

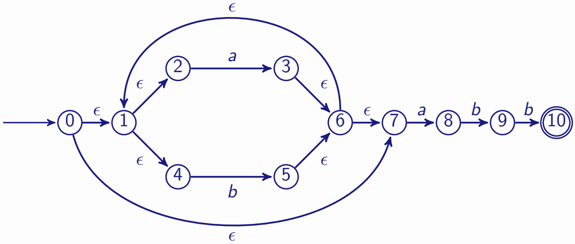
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
3	b	5, 10	1, 2, 4, 5, 6, 7, 10	**4**
4	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**

Conversão de NFA para DFA

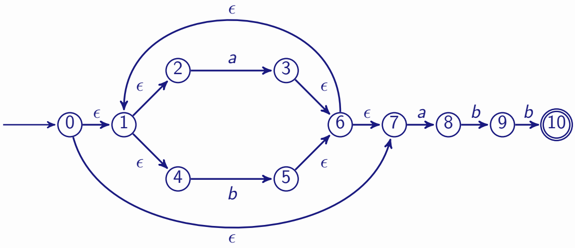
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
3	b	5, 10	1, 2, 4, 5, 6, 7, 10	**4**
4	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
4	b	5	1, 2, 4, 5, 6, 7	**2**

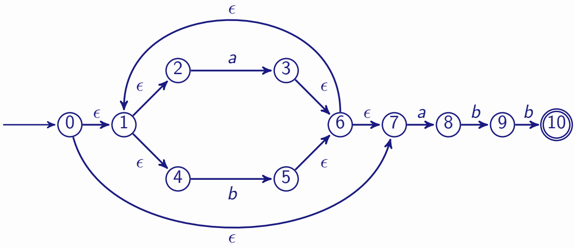
Conversão de NFA para DFA

Construção de Subconjuntos - Processo de determinização

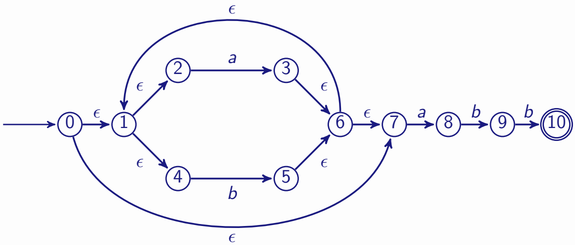


Conversão de NFA para DFA

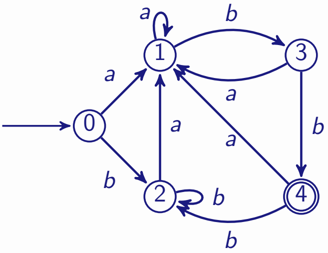
Construção de Subconjuntos - Processo de determinização



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
3	b	5, 10	1, 2, 4, 5, 6, 7, 10	**4**
4	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
4	b	5	1, 2, 4, 5, 6, 7	**2**



I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}
-	-	0	0, 1, 2, 4, 7	**0**
0	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
0	b	5	1, 2, 4, 5, 6, 7	**2**
1	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
1	b	5, 9	1, 2, 4, 5, 6, 7, 9	**3**
2	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
2	b	5	1, 2, 4, 5, 6, 7	**2**
3	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
3	b	5, 10	1, 2, 4, 5, 6, 7, 10	**4**
4	a	3, 8	1, 2, 3, 4, 6, 7, 8	**1**
4	b	5	1, 2, 4, 5, 6, 7	**2**



O número de estados do DFA obtido através da construção de subconjuntos não é, necessariamente, mínimo

Conversão de DFA para minDFA

O número de estados do DFA obtido através da construção de subconjuntos não é, necessariamente, mínimo

Abordagem:

- Aplicação do Algoritmo de Hopcroft

O número de estados do DFA obtido através da construção de subconjuntos não é, necessariamente, mínimo

Abordagem:

- Aplicação do Algoritmo de Hopcroft
  - Partição inicial constituída por: estados finais e estados não finais

O número de estados do DFA obtido através da construção de subconjuntos não é, necessariamente, mínimo

Abordagem:

