Aula 00 – Autômatos Finitos

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação

Introdução

- Os autômatos finitos (AFs) formam uma família de mecanismos abstratos de computação.
- Servem como uma abstração de inúmeros dispositivos mecânicos e eletro/eletrônicos.
- **Estes *slides***: definição, conceitos e propriedades de AFs.
- **Objetivos**: revisar/apresentar o fundamental teórico para a comparação com Máquinas de Turing.

Referências

Chapter 5 – Finite Automata
T. Sudkamp
Chapter 1 – Regular Languages
M. Sipser
Chapter 2 – Finite Automata and Regular Languages
A. Maheshwari

Introdução

- Computação de um AF determina se uma string:
  - satisfaz um conjunto de condições; ou
  - corresponde a um padrão definido.
- Entrada de um AF: uma string *s*.
- Saída de um AF: aceita ou rejeita *s*.
- Exemplos de materializações de uma AFs: máquina de venda de bebidas, fechadura com teclado numérico, etc.
- Nestes exemplos, espera-se que o comportamento das AF seja determinístico.
- Ao final dos *slides*: estudo dos efeitos do não-determinismo nas capacidades das AFs.

Uma Máquina de Estados Finita

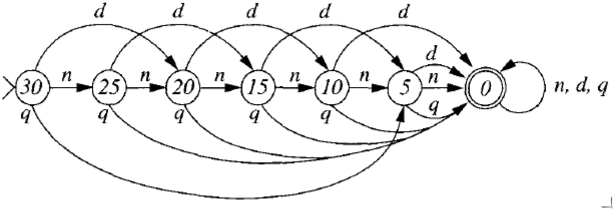
- Uma definição formal de uma máquina não se importa com *hardware*: irrelevante.
- Se importa com: descrição das operações internas realizadas para processar a entrada.
- O quê deve ser abstraído e o quê deve ficar na representação de uma máquina (sistema)?

Exemplo: máquina de venda automática

- Suponha um dispositivo de venda automática.
- Esse dispositivo é uma máquina de estados finita.
- Entrada: moedas de 5 centavos (*nickel* – *n*), 10 centavos (*dimes* – *d*) e 25 centavos (*quarter* – *q*).
- Quando 30 centavos são inseridos ⇒ máquina libera o produto.
- Máquina não dá troco. "Rouba" valores que excederem os 30 centavos.
- Máquina tem memória limitada ⇒ "sabe" a quantidade de dinheiro inserida ou faltando.
- Informação registrada no estado interno da máquina.
- Estado é alterado sempre que uma entrada (moeda) é recebida e processada.

Exemplo: máquina de venda automática

- Estados são rotulados com a quantidade de centavos faltando para completar o preço do produto.
- Projeto da máquina pode ser representado pelo seguinte diagrama de estados (um grafo direcionado rotulado).



- Estado 30 é o estado inicial.
- Estado 0 é o estado de aceite.
- Arcos são rotulados com *n*, *d* ou *q*, indicando a moeda de entrada.

Exemplo: máquina de venda automática

- Entrada da máquina: strings do conjunto $\{n, d, q\}^*$.
- Execução começa no estado inicial.
- Cada símbolo da entrada gera uma transição pelo arco correspondente.
- Se não há transição compatível com símbolo atual $\Rightarrow$ erro.
- Repete até processar toda a string.
- Se a computação termina em um estado de aceite $\Rightarrow$ aceita string.
- Senão rejeita.
- Exemplo: máquina do slide anterior aceita *dndn* e rejeita *nndn*.

Autômatos Finitos Determinísticos

A descrição independente de implementação do exemplo é chamada de máquina abstrata. A definição abaixo apresenta um tipo dessas máquinas.

Definição 5.2.1 (Sudkamp)

Um autômato finito determinístico – *deterministic finite automaton* (DFA) é uma tupla $M = (Q, \Sigma, \delta, q_o, F)$ aonde:

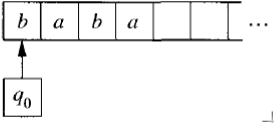
- Q é um conjunto finito de estados;
- Σ é um alfabeto;
- $q_0 \in Q$ é o estado inicial;
- $F \subseteq Q$ são os estados finais ou de aceite; e
- $\delta : Q \times \Sigma \to Q$ é a função de transição.

Autômatos Finitos Determinísticos

- Um DFA pode ser imaginado como uma máquina física (mecânica), composta por:
  1. um único registrador interno: indica o estado atual;
  2. um conjunto de valores para o registrador: Q;
  3. uma fita contendo a entrada;
  4. uma cabeça de leitura para fita; e
  5. um conjunto de instruções.
- A fita é dividida em quadrados, cada um guarda um símbolo de Σ.
- Não existe um limite superior para o tamanho da entrada.
- $\Rightarrow$ Fita tem comprimento arbitrário (*unbounded*).
- Entrada fica no início da fita.