- Aplicação do Algoritmo de Hopcroft
  - Partição inicial constituída por: estados finais e estados não finais
  - Fragmentar partições por símbolos de entrada diferentes

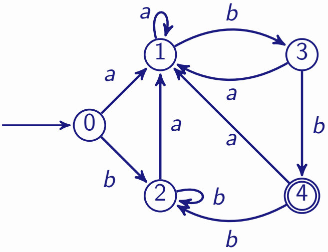
O número de estados do DFA obtido através da construção de subconjuntos não é, necessariamente, mínimo

Abordagem:

- Aplicação do Algoritmo de Hopcroft
  - Partição inicial constituída por: estados finais e estados não finais
  - Fragmentar partições por símbolos de entrada diferentes
    - Repetir até que não sejam criados novos subgrupos

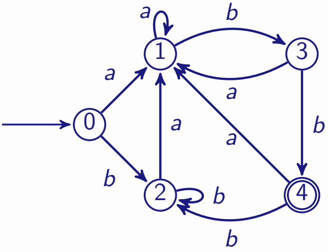
Exemplo:

Autómato finito determinista (DFA)



Exemplo:

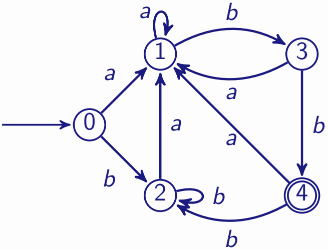
Autómato finito determinista (DFA)

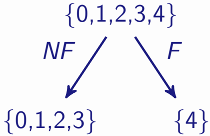


{0,1,2,3,4}

Exemplo:

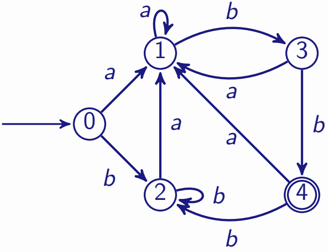
Autómato finito determinista (DFA)

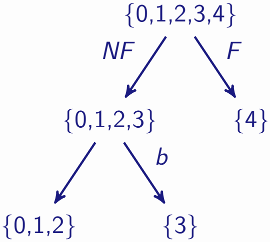




Exemplo:

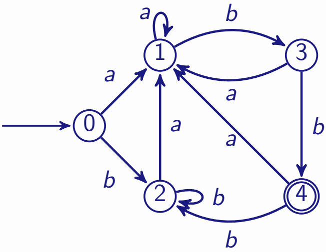
Autómato finito determinista (DFA)

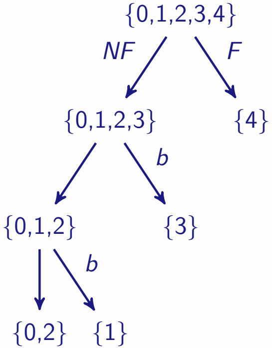




Exemplo:

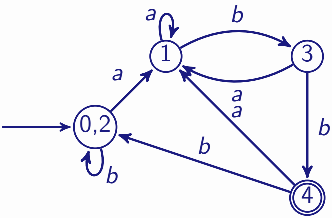
Autómato finito determinista (DFA)

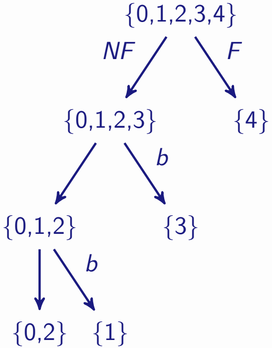




Exemplo:

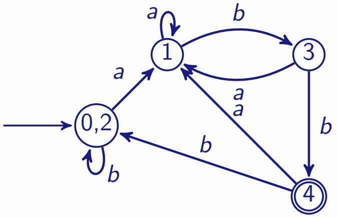
Autómato finito determinista (DFA)

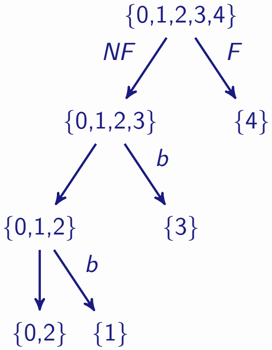




Exemplo: (Final)

Autómato finito determinista (DFA)





Outline

Reconhecimento de elementos da linguagem

Expressões Regulares

Autómatos Finitos

Conversão entre Representações

Reconhecimento multi-string

Reconhecimento multi-string

Depois de obtido o DFA mínimo, pode iniciar-se o processamento das palavras na entrada, e decidir se são ou não aceites pela expressão regular

Problema:

- O reconhecimento de uma palavra é apenas parte do problema
- Um analisador lexical deve conseguir reconhecer uma frase (multi-palavra)

Reconhecimento multi-string

Abordagem: Dividir a linguagem numa união de expressões regulares $L = (r_1 \mid r_2 | \ldots \mid r_n)^*$

Subdivisão do problema devido às propriedades de fecho

- Considerar um estado inicial que deriva, através de transições vazias, cada uma das expressões regulares
- Associar cada estado final à respectiva expressão regular
- Após o reconhecimento, o analisador recomeça no estado inicial

Nota: o processamento considera sempre o reconhecimento:

- da sequência mais comprida
- das palavras pela ordem da entrada

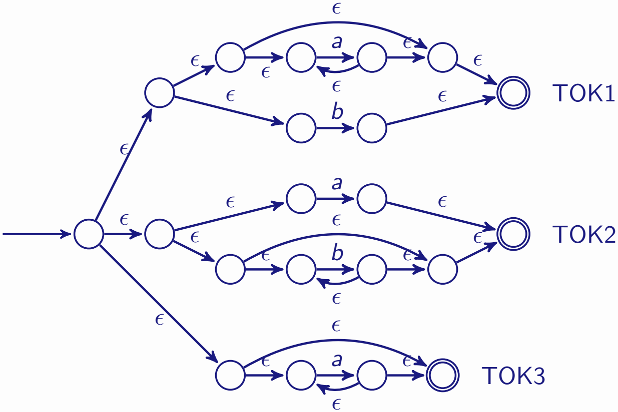
$$L = \{a^*|b, a|b^*, a^*\}$$

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

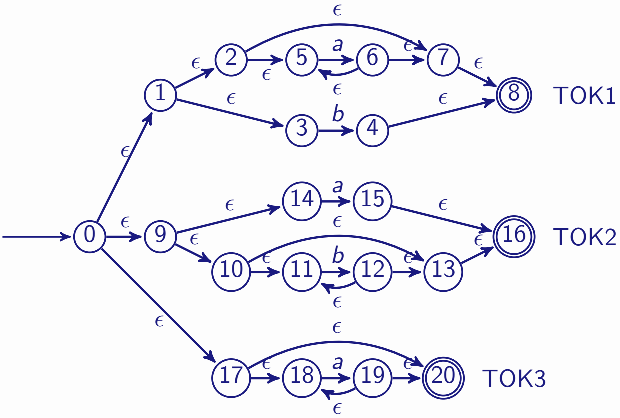
Reconhecimento multi-string

1a Etapa: Conversão REs → NFA - Algoritmo de Thompson

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$



$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$



$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $TOK1 = a^*|b$ $TOK2 = a|b^*$ $TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2
3	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2
3	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
3	b	-	-	-	-

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2
3	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
3	b	-	-	-	-
4	a	-	-	-	-

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2
3	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
3	b	-	-	-	-
4	a	-	-	-	-
4	b	12	11, 12, 13, 16	**4**	TOK2

$L = \{a^*|b, a|b^*, a^*\}$ $TOK1 = a^*|b$ $TOK2 = a|b^*$ $TOK3 = a^*$

Nota: A tabela de análise é diferente! (coluna Token adicional)

O token emitido é o correspondente ao estado final com menor númeração.