Autômatos Finitos Determinísticos

- Cabeça de leitura lê um único quadrado da fita por vez.
- Corpo da máquina: cabeça de leitura + registrador.
- Exemplo: configuração inicial de uma computação com entrada *baba*:

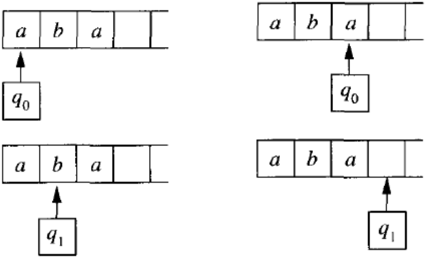


- O estado da máquina q_i e o símbolo lido a determina a instrução a ser executada (transição do DFA).
- Execução da instrução altera o estado e move a cabeça de leitura um quadrado para a direita.
- Próximo estado é $\delta(q_i, a)$. Identificado de forma única.

Autômatos Finitos Determinísticos

- Objetivo da computação de um DFA: aceitar/rejeitar string.
- Computação termina quando a cabeça lê um quadrado vazio (branco).
- Aceite da string *aba*.

$$
\begin{array}{lll}
\text{M:} & Q = \{q_0, q_1\} & \delta(q_0, a) = q_1 \\
 & \Sigma = \{a, b\} & \delta(q_0, b) = q_0 \\
 & F = \{q_1\} & \delta(q_1, a) = q_1 \\
 & & \delta(q_1, b) = q_0
\end{array}
$$



Autômatos Finitos Determinísticos

Um DFA é um aceitador (reconhecedor) de uma linguagem.

Definição 5.2.2 (Sudkamp)

Seja $M = (Q, \Sigma, \delta, q_o, F)$ um DFA. A linguagem de M, denotada por L(M), é o conjunto das strings em Σ^* aceitas por M.

- Linguagem do exemplo anterior?
- Conjunto de todas as strings sobre $\{a, b\}$ que terminam em a.

Autômatos Finitos Determinísticos

- DFA é uma máquina *read-only*: processa a entrada da esquerda para a direita e nunca mais volta.
- Um símbolo depois de lido não afeta mais a computação.
- $\Rightarrow$ O resultado da computação depende somente do estado atual e da parte da entrada ainda não processada.
- Isso é chamado de configuração da máquina, representado por $[q_i, w]$.
- $q_i \in \mathrm{Q}$ é o estado atual.
- $w \in \Sigma^*$ é a entrada ainda não processada.
- Um ciclo de instrução de um DFA transforma uma configuração em outra.
- *Notação*: $[q_i, aw] \vdash_{\overline{\mathrm{M}}} [q_j, w]$.
- Leia $\vdash_{\overline{\mathrm{M}}}$ como "gera", "computa" ou "produz" (*"yields"*).

Autômatos Finitos Determinísticos

Definição 5.2.2 (Sudkamp)

A função $\vdash_{\overline{M}}$ sobre $Q \times \Sigma^+$ é definida como

$$[q_i, aw] \vdash_{\overline{M}} [\delta(q_i, a), w]$$

para $a \in \Sigma$ e $w \in \Sigma^*$, onde δ é a função de transição do DFA M.

- Omite-se o M do símbolo quando não há possibilidade de ambiguidade.
- $[q_i, u] \overset{*}{\vdash} [q_j, v]$ indica que a configuração $[q_j, v]$ pode ser obtida de $[q_i, u]$ através de zero ou mais transições.

Autômatos Finitos Determinísticos

Exemplo 5.2.1 (Sudkamp)

O DFA M definido abaixo aceita as strings que contém a substring *bb*, i.e., $L(M) = (\mathbf{a} \cup \mathbf{b})^*\mathbf{bb}(\mathbf{a} \cup \mathbf{b})^*$. Elementos de M:

$$Q = \{q_0, q_1, q_2\} \qquad \Sigma = \{a, b\} \qquad F = \{q_2\}.$$

A função de transição δ é dada como uma tabela de transição.

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

Traces das computações de M para duas possíveis entradas.

$$[q_0, abba] \vdash [q_0, bba] \vdash [q_1, ba] \vdash [q_2, a] \vdash [q_2, \lambda] \quad \surd$$

$$[q_0, abab] \vdash [q_0, bab] \vdash [q_1, ab] \vdash [q_0, b] \vdash [q_1, \lambda] \quad \times$$

Autômatos Finitos Determinísticos

- Função de transição δ especifica ação da máquina para um dado estado e símbolo do alfabeto.
- Conveniente estender para a função $\hat{\delta}$ cuja entrada é um estado e uma string sobre o alfabeto.