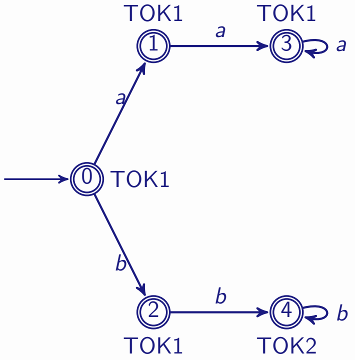
I_n	$\alpha \in \Sigma$	move(I_n, α)	ϵ-closure(move(I_n, α))	I_{n+1}	Token
-	-	0	0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 20	**0**	TOK1
0	a	6, 15, 19	5, 6, 7, 8, 15, 16, 18, 19, 20	**1**	TOK1
0	b	4, 12	4, 8, 11, 12, 13, 16	**2**	TOK1
1	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
1	b	-	-	-	-
2	a	-	-	-	-
2	b	12	11, 12, 13, 16	**4**	TOK2
3	a	6, 19	5, 6, 7, 8, 18, 19, 20	**3**	TOK1
3	b	-	-	-	-
4	a	-	-	-	-
4	b	12	11, 12, 13, 16	**4**	TOK2

Reconhecimento multi-string

2a Etapa: Conversão NFA $\rightarrow$ DFA

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

Nota: Os estados finais são anotados com o Token correspondente que reconhecem.

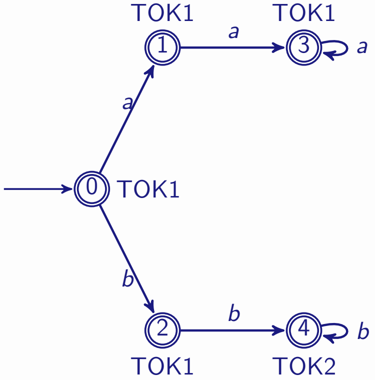


Reconhecimento multi-string

3a Etapa: Minimização do DFA

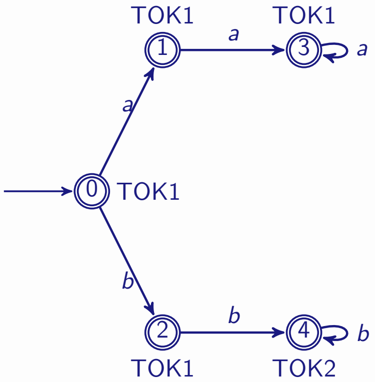
$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

Nota: A minimização é diferente! (adicionalmente separar por token)



$L = \{a^*|b, a|b^*, a^*\}$ $\quad$ $TOK1 = a^*|b$ $\quad$ $TOK2 = a|b^*$ $\quad$ $TOK3 = a^*$

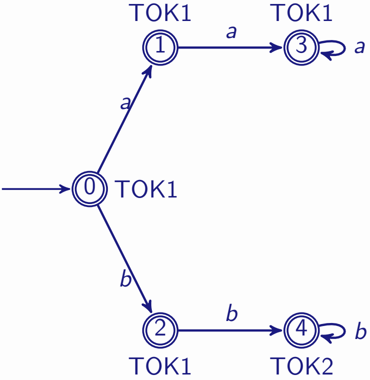
Nota: A minimização é diferente! (adicionalmente separar por token)

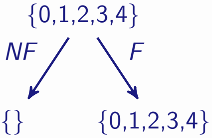


$\{0,1,2,3,4\}$

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

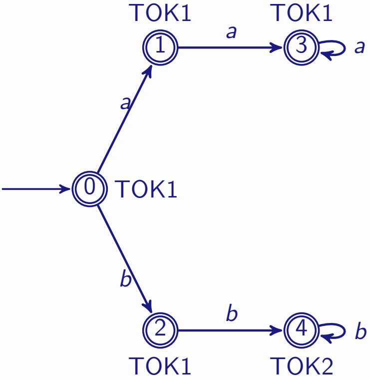
Nota: A minimização é diferente! (adicionalmente separar por token)

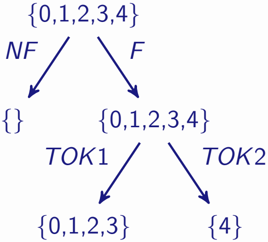




$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

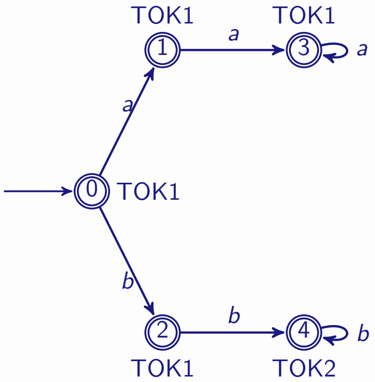
Nota: A minimização é diferente! (adicionalmente separar por token)