Definição 5.2.4 (Sudkamp)

A função de transição estendida de um DFA é a função $\hat{\delta} : Q \times \Sigma^* \to Q$. Os valores de $\hat{\delta}$ são definidos recursivamente pelo comprimento da string de entrada.

1. **Base:** $\hat{\delta}(q_i, \lambda) = q_i$ e $\hat{\delta}(q_i, a) = \delta(q_i, a)$ para todo $a \in \Sigma$.
2. **Passo recursivo:** $\hat{\delta}(q_i, ua) = \delta(\hat{\delta}(q_i, u), a)$ para qualquer string u onde $length(u) > 0$.

- A computação de uma máquina no estado q_i com string w para no estado $\hat{\delta}(q_i, w)$.
- Quando w é a string de entrada completa, $\hat{\delta}(q_0, w)$ indica o estado final da computação.
- A função $\hat{\delta}(q_0, w)$ simula as aplicações recursivas de δ para processar w.
- A string w é aceita se $\hat{\delta}(q_0, w) \in \mathrm{F}$.
- A linguagem de um DFA M é

$$\mathrm{L}(\mathrm{M}) = \{w \mid \hat{\delta}(q_0, w) \in \mathrm{F}\}.$$

Diagrama de Estados

- Um diagrama de estados de um DFA é um grafo direcionado e rotulado onde:
  - Nós representam os estados da máquina; e
  - Arcos representam as transições de δ.
- Existe uma correspondência inequívoca entre a definição de um DFA e seu diagrama de estados (Def. 5.3.1).
- $\Rightarrow$ Usa-se o diagrama de estados para (exibir/definir) um DFA por ser mais intuitivo.
- Transição do DFA $\Rightarrow$ Arco no diagrama de estados.
- Uma sequência de arcos no diagrama define um caminho *(path)* que “soletra” a string de entrada.

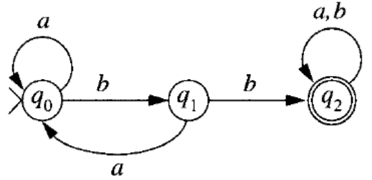
Diagrama de Estados

- As computações de um DFA e os caminhos no seu diagrama de estados são equivalentes.
- Seja $\mathbf{p}_w$ um caminho começando em q_0 que soletra w e seja q_w o nó terminal de $\mathbf{p}_w$. É possível provar que (Teorema 5.3.2):
  - Existe um único caminho para cada string $w \in \Sigma^*$.
  - Ao término do processamento de w, o DFA está no estado q_w.
- A equivalência aqui descrita permite que os *traces* da computação do DFA (operador $\vdash$) sejam interpretados visualmente como caminhos no diagrama.

Diagrama de Estados

Exemplo 5.3.1 (Sudkamp)

O diagrama de estados do DFA do Exemplo 5.2.1 é:

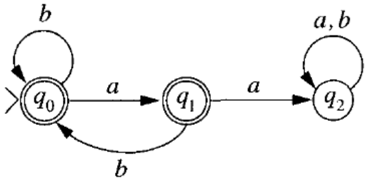


Estados registram o número de *b*'s consecutivos que já foram processados.

Diagrama de Estados

Exemplo 5.3.2 (Sudkamp)

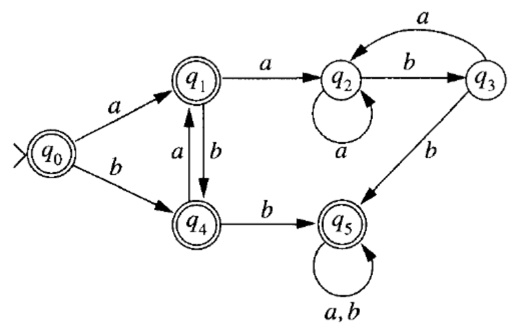
Qual é a linguagem do DFA M abaixo?



$L(M) = (\mathbf{b} \cup \mathbf{ab})^*(\mathbf{a} \cup \lambda)$, o conjunto de strings sobre $\{a, b\}$ que não contém a substring *aa*.

Exemplo 5.3.3 (Sudkamp)

Strings sobre $\{a, b\}$ que contém a substring *bb* ou não contém a substring *aa* são aceitas pelo DFA abaixo:



Esta linguagem é a união das linguagens dos dois exemplos anteriores.

Exemplo 5.3.6 (Sudkamp)

Seja $\Sigma = \{0, 1, 2, 3\}$. Uma string em Σ^* é uma sequência de dígitos de Σ. O DFA M abaixo determina se a soma dos dígitos da string de entrada é divisível por quatro.

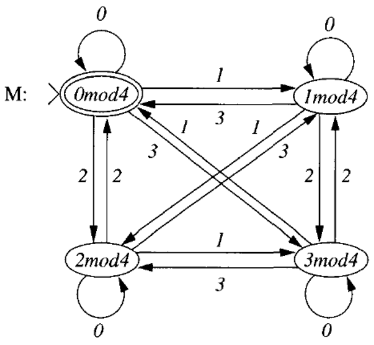
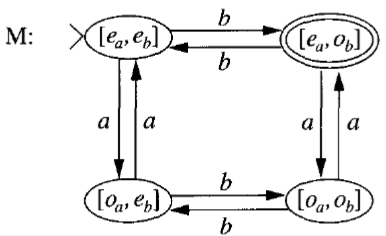


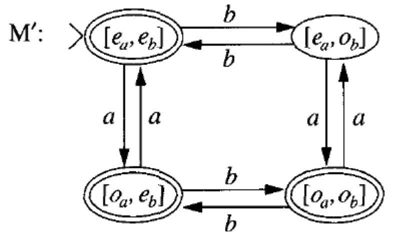
Diagrama de Estados

- Definição de DFA permite somente duas possíveis saídas: aceitar ou rejeitar a string de entrada.
- Estender a definição de saída associando um valor com cada estado.
- Resultado da computação é o valor associado ao estado onde a computação termina.
- $\Rightarrow$ Máquina de Moore (E.F. Moore, 1956).
- Outra possibilidade: associar saída com as transições da máquina.
- $\Rightarrow$ Máquina de Mealy (G.H. Mealy, 1955).
- Máquina do exemplo anterior é um somador módulo 4: associe o valor i ao estado $i mod 4$.

Exemplos 5.3.7 e 5.3.8 (Sudkamp)

DFA M aceita a linguagem formada pelas strings sobre $\{a, b\}$ que contém um número par de a's e um número ímpar de b's.





DFA M′ aceita a linguagem complementar de M, isto é:

$$\mathrm{L}(\mathrm{M}') = \{a, b\}^* - \mathrm{L}(\mathrm{M}).$$

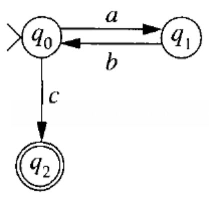
Diagrama de Estados

- Por definição, um DFA precisa processar a entrada toda.
- Desnecessário quando o resultado já foi estabelecido (e.g., DFA em um estado sem transição).
- Solução: determinismo incompleto – cada configuração tem no máximo uma ação especificada.
- $\Rightarrow$ A função δ passa a ser parcial.
- Computação agora pode parar se não há uma transição válida para o símbolo lido.
- Computação que termina antes de processar toda a string rejeita a entrada.

Diagrama de Estados

Exemplo 5.3.9 (Sudkamp)

O diagrama de estados abaixo define um DFA com especificação incompleta que aceita $(\mathbf{ab})^*\mathbf{c}$.



Computação para entrada *abcc* é rejeitada pois não é possível processar o último c no estado q_2.

Diagrama de Estados

- Duas máquinas que aceitam a mesma linguagem são ditas equivalentes.
- DFA com especificação incompleta $\Rightarrow$ DFA equivalente: adicionar um estado de erro.

Exemplo 5.3.10 (Sudkamp)

O DFA abaixo aceita a mesma linguagem do DFA anterior.

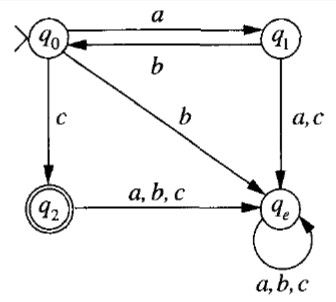
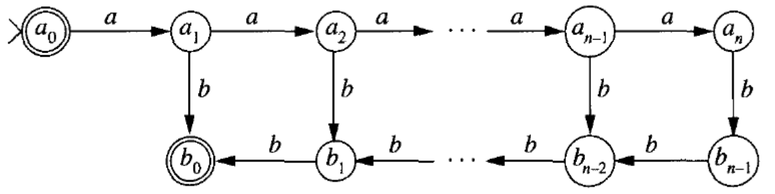


Diagrama de Estados

Exemplo 5.3.11 (Sudkamp)

O DFA com especificação incompleta definido pelo diagrama abaixo aceita a linguagem $\{a^i b^i \mid i \leq n\}$, para um natural fixo n.



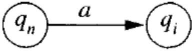
- Estados a_k contam o número de *a*'s.
- Estados b_k garantem o mesmo número de *b*'s.
- É possível fazer o mesmo para aceitar $\{a^i b^i \mid i \geq 0\}$?
- $\Rightarrow$ Não. Requer um número infinito de estados.