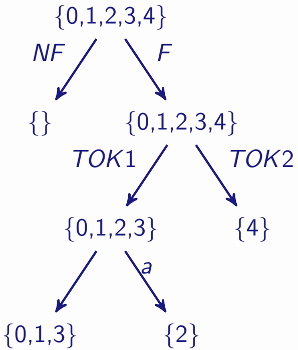




$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

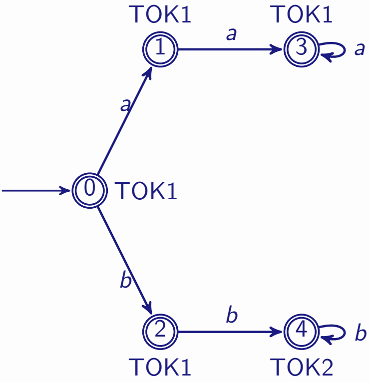
Nota: A minimização é diferente! (adicionalmente separar por token)

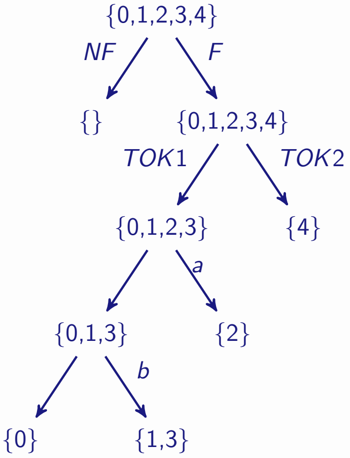




$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

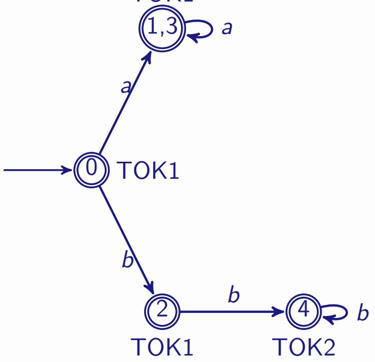
Nota: A minimização é diferente! (adicionalmente separar por token)

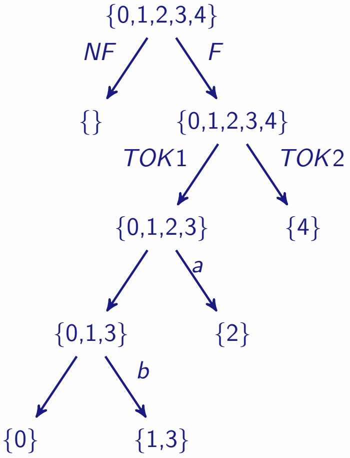




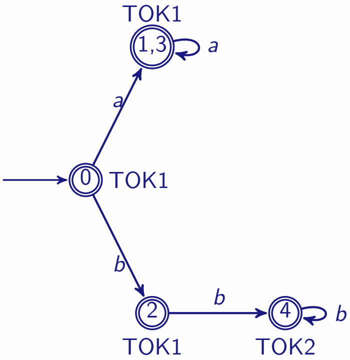
$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

Nota: A minimização é diferente! (adicionalmente separar por token)





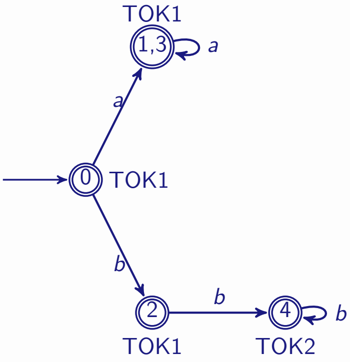
$L = \{a^*|b, a|b^*, a^*\}$ $TOK1 = a^*|b$ $TOK2 = a|b^*$ $TOK3 = a^*$