Autômatos Finitos Não-Determinísticos

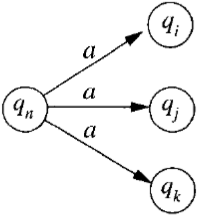
- Modificar definição de máquina para permitir computações não-determinísticas.
- Em um autômato finito não-determinístico – *nondeterministic finite automaton* (NFA), várias instruções podem ser executadas para uma dada configuração.
- Contra-intuitivo para máquinas de computação.
- Mas flexibilidade do não-determinismo facilita a construção do autômato.
- Uma transição em um NFA tem o mesmo efeito que em um DFA: mudar o estado da máquina a partir da configuração atual.

Autômatos Finitos Não-Determinísticos

- Função de transição deve indicar todos os possíveis estados alvo.
- $\Rightarrow$ Valor da função δ agora é um conjunto de estados.
- Formalmente, $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ (Def. 5.4.1).
- Demais componentes de um NFA são iguais ao do DFA.
- Exemplos de δ não-determinísticas:



$\delta(q_n, a) = \{q_i\}$

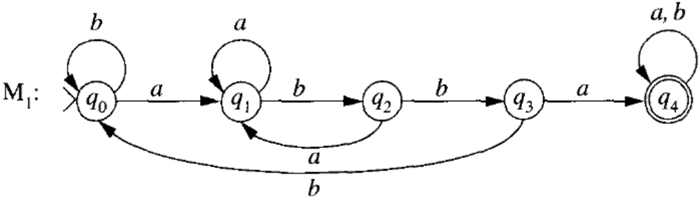


$\delta(q_n, a) = \{q_i, q_j, q_k\}$

$\delta(q_n, a) = \emptyset$

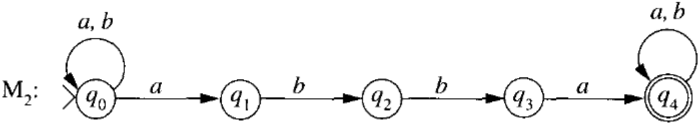
Autômatos Finitos Não-Determinísticos

- Exemplo para ilustrar as principais diferenças entre DFAs e NFAs.
- Seja o DFA M_1, com $L(M_1) = (\mathbf{a} \cup \mathbf{b})^*\mathbf{abba}(\mathbf{a} \cup \mathbf{b})^*$:



- Estados q_0 a q_3 gravam o "progresso" na obtenção da substring *abba*.
- Computação determinística precisa "retroceder" quando uma substring não possui a forma desejada.

- Um NFA M_2 com $L(M_2) = L(M_1)$:



- Não-determinismo está na escolha da transição saindo de q_0 quando um a é processado.
- NFA $\Rightarrow$ múltiplas computações para uma string de entrada!
- Exemplo: computações de M_2 para string *aabba*.

$[q_0, aabba] \vdash [q_1, abba]$

$[q_0, aabba] \vdash [q_0, abba] \vdash [q_1, bba] \vdash [q_2, ba] \vdash [q_3, a] \vdash [q_4, \lambda]$

- Ainda há outras possibilidades de computações além das acima.

Autômatos Finitos Não-Determinísticos

- Qual é o critério de aceite de uma string em um NFA?
- Deve existir ao menos uma computação que:
  1. processa toda a string de entrada; e
  2. termina em um estado de aceite.
- No exemplo anterior: a string *aabba* é aceita por M_2.
- A existência de outras computações que não aceitam a string é irrelevante.

Definição 5.4.2 (Sudkamp)

A linguagem de um NFA M, denotada por L(M), é o conjunto de strings aceitas por M. Isto é, $L(M) = \{w \mid$ existe uma computação $[q_0, w] \overset{*}{\vdash} [q_i, \lambda]$ com $q_i \in F\}$.

Autômatos Finitos Não-Determinísticos

- Máquinas não-determinísticas geralmente são construídas para usar uma estratégia de busca.
- Computação aonde há mais de uma transição possível $\Rightarrow$ *branch*: máquina "se multiplica" e cada uma segue por um caminho.
- Isso é razoável?
- DFAs são claramente implementáveis em HW ou SW.
- E os NFAs?
- Podem ser simulados por multiprocessamento.
- "Máquina se multiplica" $\Rightarrow$ cria novos processos.