Reconhecimento multi-string

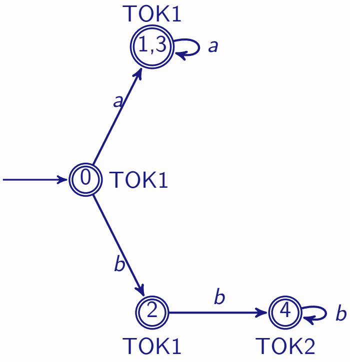
4a Etapa: Processamento da entrada

$$L = \{a^*|b, a|b^*, a^*\} \qquad TOK1 = a^*|b \qquad TOK2 = a|b^* \qquad TOK3 = a^*$$



I_n	Entrada	I_{n+1}/Token

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

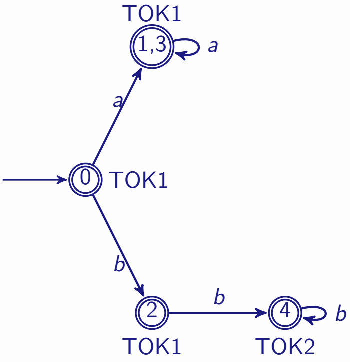


I_n	Entrada	I_{n+1}/Token
0	aababb$	13

Reconhecimento multi-string

4a Etapa: Processamento da entrada

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$

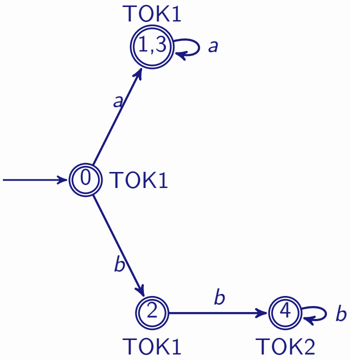


I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13

Reconhecimento multi-string

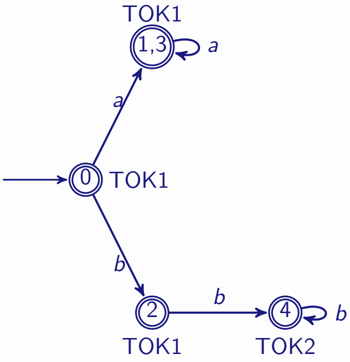
4a Etapa: Processamento da entrada

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$



I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1

$$L = \{a^*|b, a|b^*, a^*\} \qquad TOK1 = a^*|b \qquad TOK2 = a|b^* \qquad TOK3 = a^*$$

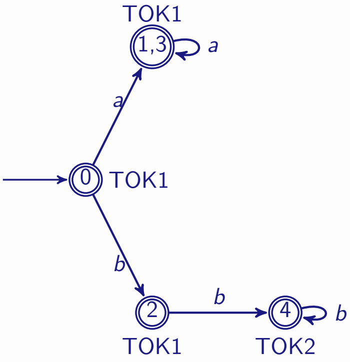


I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2

Reconhecimento multi-string

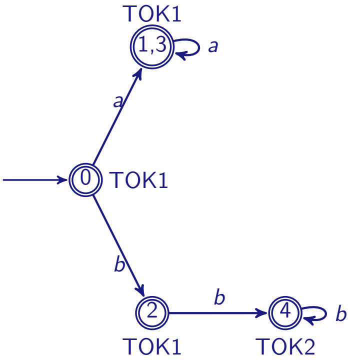
4a Etapa: Processamento da entrada

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$



I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1

$$L = \{a^*|b, a|b^*, a^*\} \qquad TOK1 = a^*|b \qquad TOK2 = a|b^* \qquad TOK3 = a^*$$

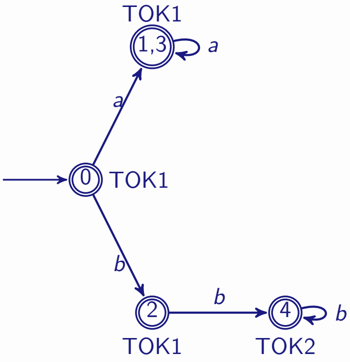


I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13

Reconhecimento multi-string

4a Etapa: Processamento da entrada

$$L = \{a^*|b, a|b^*, a^*\} \qquad TOK1 = a^*|b \qquad TOK2 = a|b^* \qquad TOK3 = a^*$$