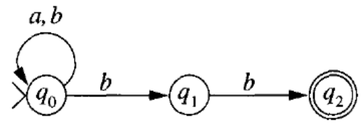
Exemplo 5.4.1 (Sudkamp)

O NFA M definido abaixo aceita a linguagem $L(M) = (\mathbf{a} \cup \mathbf{b})^*\mathbf{bb}$. Elementos de M:

$$Q = \{q_0, q_1, q_2\} \qquad \Sigma = \{a, b\} \qquad F = \{q_2\}.$$

δ	a	b
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

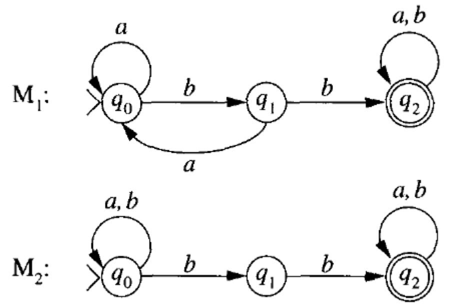
Diagrama de estados de M:



Autômatos Finitos Não-Determinísticos

Exemplo 5.4.2 (Sudkamp)

As máquinas M_1 e M_2 abaixo aceitam $(\mathbf{a} \cup \mathbf{b})^*\mathbf{bb}(\mathbf{a} \cup \mathbf{b})^*$:

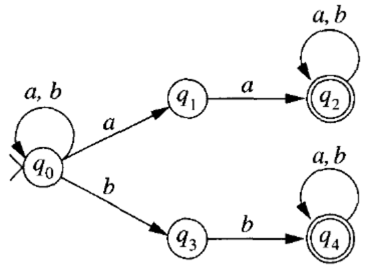


M_1 é o DFA do Exemplo 5.3.1. M_2 é um NFA equivalente.

Autômatos Finitos Não-Determinísticos

Exemplo 5.4.3 (Sudkamp)

Qual é a linguagem aceita pelo NFA M abaixo?



M aceita strings sobre $\{a, b\}$ contendo a substring *aa* ou *bb*. Formalmente:

$$L(M) = (\mathbf{a} \cup \mathbf{b})^*(\mathbf{aa} \cup \mathbf{bb})(\mathbf{a} \cup \mathbf{b})^*.$$

Transições-λ

- Transições em DFAs e NFAs requerem a leitura de um símbolo da entrada.
- Relaxamento para NFA: transições que não mexem na entrada $\Rightarrow$ transições-λ.
- NFA-λ: classe de máquinas não-determinísticas que possuem transições-λ.
- Por que fazer isso?
- Transições-λ facilitam o projeto de máquinas que aceitam linguagens complexas.

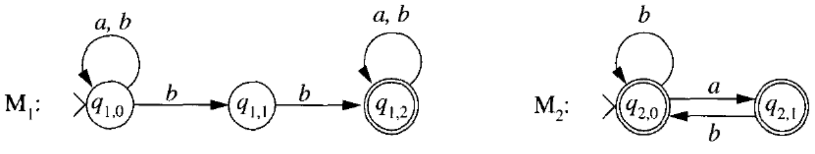
Transições-λ

- Função de transição de um NFA-λ precisa ser ajustada.
- Formalmente, $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$ (Def. 5.5.1).
- É possível que as computações em NFA-λ continuem após a entrada ser totalmente processada!
- Critério de aceite é o mesmo de um NFA.
- Exclui caso patológico acima.
- Possibilidade de mudança de estado sem consumir a entrada permite uma construção incremental de máquinas.

Transições-λ

Exemplo 5.5.1 (Sudkamp)

Sejam M_1 e M_2 dois NFAs como abaixo.



Temos:

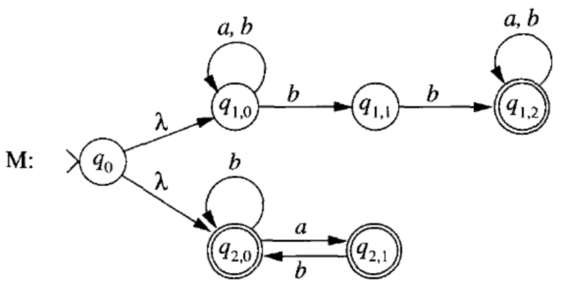
$$\begin{aligned} L(M_1) &= (\mathbf{a} \cup \mathbf{b})^*\mathbf{bb}(\mathbf{a} \cup \mathbf{b})^* \\ L(M_2) &= (\mathbf{b} \cup \mathbf{ab})^*(\mathbf{a} \cup \lambda). \end{aligned}$$

Podemos combinar a duas máquinas em uma só usando transições-λ.

Transições-λ

Exemplo 5.5.1 (cont.)

A linguagem do NFA-λ M abaixo é $L(M_1) \cup L(M_2)$.

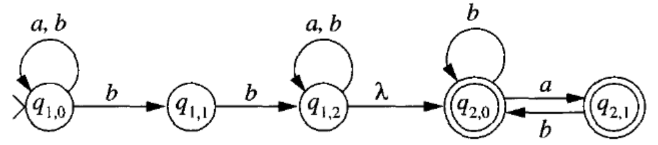


Construir um DFA equivalente a M é bem mais complexo (Exemplo 5.3.3).

Transições-λ

Exemplo 5.5.2 (Sudkamp)

A linguagem do NFA-λ abaixo é $L(M_1)L(M_2)$.

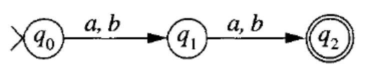


Ou seja, basta conectar as duas máquinas por uma transição-λ.

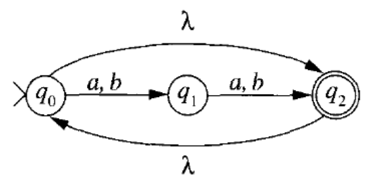
Transições-λ

Exemplo 5.5.3 (Sudkamp)

Autômato abaixo aceita strings de comprimento dois.



Autômato que aceita qualquer string de comprimento par.



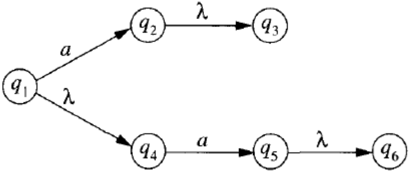
Removendo Não-Determinismo

- Classes de máquinas vistas até agora: DFA, NFA, NFA-λ.
- Cada uma é uma generalização da anterior.
- Pergunta chave: Ao permitir não-determinismo nós criamos uma classe mais poderosa de máquinas?
- Isto é, existe uma linguagem aceita por um NFA que não é aceita por nenhum DFA?
- A resposta é não!
- Existe um algoritmo genérico que converte qualquer NFA-λ em um DFA equivalente.

- Transições em DFAs e NFAs causam o processamento de um símbolo de entrada.
- Como fazer com os NFA-λ?
- $\Rightarrow$ Criar uma função de transição da entrada t.
- Valores de t são os estados alcançáveis através de uma transição normal e zero ou mais transições-λ.
- Exemplo. Para o diagrama abaixo:

$$t(q_1, a) = \{q_2, q_3, q_5, q_6\}$$

Path	String Processed
q_1, q_2	a
q_1, q_2, q_3	a
q_1, q_4	λ
q_1, q_4, q_5	a
q_1, q_4, q_5, q_0	a



Removendo Não-Determinismo

- Definição da função $t(q_i, a)$ pode ser dividida em três partes.
  1. Construir o conjunto de estados alcançáveis a partir de q_i sem processar um símbolo da entrada.
  2. Processar um a para todos os estados do item 1.
  3. Seguir arcos-λ partindo dos estados do item 2.
- Para realizar as operações acima, é necessário conhecer os caminhos no diagrama que geram a string nula.
- O fecho-λ de um estado q_i é composto por todos os estados alcançáveis a partir de q_i através de apenas transições-λ.
- Notação: λ-*closure*(q_i). (Definição 5.6.1 – implementável como um algoritmo simples).

Removendo Não-Determinismo

Definição 5.6.2 (Sudkamp)

Dado um NFA-λ M com função de transição δ, a função de transição da entrada t : $Q \times \Sigma \rightarrow \mathcal{P}(Q)$ é definida como abaixo.

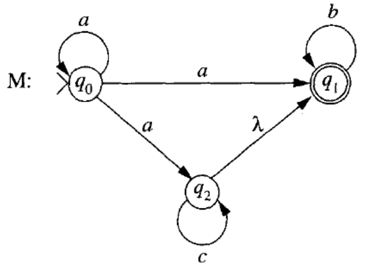
$$t(q_i, a) = \bigcup_{q_j \in \lambda\text{-}closure(q_i)} \lambda\text{-}closure(\delta(q_j, a))$$

- Agora podemos tratar NFA-λ como NFAs normais usando a função t.
- Para qualquer NFA sem transições-λ temos que $t = \delta$.
- Usamos a função t para construir um DFA equivalente.

Removendo Não-Determinismo

Exemplo 5.6.1 (Sudkamp)

O NFA-λ abaixo aceita a linguagem $\mathbf{a}^{+}\mathbf{c}^{*}\mathbf{b}^{*}$. As funções δ e t são apresentadas nas tabelas.



δ	a	b	c	λ
q_0	$\{q_0, q_1, q_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1\}$

t	a	b	c
q_0	$\{q_0, q_1, q_2\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_1\}$	$\{q_1, q_2\}$

Removendo Não-Determinismo

- Aceite em um NFA: existe (ao menos) uma computação que processa toda a entrada e para em um estado final.
- Em um NFA-λ: podem haver vários caminhos para o processamento da entrada.
- Em um DFA: há exatamente um caminho para cada entrada.
- $\Rightarrow$ Remover não-determinismo requer que o DFA simule as explorações simultâneas de todas as possíveis computações no NFA-λ.
- Isso pode ser feito de forma algorítmica (Algoritmo 5.6.3).