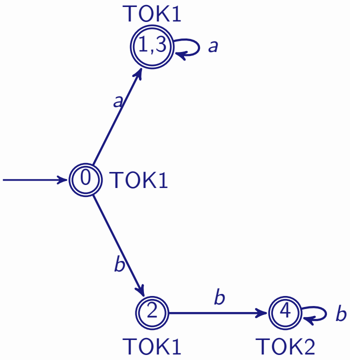


I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13
13	bb$	TOK1

Reconhecimento multi-string

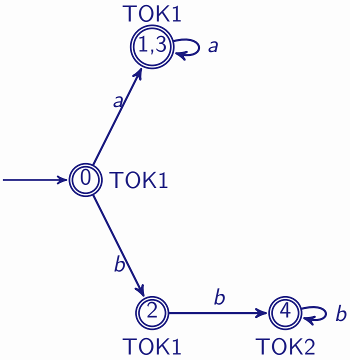
4a Etapa: Processamento da entrada

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b \quad TOK2 = a|b^* \quad TOK3 = a^*$



I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13
13	bb$	TOK1
0	bb$	2

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$

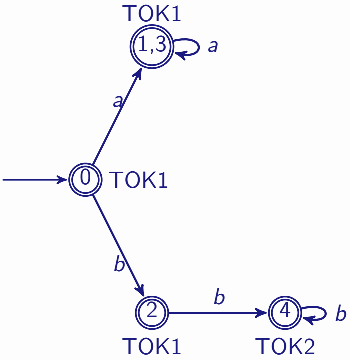


I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13
13	bb$	TOK1
0	bb$	2
2	b$	4

Reconhecimento multi-string

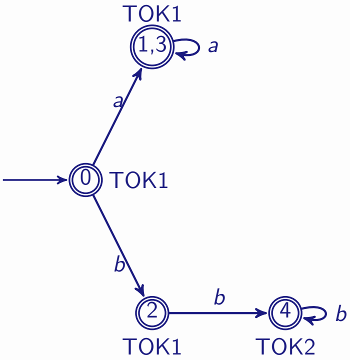
4a Etapa: Processamento da entrada

$$L = \{a^*|b, a|b^*, a^*\} \qquad TOK1 = a^*|b \qquad TOK2 = a|b^* \qquad TOK3 = a^*$$



I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13
13	bb$	TOK1
0	bb$	2
2	b$	4
4	$	TOK2

$L = \{a^*|b, a|b^*, a^*\}$ $\quad TOK1 = a^*|b$ $\quad TOK2 = a|b^*$ $\quad TOK3 = a^*$



I_n	Entrada	I_{n+1}/Token
0	aababb$	13
13	ababb$	13
13	babb$	TOK1
0	babb$	2
2	abb$	TOK1
0	abb$	13
13	bb$	TOK1
0	bb$	2
2	b$	4
4	$	TOK2

A entrada *aababb* é processada em 10 passos,
com os seguintes tokens e lexemas correspondentes:
TOK1 (aa), TOK1 (b), TOK1 (a), TOK2 (bb)

Backtracking:

- Caso não seja possível avançar a partir de um estado
- Faz backtracking até ao último estado final visto
  - os símbolos consumidos desde o último estado final voltam para a entrada
  - o token é emitido
- Caso se volte até ao estado inicial e este não for final, a entrada não pertence à linguagem

Reconhecimento multi-string

4a Etapa: Processamento da entrada

Backtracking:

- Caso não seja possível avançar a partir de um estado
- Faz backtracking até ao último estado final visto
  - os símbolos consumidos desde o último estado final voltam para a entrada
  - o token é emitido
- Caso se volte até ao estado inicial e este não for final, a entrada não pertence à linguagem

Exemplo: https://web.tecnico.ulisboa.pt/~david.matos/w/pt/index.php/Theoretical_Aspects_of_Lexical_Analysis/Exercise_6

Questões?

Dúvidas?