Removendo Não-Determinismo

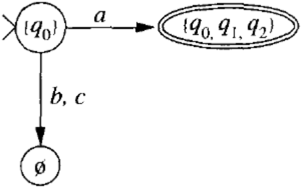
Algorithm 6.6.3
Construction of DM, a DFA Equivalent to NFA-λ M

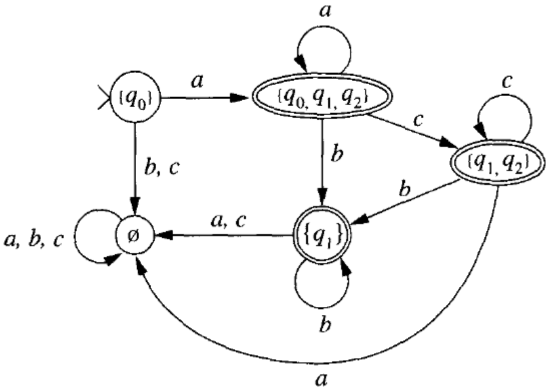
input: an NFA-λ $M = (Q, \Sigma, \delta, q_0, F)$
    input transition function t of M

1. initialize Q' to $\{\lambda\text{-}closure(q_0)\}$
2. **repeat**
    2.1. **if** there is a node $X \in Q'$ and a symbol $a \in \Sigma$ with no arc leaving X labeled a **then**
        2.1.1. let $Y = \bigcup_{q_i \in X} t(q_i, a)$
        2.1.2. **if** $Y \notin Q'$ **then** set $Q' := Q' \cup \{Y\}$
        2.1.3. add an arc from X to Y labeled a
    else done $:= true$
   until done
3. the set of accepting states of DM is $F' = \{X \in Q' \mid X \text{ contains an element } q_i \in F\}$

Prova de correção do algoritmo acima: Teorema 5.6.4 e Corolário 5.6.5.

Removendo não-determinismo do Exemplo 5.6.1.

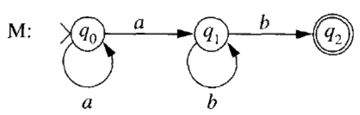


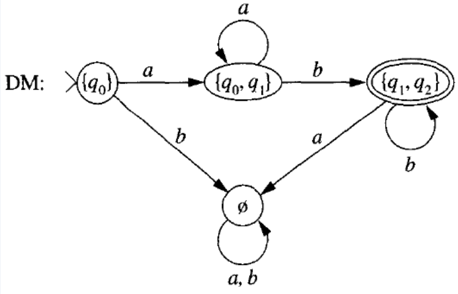


Removendo Não-Determinismo

Exemplo 5.6.2 (Sudkamp)

O NFA M abaixo aceita a linguagem $\mathbf{a}^+\mathbf{b}^+$. O DFA DM equivalente é dado a seguir.

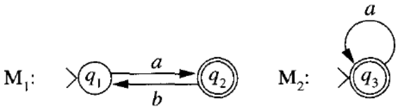


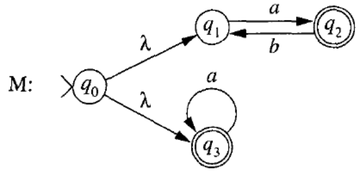


Removendo Não-Determinismo

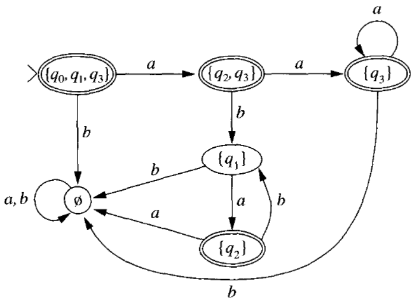
Exemplo 5.6.4 (Sudkamp)

As máquinas M$_1$ e M$_2$ aceitam **a**(**ba**)* e **a***, respectivamente.





Juntar e determinizar:



Removendo Não-Determinismo

- Algoritmo 5.6.3 (*subset construction*) é simples de implementar.
- Sempre termina: cria no máximo $card(\mathcal{P}(Q))$ estados.
- Limite do número de repetições: $card(\mathcal{P}(Q))card(\Sigma)$.
- Explosão combinatória: se NFA tem n estados o DFA pode ter 2^n!
- Pode de fato acontecer na prática: vide Exemplo 5.6.3 no livro do Sudkamp.
- Próximo passo é a minimização do DFA: algoritmo de Hopcroft.
- Fora do escopo dessa disciplina.

Aula 00 – Autômatos Finitos

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação