Introdução à Teoria da Computação

Márcio Moretto Ribeiro

18 de Outubro de 2016

Conteúdo

Capítulo 1

Introdução

1.1 Apresentação

O curso de Teoria da computação procura responder duas perguntas centrais da área de Ciência da Computação:

1. Que problemas são resolvíveis de forma automática? (computabilidade)
2. Que problemas são resolvíveis de maneita eficiente? (complexidade)

Para responder à primeira pergunta primeiro devemos esclarecer alguns pontos:

- O que estamos chamando de *problema* no sentido computacional do termo?
- O que é um *método automático* de resolução?

Extencionalmente, um *problema computacional* é uma espécie de função que descreve os valores aceitos como *entrada* (domínio) e os valores esperados como *saída*. A solução de um problema computacional, como estamos acostumados, é uma sequência de instruções inequívocas – um algoritmo – que dado um elemento válido de entrada produz a saída esperada.

Exemplo 1.1.1:

O *problema da ordenação* pode ser descrito da seguinte maneira:

Entrada: Uma sequência de n inteiros $\langle a_1, \ldots, a_n \rangle$.

Saída: Uma permutação da entrada $\langle a'_1, \ldots, a'_n \rangle$ em que $a_i \leq a_j$ para todo $i < j$.

Uma *instância* desse problema seria dada por uma entrada específica (e.g. $\langle 4, 2, 42, 24 \rangle$) e sua saída ($\langle 2, 4, 24, 42 \rangle$). E a solução do problema é qualquer dos algoritmos de ordenação que estudamos no curso de análise de algoritmos.

Nosso foco neste curso será nos problemas ditos de *decisão*, ou seja, naqueles cuja saída deve ser `SIM` ou `NÃO`

Exemplo 1.1.2:

O seguinte é um problema clássico de decisão:

Entrada: Um inteiro $n > 1$.

Saída: `SIM` se `n` é primo e `NÃO` caso contrário.

Como veremos na próxima seção existe uma relação íntima entre problemas de decisão e linguagens formais, a saber, para cada problema de decisão existe uma linguagem formal equivalente e vice-versa. De fato, grande parte do curso estudaremos classes de linguagens formais.

Voltemos então para a pergunta: o que é um método automático de resolução?

Para responder a essa pergunta, que não tem nada de trivial, precisamos de um *modelo e computação* do funcionamento de dispositivos eletrônicos. Como qualquer outro modelo, faremos abstrações, simplificaremos e desprezaremos variáveis.

Podemos refrasear nossa pergunta, de maneira agora um pouco mais precisa:

- Que linguagens são reconhecíveis por certo modelo de computação?

Durante o curso estudaremos uma sequência de modelos de computação com *expressividade* crescente. Começaremos com modelos simples, adequados para representar dispositivos simples e capazes de reconhecer uma certa classe de linguagens. Conforme avançarmos no curso estudaremos modelos mais sofisticados capazes de representar classes cada vez maiores de linguagens.

O diagrama abaixo apresenta um resumo dos três primeiros capítulos do livro. No Capítulo **??** estudaremos Autômatos Finitos Determinísticos

e não-Determinísticos, mostraremos que eles são equivalentes e são capazes de reconhecer exatamente a classe das linguagens regulares. No Capítulo **??** começaremos mostrando que nem toda linguagem é regular, estudaremos as Linguagens Livres de Contexto e os Autômatos de Pilha e mostraremos a equivalência entre os dois. No Capítulo **??** veremos Máquinas de Turing Determinísticas e não-Determinísticas, Linguagens Recursivas e Recursivamente Enumeráveis.

Modelos de Computação		**Classes de Linguagens**	
AFD AFN	$\Leftrightarrow$	Linguagens Regulares	*expressividade* $\downarrow$
AP	$\Leftrightarrow$	Linguagens Livres de Contexto	
MT	$\Leftrightarrow$	Linguagens Recursivas Ling. Recursivamente Enumeráveis	

Tabela 1.1: Resumo da apostila

As Máquinas de Turing (MTs) são o modelo mais completo que veremos. Na verdade a Tese de Church afirma que de fato, as MTs são o modelo mais completo possível. Em outras palavras: se algo pode ser computado, então ele pode ser computado por uma MT.

Por outro lado, veremos que existem linguagens que não são reconhecíveis por MTs, ou equivalentemente, existem problemas de decisão que não admitem solução computacional (*indecidíveis*).

$$Regulares \subset LLCs \subset Recursivas \subset REs \subset Linguagens$$

Para finalizar o curso, no último capítulo nos voltaremos para a segunda pergunta central:

- Que problemas podem ser resolvidos de maneira eficiente?

A resposta dessa pergunta certamente depende do modelo de computação que estamos considerando e o que entendemos por *eficiência*. Vamos convencionar que estamos falando de MTs e que por eficientes queremos dizer soluções que consomem tempo polinomial em relação ao tamanho da entrada. Assim, podemos refrasear a questão central da seguinte forma:

- Para quais problemas existe uma MT que o resolve em tempo polinomial?

Chamaremos essa classe de problemas de P. Note que, diferente do curso de análise de algoritmos em que o foco está nos algoritmos e sua eficiência, aqui o foco está nos problemas. A pergunta não é quão eficiente é uma determinada solução de um problema, mas que problemas estão em cada classe.

Se substituirmos as MT Determinísticas por não-Determinísticas, teremos outra classe de problemas, a classe NP.

O maior problema em aberto hoje na computação, e quiçá na matemática, é exatamente saber se essas classes coincidem:

$$P \stackrel{?}{=} NP$$

No fim do curso procuraremos definir o problema de maneira formal e apresentar os poucos resultados mais simples sobre esse assunto.

1.2 Problemas de Decisão

Vamos começar estudando um problema de decisão de grande importância na Teoria da Computação por se tratar do primeiro problema demonstradamente NP-completo (veremos sobre isso no capítulo **??**). O problema que nos referimos é o de decidir se uma fórmula proposicional é ou não satisfatível. Recordemos do curso de Matemática Discreta que uma fórmula proposicional sempre pode ser escrita na Forma Normal Conjuntiva (FNC), ou seja, como uma conjunção de disjunção de literais. Vamos então definir a linguagens das fórmulas supondo que ela esteja na FNC para facilitar.

Partimos de um conjuntos finito cujos elementos são chamados *variáveis proposicionais* $\mathbb{P} = \{p_1 \dots p_n\}$.

Um *literal* é qualquer elemento de $\mathbb{L} = \mathbb{P} \cup \overline{\mathbb{P}}$ onde $\overline{\mathbb{P}} = \{\bar{p_i} : p_i \in \mathbb{P}\}$. Ou seja, um literal é uma variável ou sua negação que representamos pelo mesmo símbolo com um traço em cima.

Uma sequência de literais é chamado de *cláusula* e a representaremos como $l_1 l_2 \dots l_n$ onde $l_i \in \mathbb{L}$.

Exemplo 1.2.1:

Seja $\mathbb{P} = \{p_1, p_2, p_3\}$ então as seguintes são cláusulas:

$p_1 p_2 p_3 \bar{p_1}$

$p_2 \bar{p_2} p_1$

p_1

$p_1 p_2 p_3$

Uma sequência de cláusulas e chamada de *fórmula* e será representada como $c_1; c_2; \ldots; c_n$.

Exemplo 1.2.2:

Seja $\mathbb{P} = \{p_1, p_2, p_3\}$ então as seguintes são fórmulas:

$p_1 p_2; p_2 p_3; \bar{p_3}$

$p_1 \bar{p_1}; p_2 \bar{p_2}$

$p_1 p_2 p_3$

Interpretaremos uma sequência de literais de maneira disjuntiva e uma sequência de cláusulas de maneira conjuntiva da seguinte forma. Uma função $v : \mathbb{L} \to \{0, 1\}$ é uma *valoração* se para todo $p \in \mathbb{L}$ temos:

$$\begin{array}{lll} v(p) = 1 & \text{sse} & v(\bar{p}) = 0 \\ v(p) = 0 & \text{sse} & v(\bar{p}) = 1 \end{array}$$

Dizemos que uma valoração v *satisfaz uma cláusula* $l_1 \ldots l_n$ se $v(l_i) = 1$ para algum i. Em outras palavras, a valoração satisfaz a cláusula se ela atribuiu o valor verdade para algum literal da cláusula. Uma valoração v *satisfaz uma fórmula* $c_1; \ldots; c_n$ ela satisfaz cada uma das cláusulas da fórmula.

Exemplo 1.2.3:

Seja $\mathbb{P} = \{p_1, p_2, p3\}$. A valoração v tal que $v(p_1) = v(p_2) = 1$ e $v(p_3) = 0$ satisfaz as seguintes fórmulas:

$p_1 \bar{p_2}; p_2; p_3 p_1$

$p_1 \bar{p_1}; p_2 \bar{p_2}$

$\bar{p_3}$

Por outro lado, v não satisfaz as seguintes fórmulas:

p_3

$p_3; p_1p_2; p_1$

$p_3p_1; \bar{p_1}\bar{p_2}$

Uma fórmula é dita *satisfatível* se existe uma valoração v que a satisfaça.

Exemplo 1.2.4:

São exemplos de fórmulas satisfatíveis:

$p_1p_2; \bar{p_1}\bar{p_2}$

p_1p_2

p_1

São exemplos de fórmulas *não* satisfatíveis:

$p_1; \bar{p_1}$

$p_1\bar{p_2}; p_2; \bar{p_1}$

O problema da satisfatibilidade, ou simplesmente SAT, é um problema de decisão que pode ser enunciado da seguinte forma:

Entrada: Uma fórmula α qualquer sobre $\mathbb{P}$.

Saída: `SIM` se α é satisfatível e `NÃO` caso contrário.

1.3 Linguagens Formais

Um *alfabeto* é um conjunto finito qualquer Σ cujos elementos são chamados *símbolos*.

Exemplo 1.3.1:

São exemplos de alfabeto:

$\Sigma = \{p_1, p_2, p_3\}$

$\overline{\Sigma} = \{\bar{p_1}, \bar{p_2}, \bar{p_3}\}$

$\Sigma \cup \overline{\Sigma} = \{p_1, p_2, p_3, \bar{p_1}, \bar{p_2}, \bar{p_3}\}$

Uma sequência de símbolos de um alfabeto Σ é chamada de uma *string* ou *palavra* sobre esse alfabeto. A string vazia, que representa a sequencia de zero símbolos, será representa por ε. O comprimento de uma string s é representado por $|s|$.

Exemplo 1.3.2:

São string sobre $\Sigma = \{0, 1\}$ os seguintes:

01110

11

1

ε

Além disso temos que:

$|01110| = 5$

$|11| = 2$

$|1| = 1$

$|\varepsilon| = 0$

Sejam $x = a_1 \dots a_n$ e $y = b_1 \dots b_m$ duas strings. A *concatenação* de x com y será representada por $x \cdot y = xy = a_1 \dots a_n b_1 \dots b_n$. Note que nem sempre $x \cdot y = y \cdot x$. Além disso, para todo x temos que $\varepsilon \cdot x = x \cdot \varepsilon = x$.

O conjunto de todas as strings sobre um alfabeto Σ será representada por Σ^*. Um conjunto de strings A sobre Σ é chamado de uma *linguagem sobre* Σ i.e. $A \subseteq \Sigma^*$ é uma linguagem.

Exemplo 1.3.3:

São linguagens sobre $\Sigma = \{p_1, p_2\}$:

$A = \{p_1, p_2, p_1p_2, p_1p_1\}$

$B = \{p_1\}$

$C = \emptyset$

$D = \{\varepsilon, p_1, p_1p_1, p_1p_1p_1, \dots\}$

Note que, como no último exemplo, uma linguagem pode ser infinita. Existe um problema de decisão naturalmente associado a cada linguagem L, o problema do reconhecimento:

Entrada: $x \in \Sigma^*$

Saída: `SIM` se $x \in L$ e `NÃO` caso contrário.

Conversamente, todo problema de decisão possio uma linguagem formal naturalmente associada da seguinte forma. Seja A a linguagem das entradas aceitas como válidas para o problema. Considere agora todas as strings x para as quais o problema de decisão deve responder `SIM`. O conjunto dessas strings é a linguagem associada ao problema.

Exemplo 1.3.4:

O problema SAT induz uma linguagem formal $A \subseteq \Sigma^*$ aonde $\Sigma = \{p_1, \ldots, p_n, \bar{p_1}, \ldots, \bar{p_n}\}$. A linguagem das fórmulas satisfatíveis.

1.4 Bibliografia

- Introdução à Teoria da Computação - Michael Sipser
- Elementos da Teoria da Computação - Lewis e Papadimitrius
- Computabilidade, Funções Computáveis, Lógica e os Fundamentos da Matemática - Carnielli e Epstein
- Computational Complexity - Christos H. Papadimitriou

Capítulo 2

Linguagens Regulares

Neste capítulo estudaremos um modelo simples de computação, os autômatos finitos, e a classe das linguagens regulares.

2.1 Introdução

Voltaremos nossa atenção um instante para conjuntos (classes) de linguagens, ou seja, conjuntos de conjuntos de strings.

Exemplo 2.1.1:

$\mathscr{L} = \{A : A \subseteq \Sigma^*\}$ é o conjunto de todas as linguagens sobre Σ

$\emptyset$ é a classe vazia.

$\{\emptyset\}$ é a classe que contém apenas a linguagem vazia.

$\{\{\varepsilon\}, \emptyset\}$ é a classe que contém a linguagem vazia e a linguagem que possui apenas a string vazia.

Podemos aplicar operações sobre linguagens. Como linguagens são conjuntos de strings, podemos tomar a *união* de duas linguagens:

$$A \cup B = \{x \in \Sigma^* : x \in A \text{ ou } x \in B\}$$

Exemplo 2.1.2:

$$\begin{aligned} A &= \{p_1p_2, p_1, p_2p_1\} \\ B &= \{p_1p_1, p_3, p_1\} \\ A \cup B &= \{p_1p_2, p_1, p_2p_1, p_1p_1, p_3\} \end{aligned}$$

Outra operação sobre linguagens é *concatenação* que consiste na concatenação de cada combinação de strings da linguagem:

$$A \circ B = \{x \cdot y \in \Sigma^* : x \in A \text{ e } x \in B\}$$

Exemplo 2.1.3:

$$\begin{aligned} A &= \{p_1p_2, p_1\} \\ B &= \{p_1p_1, p_3\} \\ A \circ B &= \{p_1p_2p_1p_1, p_1p_2p_3, p_1p_1p_1, p_1p_3\} \\ B \circ A &= \{p_1p_1p_1p_2, p_1p_1p_1, p_3p_1p_2, p_3p_1\} \\ A \circ A &= \{p_1p_2p_1p_2, p_1p_2p_1, p_1p_1p_2, p_1p_1\} \end{aligned}$$

Podemos, por fim, aplica a *estrela de Kleene* sobre uma linguagem para produzir todas as possíveis concatenações dos elementos:

$$A^* = \{x_1 \dots x_k \in \Sigma^* : x_i \in A\}$$

Exemplo 2.1.4:

$$\begin{aligned} A &= \{a, b\} \\ A^* &= \{\varepsilon, a, b, aa, ab, bb, ba, aaa, aab, aba, abb, \dots\} \end{aligned}$$

Repare que a notação Σ^* é consistente com a definição de estrela de Kleene.

As operações de união, concatenação e estrela de Kleene sobre linguagens são chamadas *operações regulares*.

Exemplo 2.1.5:

$$\begin{aligned} A &= \{a\} \\ B &= \{aa, b\} \\ A \circ B &= \{aaa, ab\} \\ A \cup (A \circ B) &= \{a, aaa, ab\} \\ (A \cup (A \circ B))^* &= \{\varepsilon, a, aaa, ab, aa, aaaa, aab, aaaaaa, aaaab, \dots\} \end{aligned}$$

Uma classe de linguagens $\mathscr{L}$ é *fechada por união* quando temos que:

$$\text{se } A, B \in \mathscr{L} \text{ então } A \cup B \in \mathscr{L}$$

Analogamente, uma classe de linguagens $\mathscr{L}$ é *fechada por concatenação* quando temos que:

$$\text{se } A, B \in \mathscr{L} \text{ então } A \circ B \in \mathscr{L}$$

Por fim, $\mathscr{L}$ é *fechada pela estrela de Kleene* quando temos que:

$$\text{se } A \in \mathscr{L} \text{ então } A^* \in \mathscr{L}$$

Exemplo 2.1.6:

$$\begin{aligned} \mathscr{L}_1 &= \{\{a\}, \{b\}\} \\ \mathscr{L}_2 &= \{\{a\}, \{b\}, \{a, b\}\} \\ \mathscr{L}_3 &= \{\{a\}, \{a, aa\}, \{a, aa, aaa\} \dots\} \\ \mathscr{L}_4 &= \{\{a\}, \{\varepsilon, a, aa, aaa, \dots\}\} \end{aligned}$$

$\mathscr{L}_1$ não é fechada por união, mas $\mathscr{L}_2$ é. $\mathscr{L}_3$ é fechada por concatenação e $\mathscr{L}_4$ é fechada pela estrela de Kleene.

A classe das *linguagens regulares* é a menor classe de linguagens fechada por união, concatenação e estrela de Kleene que contém a seguinte linguagem:

$$\{\{a\} : a \in \Sigma\}$$

Uma foram alternativa de definir linguagens regulares é por meio de expressões regulares. Uma *expressão regular* pode ser definida da seguinte forma:

se $r \in \Sigma$ então r é uma expressão regular,

ϵ é uma expressão regular,

$\emptyset$ é uma expressão regular,

se r_1 e r_2 são expressões regulares então $r_1 \cup r_2$ é uma expressão regular,

se r_1 e r_2 são expressões regulares então $r_1 r_2$ é uma expressão regular e

se r é uma expressão regular então $r^\star$ é uma expressão regular.

Exemplo 2.1.7:

São expressões regulares:

$\emptyset$

01

$01^\star \cup 1$

$\epsilon \cup \emptyset$

Denotaremos $L(r)$ a linguagem *expressa* pela expressão regular r:

$L(a) = \{a\}$ para todo $a \in \Sigma$

$L(\epsilon) = \{\varepsilon\}$

$L(\emptyset) = \emptyset$

$L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$

$L(r_1 r_2) = L(r_1) \circ L(r_2)$

$L(r^\star) = L(r)^*$

Exemplo 2.1.8:

$L(\emptyset) = \emptyset$

$L(01) = L(0) \circ L(1) = \{01\}$

$L(01^\star \cup 1) = L(01^\star) \cup L(1) = L(0) \circ L(1^\star) \cup \{1\} = \{0\} \circ \{1\}^* \cup \{1\} = \{0, \varepsilon, 1, 11, 111 \dots\}$

$L(\epsilon \cup \emptyset) = L(\epsilon) \cup L(\emptyset) = \{\varepsilon\} \cup \emptyset = \{\varepsilon\}$

2.2 Autômatos Finitos Determinísticos

Um *Autômato Finito Determinístico* (AFD) é um modelo de computação, o mais simples que estudaremos, adequado para representar sistemas computacionais simples como portas automáticas e elevadores.

Um AFD é definido formalmente como uma 5-upla $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ em que:

Q é um conjunto finito cujos elementos são chamados *estados*,

Σ é uma *alfabeto*,

$\delta : Q \times \Sigma \to Q$ é uma função de estados e símbolos em estados chamada *função de transição*,

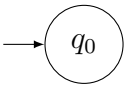
$q_0 \in Q$ é um estado chamado *inicial* e

$F \subseteq Q$ é um conjunto de estados chamados *finais*.

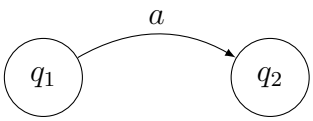
Representaremos um AFD pictoricamente por meio de um *diagrama de estados*. Nesse tipo de diagrama, cada estado $q \in Q$ é representado por uma circunferência:

Os estados finais $q \in F$ são representados por uma circunferência dupla:

O estado inicial é destacado com uma seta:



A função de transição é representada por uma seta entre os estado com uma etiqueta:



$$\delta(q_1, a) = q_2$$

Exemplo 2.2.1:

Considere o seguinte AFD:

$$\begin{aligned} M &= \langle Q, \Sigma, \delta, q_1, F \rangle \\ Q &= \{q_1, q_2, q_3\} \\ \Sigma &= \{0, 1\} \\ F &= \{q_2\} \end{aligned}$$

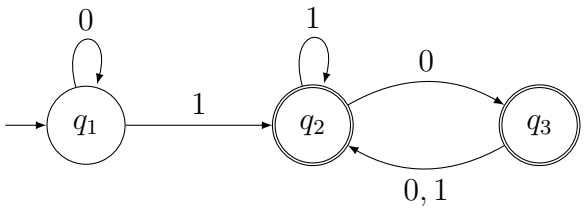
Para simplificar, normalmente escreveremos a função δ como uma tabela:

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Essa tabela indica que:

$$\begin{aligned}
\delta(q_1, 0) &= q_1 \\
\delta(q_1, 1) &= q_2 \\
\delta(q_2, 0) &= q_3 \\
\delta(q_2, 1) &= q_2 \\
\delta(q_3, 0) &= q_2 \\
\delta(q_3, 1) &= q_2
\end{aligned}$$

O seguinte diagrama de estados representa esse AFD M:



Dizemos que um AFD $M = \langle Q, \Sigma, \delta, q_o, F \rangle$ *aceita*, ou *reconhece*, uma string $\omega = a_0 a_1 \dots a_n$ se existe uma sequência de estadps $r_0, r_1, \dots, r_m$ tal que:

1. $r_0 = q_0$
2. $\delta(r_i, a_{i+1}) = r_{i+1}$
3. $r_n \in F$

Dizemos que M *consome* a string conforme passa de um estado para outro. Assim, começando pelo estado inicial, a cada passo a função de transição indica qual o próximo estado conforme consome um símbolo da string. Ao final do processo, quando todos os símbolos foram consumidos, a string é aceita se o estado atual for final.

Escrevemos $L(M)$ para a linguagem das strings aceitas por M.

$$L(M) = \{\omega \in \Sigma^* : M \text{ aceita } \omega\}$$

Exemplo 2.2.2:

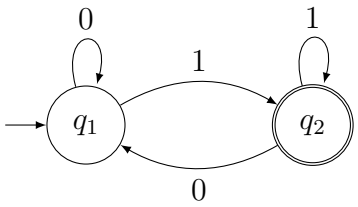
O AFD M do exemplo anterior aceita a string 1101.

1. $r_0 = q_1$
2. $r_1 = q_2$ pois $\delta(q_1, 1) = q_2$
3. $r_2 = q_2$ pois $\delta(q_2, 1) = q_2$
4. $r_3 = q_3$ pois $\delta(q_2, 0) = q_3$
5. $r_4 = q_2$ pois $\delta(q_3, 1) = q_2$
6. a string é aceita, pois $r_4 \in F$

Exemplo 2.2.3:

$$M_1 = \langle \{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\} \rangle$$

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2

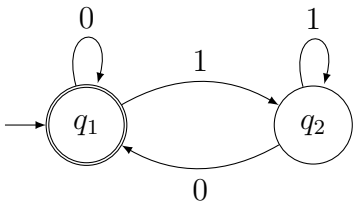


$$L(M_1) = \{\omega \in \{0, 1\}^* : \omega \text{ termina com } 1\}$$

Exemplo 2.2.4:

$$M_2 = \langle \{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_1\} \rangle$$

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2

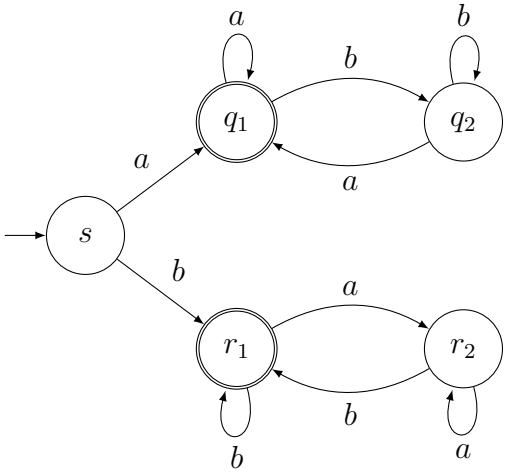


$$L(M_2) = \{\omega \in \{0,1\}^* : \omega = \varepsilon \text{ ou } \omega \text{ termina com } 0\}$$

Exemplo 2.2.5:

$$M_3 = \langle \{s, q_1, q_2, r_1, r_2\}, \{a, b\}, \delta, s, \{q_1, r_1\} \rangle$$

δ	a	b
s	q_1	r_1
q_1	q_1	q_2
q_2	q_1	q_2
r_1	r_2	r_1
r_2	r_2	r_1

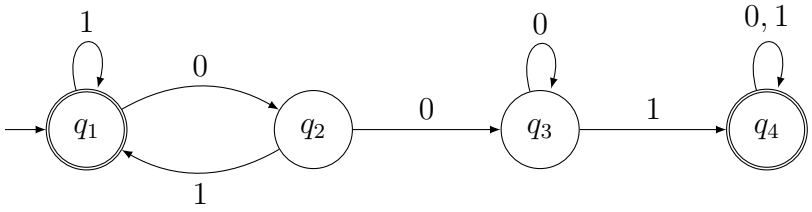


$$L(M_3) = \{\omega \in \{a,b\}^* : \omega = \text{ começa e termina com o mesmo símbolo}\}$$

Exemplo 2.2.6:

$$M_4 = \langle \{q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_1, \{q_4\} \rangle$$

δ	0	1
q_1	q_2	q_1
q_2	q_3	q_1
q_3	q_3	q_4
q_4	q_4	q_4

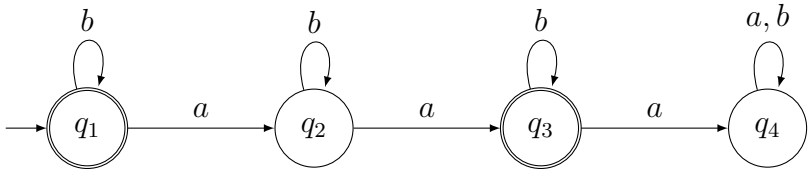


$$L(M_4) = \{\omega \in \{0, 1\}^* : \omega \text{ contém a substring } 001\}$$

Exemplo 2.2.7:

$$M_5 = \langle \{q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, q_1, \{q_3\} \rangle$$

δ	a	b
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_4	q_3
q_4	q_4	q_4

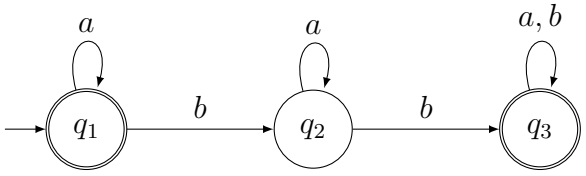


$$L(M_5) = \{\omega \in \{0,1\}^* : \omega \text{ contém exatamente dois } a\}$$

Exemplo 2.2.8:

$$M_6 = \langle \{q_1, q_2, q_3\}, \{a, b\}, \delta, q_1, \{q_3\} \rangle$$

δ	a	b
q_1	q_1	q_2
q_2	q_2	q_3
q_3	q_3	q_3



$$L(M_6) = \{\omega \in \{0,1\}^* : \omega \text{ contém pelo menos dois } b\}$$

2.3 Autômatos Finitos Não-Determinísticos

Um AFD ao ler um símbolo a em um estado q tem uma única possíbilidade de próximo estado (por isso determinístico). Na definição isso é garantido pelo fato de δ ser uma função. No diagrama de estados isso se reflete no fato de que de cada estado sai uma e uma única seta com cada símbolo do alfabeto.

Os *autômatos finitos não-determinísticos* (AFN) extendem os determinísticos em dois aspectos:

1. ao ler um símbolo em um estado o AFN possui um conjunto (possivelmente vazio) de possiblidades de próximos estados e

2. é possível mudar de estado sem consumir nenhum símbolo da entrada.

Formalmente um AFN é também definido como uma 5-upla $N = \langle Q, \Sigma, \Delta, q_0, F \rangle$, mas agora $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$. Ou seja, a entrada da função pode ser a string vazia ϵ e sua saída é um conjunto de estados.

Representamos o diagrama de estados da mesma forma que fizemos com os AFDs, mas agora é pssível que de um mesmo estado partam mais de uma seta com o mesmo síbolo, existem setas com ε e pode haver estados em que não haja seta com determinado símbolo.

Uma string é *aceita* por um AFN se existir *alguma* possibilidade de execução do autômato que consuma toda string e termine em um estado final.

Formalmente, $N = \langle Q, \Sigma, \Delta, q_0, F \rangle$ aceita uma string $\omega = y_1 \dots y_n$ onde $y_i \in \Sigma \cup \{\varepsilon\}$ se *existe* uma sequência de estados $r_0, \dots, r_n$ tal que:

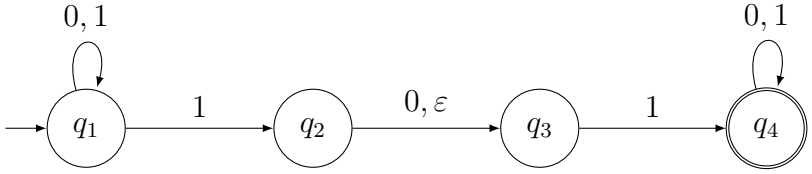
1. $r_0 = q_0$
2. $r_{i+1} \in \Delta(r_i, y_{i+1})$
3. $r_n \in F$

Novamente, escrevemos $L(N)$ para a linguagem formada pelas strings aceitas por N.

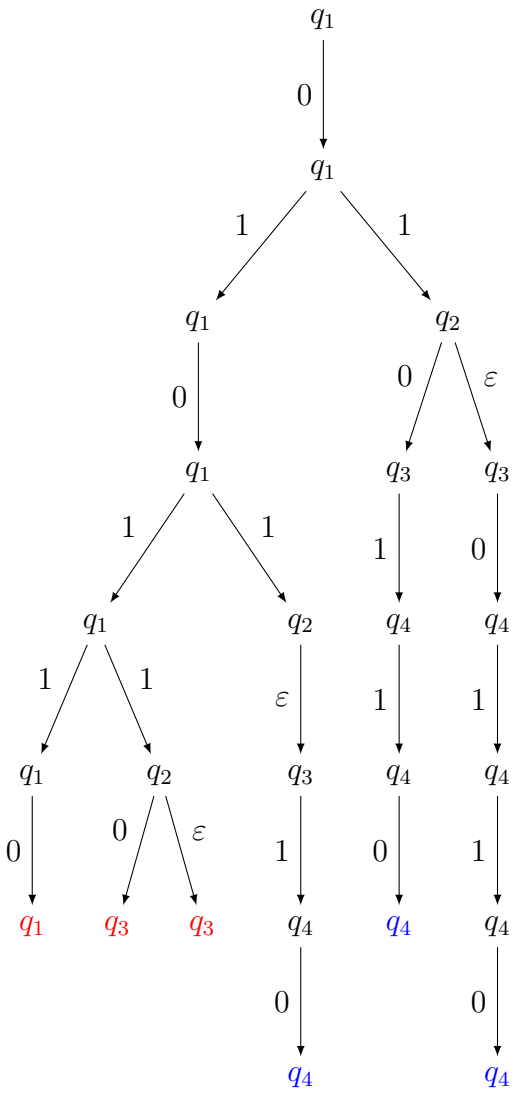
Exemplo 2.3.1:

$$N_1 = \langle \{q_1, q_2, q_3, q_4\}, \{0, 1\}, \Delta, q_1, \{q_4\} \rangle$$

Δ	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$



Vamos simular as possíveis execuções desse autômato:



Cada ramo dessa árvore representa uma possível execução do autômato para a entrada dada. O autômato para apenas quando toda a entrada foi consumida. Note que nos três ramos mais a esquerda quando isso ocorre não estamos em um estado final, mas nas três da direita sim. Basta que exita um ramo, uma possibilidade de execução, para que a string seja aceita. Assim, neste caso a string de fato é aceita, basta escolher um caminha que termine em um estado final. Por exemplo: $q_1, q_1, q_2, q_2, q_3, q_4, q_4, q_4$

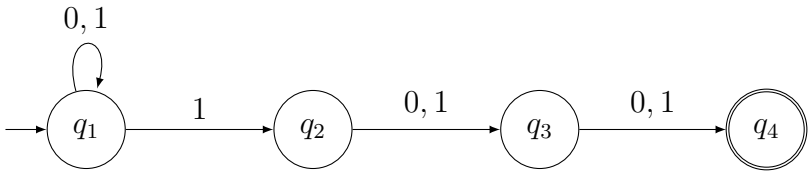
satisfaz a definição para a string $0101\varepsilon 10 = 010110$.

$$L(N_1) = \{\omega \in \{0,1\}^* : \omega \text{ contém 101 ou 11 como substring}\}$$

Exemplo 2.3.2:

$$N_2 = \langle \{q_1, q_2, q_3, q_4\}, \{0,1\}, \Delta, q_1, \{q_4\} \rangle$$

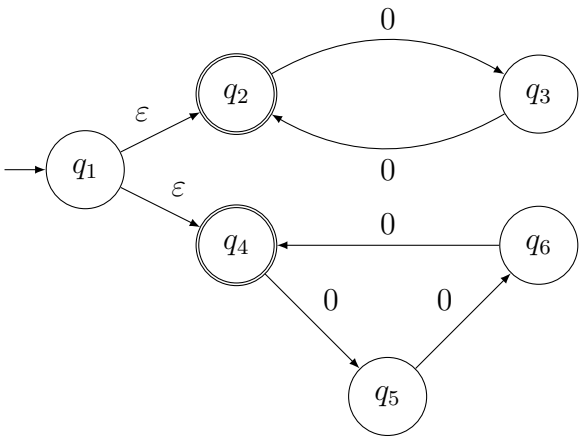
Δ	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_4\}$	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\emptyset$



$$L(N_2) = \{\omega \in \{0,1\}^* : \omega \text{ contém 1 na antepenúltima posição}\}$$

Exemplo 2.3.3:

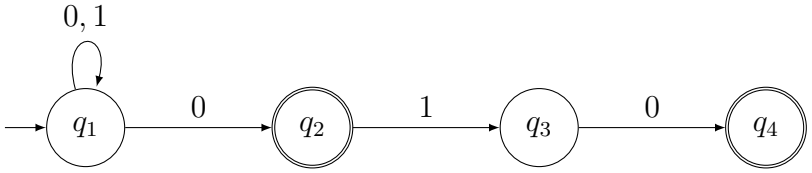
A partir daqui vamos apresentar os autômatos apenas por seu diagrama. O seguinte é o diagrama de N_3:



$$L(N_3) = \{\omega \in \{0\}^* : |\omega| \text{ é múltiplo de 2 ou de 3}\}$$

Exemplo 2.3.4:

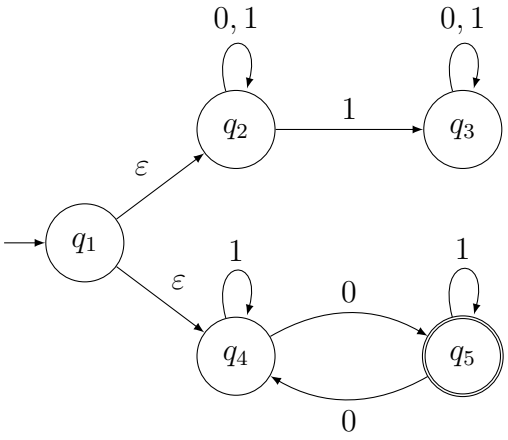
O seguinte é o diagrama de N_4:



$$L(N_4) = \{\omega \in \{0,1\}^* : |\omega| \text{ termina com } 010\}$$

Exemplo 2.3.5:

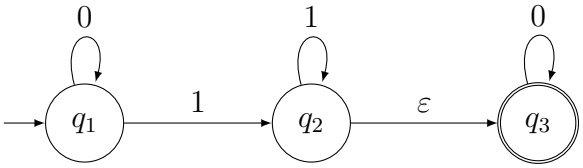
A partir daqui vamos apresentar os autômatos apenas por seu diagrama. O seguinte é o diagrama de N_3:



$$L(N_5) = \{\omega \in \{0,1\}^* : \omega \text{ contém 1 ou um número ímpar de 0s}\}$$

Exemplo 2.3.6:

O seguinte é o diagrama de N_6:



$$L(N_6) = L(0^\star 1 1^\star 0^\star)$$

2.4 AFD ≡ AFN

Vimos nas últimas seções dois modelos computacionais. O primeiro é mais próximo da descrição de dispositivos simples, mas sua descrição em termos de diagramas possui diversas limitações. O segundo modelo é mais flexível e mais fácil de descrever pictoricamente.

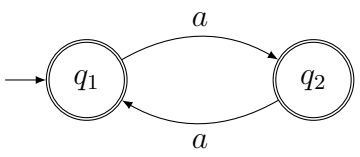
A pergunta que procuraremos responder nessa seção é se algum desses modelos é mais expressivo que o outro. Ou seja, será que algum deles é capaz de resolver problemas, reconhecer linguagens, que o outro não consegue. A resposta será negativa. De fato, ambos os modelos são equivalentes em um sentido bastante preciso. Comecemos então com essa definição.

Dois autômatos M_1 e M_2 são ditos *equivalentes* (escrevemos $M_1 \sim M_2$) se reconhecem a mesma linguagem, ou seja, se $L(M_1) = L(M_2)$.

Exemplo 2.4.1:

$$M_1 = \langle \{q_1, q_2\}, \{a\}, \delta, q_1, \{q_1, q_2\} \rangle$$

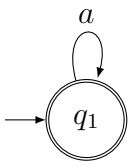
δ	a
q_1	q_2
q_2	q_1



$$L(M_1) = L(a^\star)$$

$$M_2 = \langle \{q_1\}, \{a\}, \delta, q_1, \{q_1\} \rangle$$

δ	a
q_1	q_1



$$L(M_2) = L(a^\star)$$

$$M_1 \sim M_2$$

Primeiramente devemos mostrar que AFNs são uma extensão dos AFDs. Essa parte coincide com nossa intuição uma vez que todo diagrama de um AFD é também um diagrama para um AFN (o contrário não vale!). Vamos formalizar essa ideia no seguinte teorema:

Teorema 2.4.2. *Se M é um AFD então existe um AFN N tal que $M \sim N$*

Demonstração. Seja $M = \langle Q, \Sigma, \delta, q_o, F\rangle$ um AFD qualquer. Considere o AFN $N = \langle Q, \Sigma, \Delta, q_0, F\rangle$ em que:

$$\Delta(a, q) = \begin{cases} \{q'\} & \text{se } a \in \Sigma \text{ e } \delta(a, q) = q' \\ \emptyset & \text{se } a = \epsilon \end{cases}$$

Note que os diagramas de M e de N são idênticos e segue trivialmente que $M \sim N$ □

A demonstração da outra equivalência exige mais cuidado e faremos em duas partes. Primeiro vamos supor que não fosse permitido mudar de estado sem consumir símbolos em um AFN. Ou seja, suponhamos que não seja permitido usar ε nas setas no diagrama de estados. Vamos mostrar que é possível construir um AFD equivalente a esse AFN. A ideia da construção é que cada estado no AFD simula um conjunto de estados no AFN. Conforme consumimos a string nesse AFD o estado atual representa o conjunto de todos os estados possíveis no AFN ao consumir os mesmos símbolos.

Lema 2.4.3. *Seja N um AFN em que não é permitido mudar de estados sem consumir símbolos, então existe um AFD M tal que $N \sim M$.*

Demonstração. Não faremos a demonstração completa, apenas apresentaremos a construção e posteriormente mostraremos alguns exemplos.

Seja $N = \langle Q, \Sigma, \Delta, q_0, F\rangle$, construiremos $M = \langle Q', \Sigma', \delta, q_0', F'\rangle$ da seguinte forma:

1. $Q' = 2^Q$
2. $\Sigma' = \Sigma$

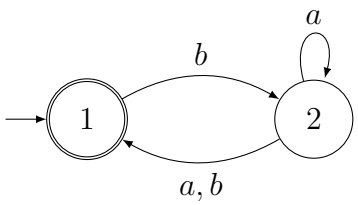
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' : R \cap F \neq \emptyset\}$
5. $\delta(R, a) = \bigcup_{r \in R} \Delta(r, a)$

□

Exemplo 2.4.4:

$$N_1 = \langle \{1, 2\}, \{a, b\}, \Delta, 1, \{1\} \rangle$$

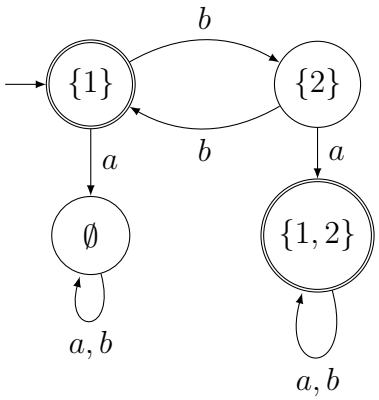
Δ	a	b
1	$\emptyset$	$\{2\}$
2	$\{1, 2\}$	$\{1\}$



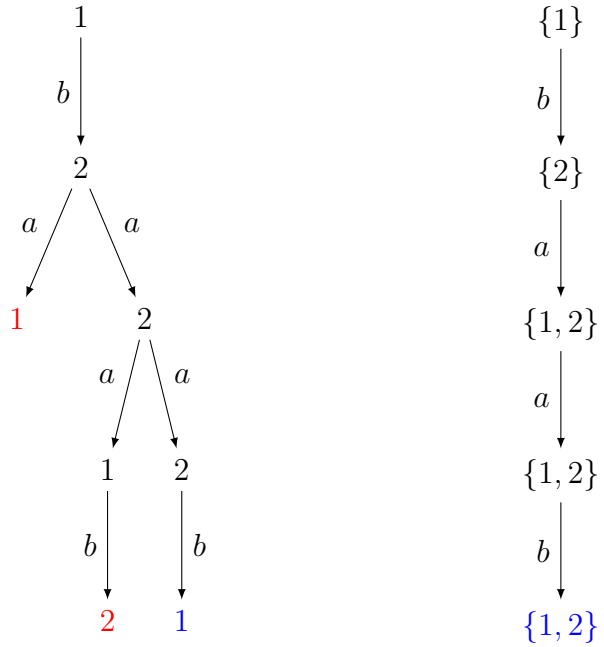
Seguindo a construção que vimos no teorema anterior:

$$M_1 = \langle \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}, \{a, b\}, \delta, \{1\}, \{\{1\}, \{1, 2\}\} \rangle$$

δ	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\emptyset$	$\{2\}$
$\{2\}$	$\{1, 2\}$	$\{1\}$
$\{1, 2\}$	$\{1, 2\}$	$\{1, 2\}$



Para terminar vamos simular nos dois a leitura da string *baab*:



Para completar nossa prova precisamos lidar com as setas rotuladas por ε. O que faremos nesse caso incluir no estado atual todos os estados que conseguimos alcançar sem precisar consumir símbolos.

Teorema 2.4.5. *Para todo AFN N existe um AFD M tal que $N \sim M$.*

Demonstração. Sejam N e M como definidos na demonstração do lema anterior Considere a seguinte função $E : Q' \to Q'$:

$$E(R) = \{q \in Q : \exists q' \in R(q' \overset{\varepsilon}{\rightsquigarrow} q)\}$$

Ou seja, existe um caminho de algum $q' \in R$ até q passando apenas por setas com etiqueta ε.

Vamos agora atualizar a definição de M em dois pontos:

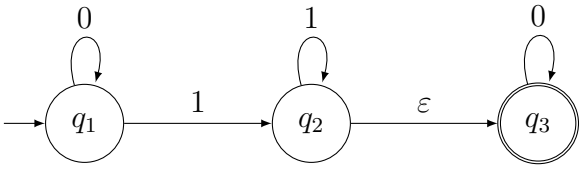
3'. $q_0 = E(\{q_0\})$

5'. $\delta(R, a) = \bigcup_{r \in R} E(\Delta(r, a))$

□

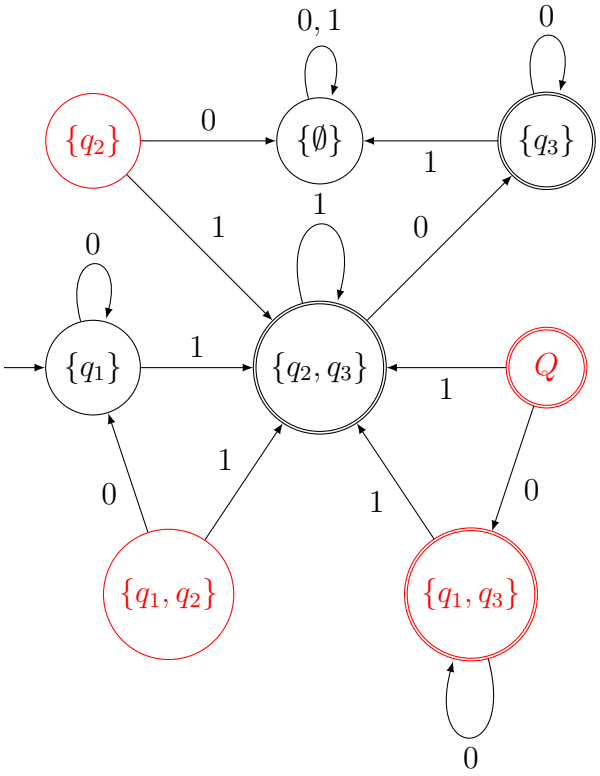
Exemplo 2.4.6:

Considere o AFN N_2 representado pelo seguinte diagrama de estados:

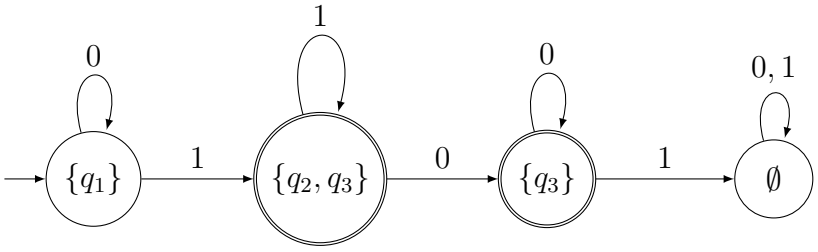


$$L(N_2) = L(0^\star 11^\star 0^\star)$$

Usando a construção dos teoremas anteriores produzimos o seguinte AFD M_2:



A construção que vimos possui sempre uma quantidade exponencialmente maior de estados, porém, alguns deles podem ser supérfluos. Note que os estados em vermelho não são alcançáveis a partir do estado inicial, logo, podem ser omitidos (formalmente, o autômato gerado ao se omitir esses estados é equivalente a esse).



Removendo os estados supérfluos é fácil ver que:

$$L(M_2) = L(N_2) = L(0^\star 11^\star 0^\star)$$

Portanto $M_2 \sim N_2$.

2.5 Linguagens Regulares são Reconhecíveis por AFNs

Na seção anterior vimos que AFDs e AFNs são capazes de reconhecer a mesma classe de linguagens. Nesta seção começaremos a investigar que essa classe é exatamente a classe das linguagens regulares.

Mostraremos nessa seção que toda linguagem regular é reconhecível por algum AFN e, consequentemente, por algum AFD. Para tanto, temos que mostrar que a classe das linguagens reconhecíveis por AFNs é fechada por $*$, $\cup$, $\circ$ e que contém $\{\{a\} : a \in \Sigma\}$.

Lema 2.5.1. *Se $A, B \subseteq \Sigma^*$ são reconhecíveis por AFNs então $A \cup B$ também é. Ou seja, a classe das linguagens reconhecíveis por AFNs é fechada por união.*

Demonstração. A hipótese garante que existem AFNs N_1 e N_2 tal que $L(N_1) = A$ e $L(N_2) = B$. Vamos construir a partir de N_1 e N_2 um AFN N tal que $L(N) = L(N_1) \cup L(N_2)$

Sejam:

$$\begin{aligned} N_1 &= \langle Q_1, \Sigma, \Delta_1, q_1, F_1 \rangle \\ N_2 &= \langle Q_2, \Sigma, \Delta_2, q_1, F_2 \rangle \end{aligned}$$

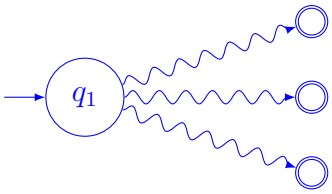
Vamos construir $N = \langle Q, \Sigma, \Delta, q_0, F \rangle$:

$$Q = Q_1 \cup Q_2 \cup \{q_0\}$$

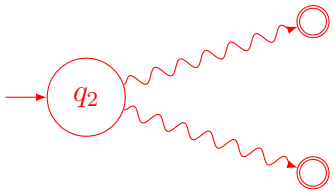
$$F = F_1 \cup F_2$$

$$\Delta(q,a) = \begin{cases} \Delta_1(q,a) & \text{se} \quad q \in Q_1 \\ \Delta_2(q,a) & \text{se} \quad q \in Q_2 \\ \{q_1, q_2\} & \text{se} \quad q = q_0 \text{ e } a = \varepsilon \\ \emptyset & \text{se} \quad q = q_0 \text{ e } a \neq \varepsilon \end{cases}$$

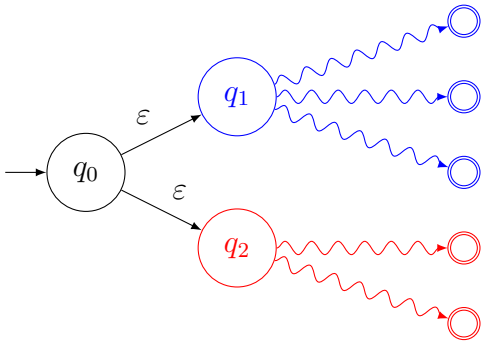
N_1



N_2



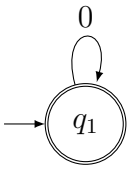
N



□

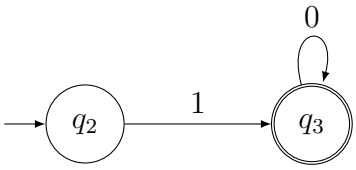
Exemplo 2.5.2:

N_1



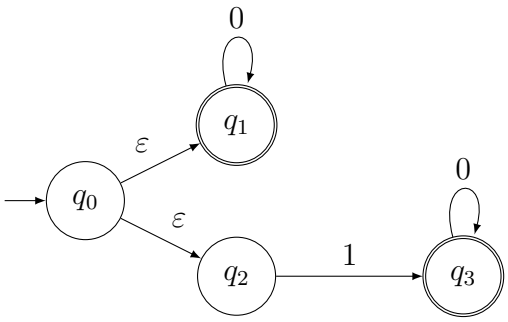
$L(N_1) = L(0^\star)$

N_2



$L(N_2) = L(10^\star)$

N



$$L(N) = L(N_1) \cup L(N_2) = L(0^\star) \cup L(10^\star) = L(0^\star \cup 10^\star)$$

Lema 2.5.3. *Se $A, B \subseteq \Sigma^*$ são reconhecíveis por AFNs então $A \circ B$ também é. Ou seja, a classe das linguagens reconhecíveis por AFNs é fechada por concatenação.*

Demonstração. A hipótese garante que existem AFNs N_1 e N_2 tal que $L(N_1) = A$ e $L(N_2) = B$. Vamos construir a partir de N_1 e N_2 um AFN N tal que $L(N) = L(N_1) \circ L(N_2)$

Sejam:

$$\begin{aligned} N_1 &= \langle Q_1, \Sigma, \Delta_1, q_1, F_1 \rangle \\ N_2 &= \langle Q_2, \Sigma, \Delta_2, q_1, F_2 \rangle \end{aligned}$$

Vamos construir $N = \langle Q, \Sigma, \Delta, q_0, F \rangle$:

$Q = Q_1 \cup Q_2$

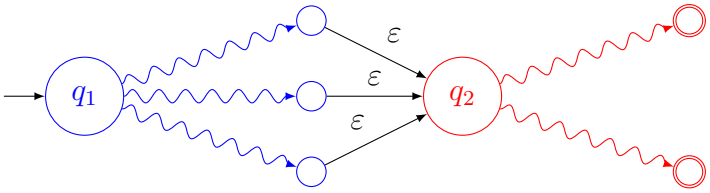
$q_0 = q_1$

$F = F_2$

$$\Delta(q,a) = \begin{cases} \Delta_1(q,a) & \text{se} \quad q \in Q_1 \text{ e } q \notin F_1 \\ \Delta_1(q,a) & \text{se} \quad q \in F_1 \text{ e } a \neq \varepsilon \\ \Delta_1(q,a) \cup \{q_2\} & \text{se} \quad q \in F_1 \text{ e } a = \varepsilon \\ \Delta_2(q,a) & \text{se} \quad q \in Q_2 \end{cases}$$

N_1 N_2

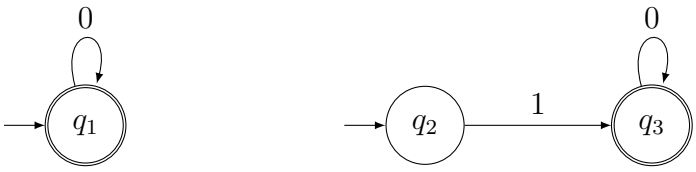
N



□

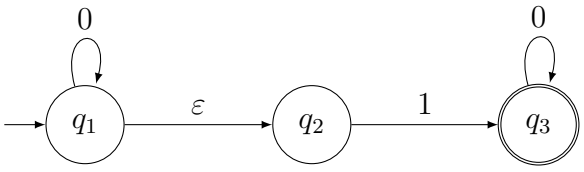
Exemplo 2.5.4:

N_1 N_2



$L(N_1) = L(0^\star)$ $L(N_2) = L(10^\star)$

N



$$L(N) = L(N_1) \circ L(N_2) = L(0^\star) \circ L(10^\star) = L(0^\star 10^\star)$$

Lema 2.5.5. *Se $A \subseteq \Sigma^*$ é reconhecível por um AFN então A^* também é. Ou seja, a classe das linguagens reconhecíveis por AFNs é fechada por estrela de Kleene.*

Demonstração. A hipótese garante que existe AFNs N_1 tal que $L(N_1) = A$. Vamos construir a partir de N_1 um AFN N tal que $L(N) = L(N_1)^*$

Seja:

$$N_1 \quad = \quad \langle Q_1, \Sigma, \Delta_1, q_1, F_1 \rangle$$

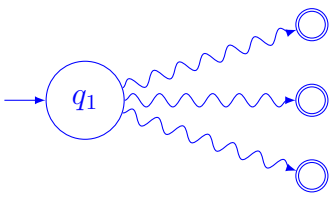
Vamos construir $N = \langle Q, \Sigma, \Delta, q_0, F \rangle$:

$Q = Q_1 \cup \{q_0\}$

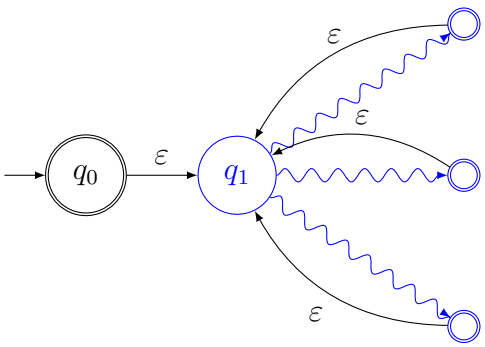
$F = F_1 \cup \{q_0\}$

$$\Delta(q,a) = \begin{cases} \Delta_1(q,a) & \text{se} \quad q \in Q_1 \text{ e } q \notin F_1 \\ \Delta_1(q,a) & \text{se} \quad q \in F_1 \text{ e } a \neq \varepsilon \\ \Delta_1(q,a) \cup \{q_1\} & \text{se} \quad q \in F_1 \text{ e } a = \varepsilon \\ \{q_1\} & \text{se} \quad q = q_0 \text{ e } a = \varepsilon \\ \emptyset & \text{se} \quad q = q_0 \text{ e } a \neq \varepsilon \end{cases}$$

N_1



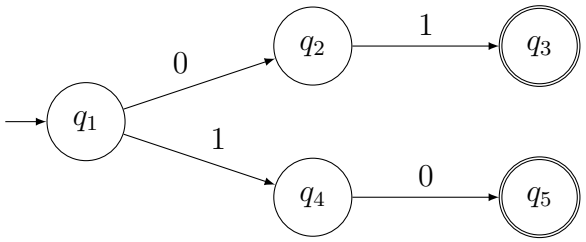
N



□

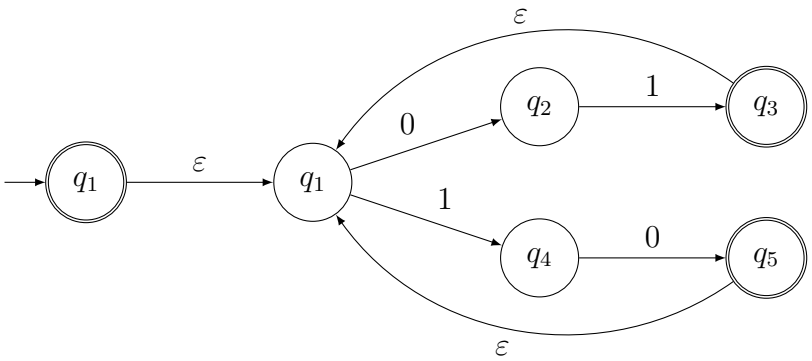
Exemplo 2.5.6:

N_1



$$L(N_1) = L(01 \cup 10)$$

N

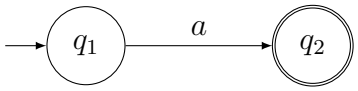


$$L(N) = L(N_1)^* = L((01 \cup 10)^\star)$$

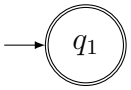
Teorema 2.5.7. *Toda linguagem regular é reconhecível por um AFN. Ou seja, a classe das linguagens reconhecíveis por AFNs contém a classe das linguagem regulares.*

Demonstração. Pelos lemas anteriores sabemos que a classe das linguagens reconhecíveis por AFNs é fechada por união, concatenação e estrela de Kleene. Para completar a prova mostraremos que a classe contém as linguagens $\{a\}$ para todo $a \in \Sigma$, $\{epsilon\}$ e $\emptyset$. Os seguintes AFNs fazem exatamente isso:

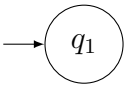
N_a



N_ε



$N_\emptyset$

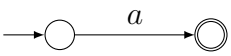


□

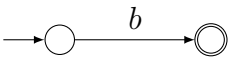
Exemplo 2.5.8:

Construiremos o autômato que reconhece $L((ab \cup a)^\star)$ usando o método visto neste capítulo:

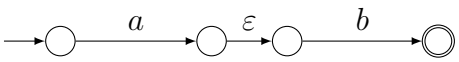
a



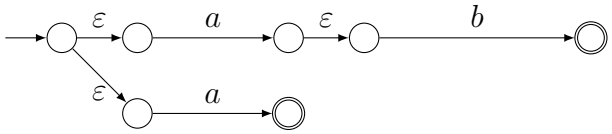
b



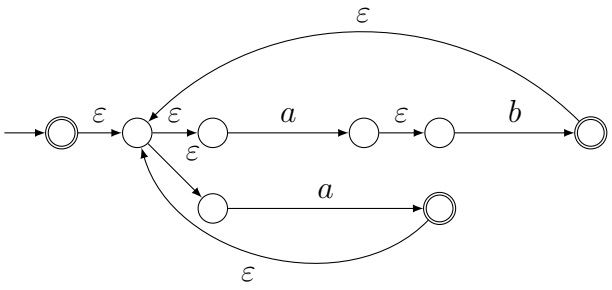
ab



$ab \cup a$



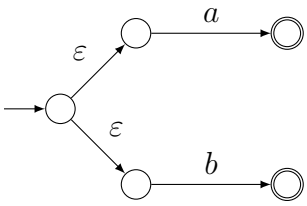
$(ab \cup a)^\star$



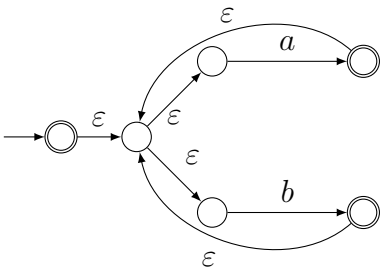
Exemplo 2.5.9:

Para concluir, construiremos o autômato que reconhece $L((a \cup b)^\star aba)$ usando o método visto neste capítulo:

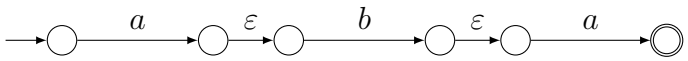
$a \cup b$



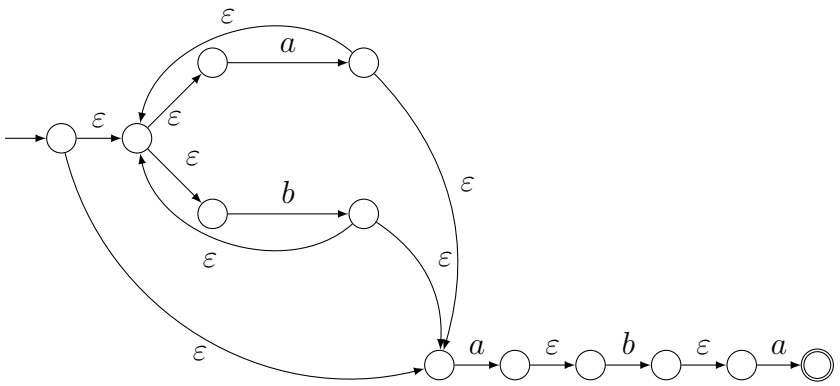
$(a \cup b)^\star$



aba



$(a \cup b)^\star aba$



2.6 Linguagens Reconhecíveis por AFNs são Regulares

Na última seção vimos que toda linguagem regular pode ser reconhecida por um autômato finito. Veremos agora a relação recíproca, a saber, que toda linguagem reconhecível por um autômato é regular. Para isso comecemos com a seguinte definição. Um *Autômato Finito Generalizado* (AFG) é uma 5-upla $\langle , \Sigma, \delta, q_i, q_f \rangle$ em que:

Q é um conjunto de estados,

Σ é um alfabeto,

$q_i \in Q$ é um estado chamado de *inicial*,

$q_f \in Q$ é um estado chamado *final* e

$\delta : (Q - \{q_f\}) \times (Q - \{q_i\}) \to R$ em que R é o conjunto das expressões regulares sobre Σ.

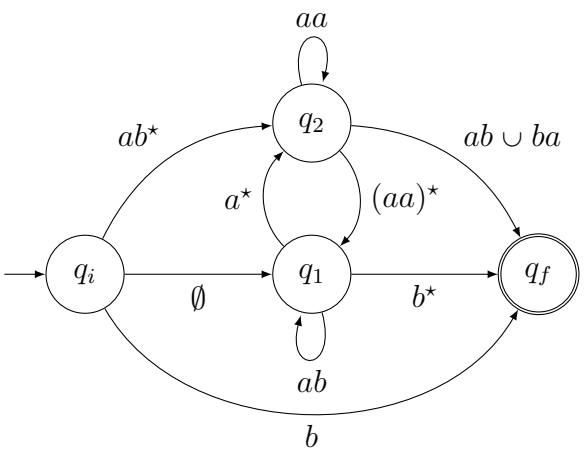
Em outras palavas, um AFG é como um AFN com um único estado final e aonde as transições são etiquetadas não com um símbolo, mas com uma expressão regular. Um AFG *aceita* uma string $\omega \in \Sigma^*$ se $\omega = \omega_1 \cdot \omega_2 \ldots \omega_k$ e existe uma sequência de estados $q_0, \ldots, q_k$ tal que:

1. $q_0 = q_i$
2. $q_k = q_f$
3. $\omega_j \in L(R_j)$ onde $R_j = \delta(q_{j-1}, q_j)$ para $0 < j \leq k$

Exemplo 2.6.1:

$$G = \langle \{q_i, q_1, q_2, q_f\}, \{a, b\}, \delta, q_i, q_f \rangle$$

δ	q_i	q_1	q_2	q_f
q_i	$\times$	$\emptyset$	$ab^\star$	b
q_1	$\times$	ab	$(aa)^\star$	$b^\star$
q_2	$\times$	$a^\star$	aa	$ab \cup ba$
q_f	$\times$	$\times$	$\times$	$\times$



$$aba, aab, abbbab, aaab, b \in L(G)$$

Omitiremos a partir de agora as setas com etiqueta $\emptyset$.

Lema 2.6.2. *Para todo AFD M existe um AFG G equivalente.*

Demonstração. Seja $M = \langle Q, \Sigma, \delta, q_o, F\rangle$. Construímos $G = \langle Q \cup \{q_i, q_f\}, \Sigma, \delta', q_i, q_f\rangle$ e para todo $q_j \in Q - \{q_f\}$ e $q_k \in Q - \{q_i\}$ temos:

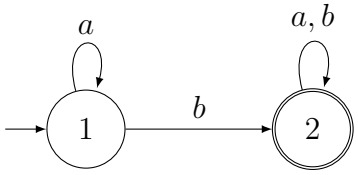
$$\delta'(q_j, q_k) = \begin{cases} \varepsilon & \text{se} \quad q_j = q_i \text{ e } q_k = q_0 \\ \varepsilon & \text{se} \quad q_j \in F \text{ e } q_k = q_f \\ \bigcup a_i & \text{se} \quad \delta(a_i, q_j) = q_k \\ \emptyset & \text{se} \quad \nexists a \in \Sigma \text{ com } \delta(a, q_j) = q_k \end{cases}$$

□

Exemplo 2.6.3:

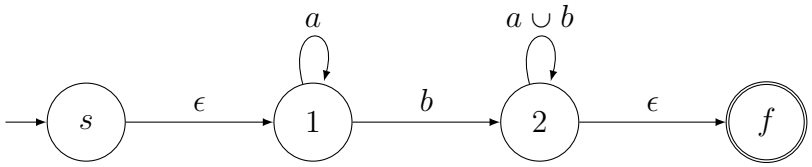
$$M = \langle \{1, 2\}, \{a, b\}, \delta, 1, \{2\}\rangle$$

δ	a	b
1	1	2
2	2	2



$$G = \langle \{1, 2, s, f\}, \{a, b\}, \delta', s, f\rangle$$

δ'	s	1	2	f
s	$\times$	ε	$\emptyset$	$\emptyset$
1	$\times$	a	b	$\emptyset$
2	$\times$	$\emptyset$	$a \cup b$	ε
f	$\times$	$\times$	$\times$	$\times$



Lema 2.6.4. *Se G é um AFG com $k > 2$ estados, então existe um AFG G' com $k - 1$ estados tal que $G \sim G'$.*

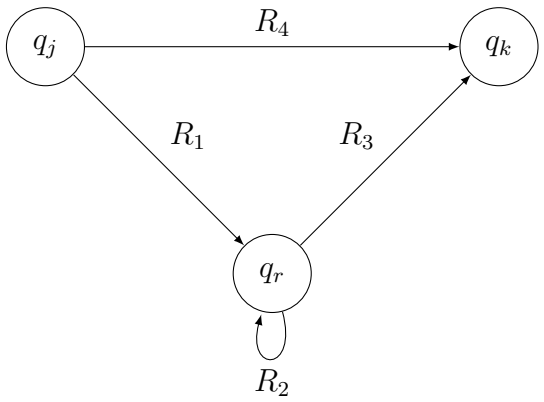
Demonstração. Seja $G = \langle Q, \Sigma, \delta, q_i, q_f \rangle$ um AFG e $q_r \in Q - \{q_i, q_f\}$, Construiremos $G' = \langle Q', \Sigma, \delta', q_i, q_f \rangle$ da seguinte forma:

$$Q' = Q - \{q_r\}$$

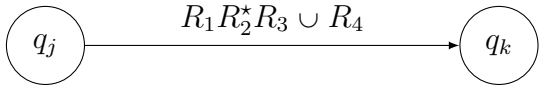
$\delta'(q_j, q_k) = R_1 R_2^\star R_3 \cup R_4$ aonde:

$$R_1 = \delta(q_j, q_r)$$
$$R_2 = \delta(q_r, q_r)$$
$$R_3 = \delta(q_r, q_k)$$
$$R_4 = \delta(q_j, q_k)$$

Diagramaticamente, partimos de um diagrama com o seguinte formato:



E depois de remover q_r chegamos nos seguinte:



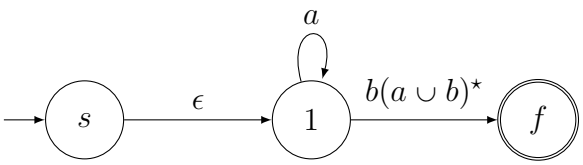
□

Exemplo 2.6.5:

Considere o AFG G do exemplo **??**. Vamos remover o estado 2 seguindo a construção do lema anterior.

$$G' = \langle \{s, 1, f\}, \{a, b\}, \delta', s, f \rangle$$

δ'	s	1	f
s	$\times$	ε	$\emptyset$
1	$\times$	a	$b(a \cup b)^\star$
f	$\times$	$\times$	$\times$



Teorema 2.6.6. *Toda linguagem reconhecível por AFD é regular.*

Demonstração. Seja A uma linguagem reconhecível por um AFD M i.e. $L(M) = A$. Pelo lema **??** existe um AFG G tal que $M \sim G$. Além disso, examinando a construção do lema **??** temos que se o número de estados de M é k então o número de estado de G é $k + 2$.

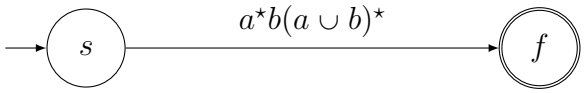
Pelo lema **??** existe G_1 com um estado a menos ($k + 1$ estados) tal que $G_1 \sim G$. Aplicando o lema k vezes obtemos $G_k \sim G$ com exatamente 2 estados: q_i e q_f. É claro que $L(G_k) = L(R)$ aonde $\delta_{G_k}(q_i, q_f) = R$. Como $G_k \sim G$, temos $L(G) = L(H) = L(R)$.

$$L(M) = L(G) = L(G_1) = \cdots = L(G_k) = L(R)$$

□

Exemplo 2.6.7:

Considere o AFD M do exemplo **??**. Mostramos que ele é equivalente a um AFG e, em no exemplo **??** mostramos um AFG equivalente com 3 estados. Removendo mais um estado ficamos com o seguinte diagrama:

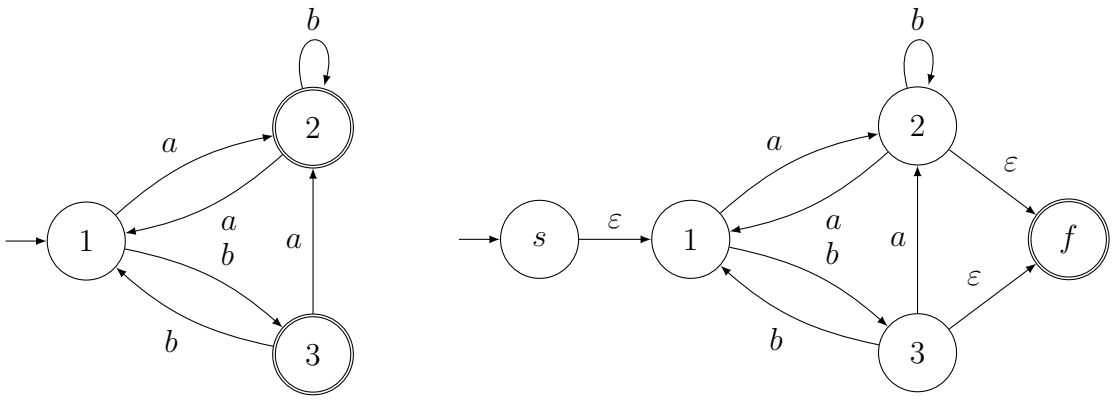


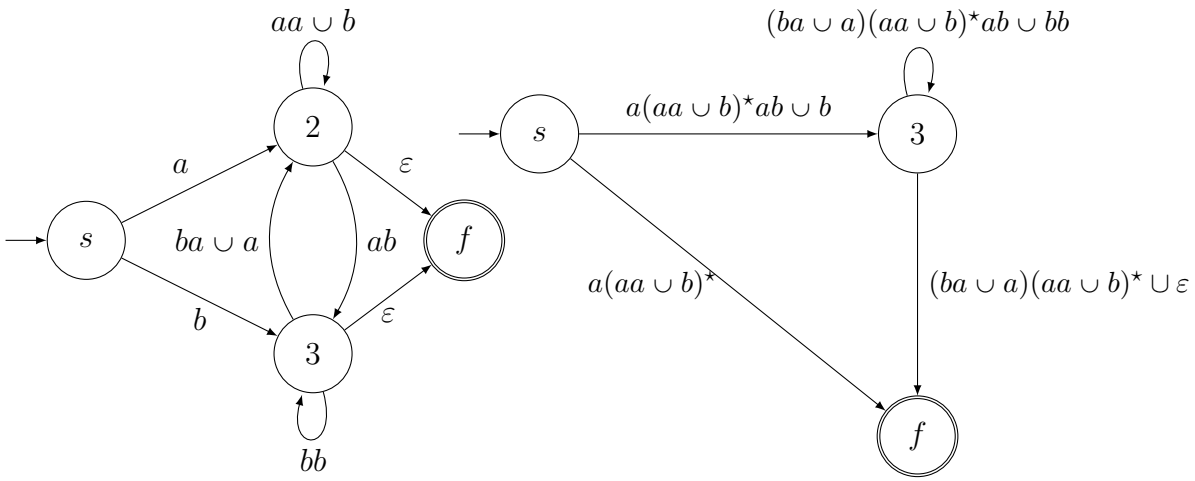
Portanto temos que:

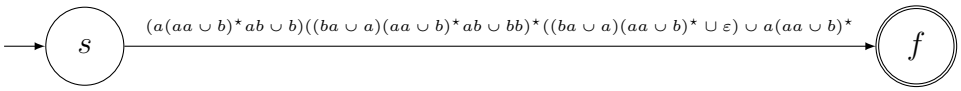
$$L(M) = L(a^\star b(a \cup b)^\star)$$

Exemplo 2.6.8:

Considere o seguinte AFD:







2.7 Linguagens Não-Regulares

Neste capítulo vimos dois modelos computacionais: AFDs e AFNs. Ambos reconhecem a mesma classe de linguagens, a saber, as linguagens regulares. Para completar esse capítulo veremos exemplos de linguagens que não são regulares.

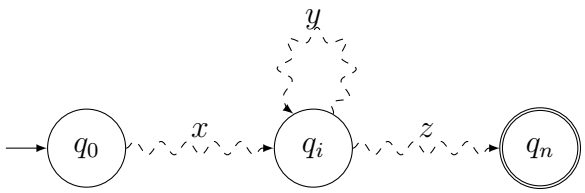
Lema 2.7.1 (Lema do Bombeamento)**.** *Se A é uma linguagem regular então exite um número p (chamado* comprimento do bombeamento*) tal que se $\omega \in A$ e $|\omega| \leq p$ então $\omega = x \cdot y \cdot z$ onde:*

1. *$x \cdot y^i \cdot z \in A$ para todo $i \leq 0$,*
2. *$|y| > 0$ e*
3. *$|x \cdot y| \leq p$*

Demonstração. Como A é regular, existe AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tal que $L(M) = A$. Seja $p = |Q|$ e seja $\omega \in A$ uma string tal que $|\omega| \leq p$.

Como $\omega = a_1 a_2 \ldots a_n \in A = L(M)$ existe uma sequência de estados $q_0, \ldots, q_n$ onde $q_n \in F$ e $q_{i+1} = \delta(q_i, a_{i+1})$. Como $|\omega| = n \geq p$, pelo *princípio da casa dos pombos*[1] existe pelo menos um estado que se repete na sequência.

Seja q_i o primeiro estado que se repete na sequência. Temos então a seguinte situação:



[1] O princípio da casa dos pombos garante que se há n pombos e p casas com $n > p$ então com certeza pelo menos uma casa terá mais de um pombo.

Ou seja, a sequência $q_0, \ldots, q_i$ reconhece a string x, a sequência $q_i, \ldots, q_i$ reconhece y e a sequência $q_i, \ldots, q_n$ reconhece z.

Portanto, ω pode ser dividido em três partes $\omega = x \cdot y \cdot z$ em que $|y| > 0$. Além disso, como q_i é o primeiro estado que se repete, temos que $|x \cdot y| \leq p$. Por fim, como ilustrado, o autômato reconhece y um número arbitrário de vezes. Ou seja, $x \cdot y^i \cdot z \in A$ para todo $i = 0, 1, 2, \ldots$ □

Exemplo 2.7.2:

Vamos mostrar que $A = \{0^n 1^n : n \geq 0\}$ não é regular.

Seja p p comprimento do bombeamento e $\omega = 0^p 1^p \in A$. Se A fosse regular, pelo lema do bombeamento, ω poderia ser dividida em três partes $\omega = xyz$ e para todo $i = 0, 1, \ldots$ teríamos $xy^i z \in A$.

Se y é formada só por 0s ou só por 1s então $xyyz$ possuiria um número diferente de 0s e 1s e, portanto, $xyyz \notin A$. Se y possuir 0s e 1s então $xyyz$ conterá pelo menos um 0 entre dois 1s e, portanto, $xyyz \notin A$.

Logo A não pode ser regular.

Exemplo 2.7.3:

Mostraremos que $A = \{\omega : \omega$ tem o mesmo número de 0s e 1s$\}$ não é regular.

Seja p o comprimento do bombeamento e $\omega = 0^p 1^p \in A$. Se A fosse regular poderíamos escrever $\omega = xyz$ com $|xy| \leq p$ e $xy^i z \in A$ para todo $i \geq 0$.

Como $|xy| \leq p$, por definição, xy contém apenas 0s. Neste caso, $xyyz$ deve conter mais 0 do que 1s. Logo $xyyz \notin A$.

Exemplo 2.7.4:

Vamos mostrar que $A = \{\omega\omega : \omega \in \{0, 1\}^*\}$ não é regular.

Seja p o comprimento do bombeamento e $\omega = 0^p 1^p 0^p 1 \in A$. Dividimos $\omega = xyz$ com $|xy| \leq p$ então y contém apenas 0s e $xyyz = 0^p 0 \ldots 0 1 0^p 1 \notin A$.

Exemplo 2.7.5:

Vamos mostrar que $A = \{1^{n^2} : n \geq 0\}$ não é regular.

Seja p o comprimento do bombeamento e $\omega = 1^{p^2} \in A$. Dividimos $\omega = xyz$ com $|xy| \leq p$. Como $|y| \leq p$ então $|xyyz| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$. Logo $p^2 < |xyyz| < (p+1)^2$.

Exemplo 2.7.6:

Por fim, vamos mostrar que $A = \{0^i 1^j : i < j\}$ não é regular.

Seja p o comprimento do bombeamento e $\omega = 0^{p+1}1^p \in A$. Tomamos $\omega = xyz$ com $|xy| \leq p$ e $|y| > 0$. Portanto, y contém apenas 0s e $xy^0z = xz \notin A$.

Neste capítulo estudamos a classe das linguagens regulares e dois modelos de computação, autômatos finitos determinísticos e não-determinísticos. Provamos que esses dois modelos são equivalentes e que a classe das linguagens que eles reconhecem coincide com a classe das linguagens regulares. Terminamos o capítulo vendo exemplos de linguagens que não são regulares.

Capítulo 3

Linguagens Livres de Contexto

Estudamos até exaustivamente uma classe de linguagens, as regulares. Apresentamos essa classe de maneira declarativa por meio de expressões regulares e imperativa com dois modelos de computação: autômatos finitos determinísticos e não-determinísticos que vimos serem equivalentes. No fim do capítulo mostramos que nem toda linguagem é regular.

Neste capítulo estudaremos uma classe de linguagens mais completa, as linguagens livres de contexto. Como as linguagens regulares, apresentaremos tais linguagens de maneira declarativa por meio de gramáticas livres de contexto e imperativa por meio dos autômatos com pilha.

3.1 Introdução

Uma *Gramática Livre de Contexto* (GLC) é uma 4-upla $\langle V, \Sigma, R, S\rangle$ em que:

> V é um conjunto finito cujos elementos são chamados *variáveis*,
>
> Σ é um conjunto finito disjunto de V (i.e. $\Sigma \cap V \neq \emptyset$) cujos elementos são chamados *terminais*,
>
> R é um conjunto finito de *regras* e cada regra é da forma $v_1 \to v_2 \dots v_n$ onde $v_1 \in V$ e $v_i \in V \cup \Sigma$ para $i = 2, \dots, n$ e
>
> $S \in V$ é uma variável chamada *inicial*.

Se u, v e w strings sobre o alfabeto $V \cup \Sigma$ e $A \to w$ é uma regra da gramática, dizemos que uAv *origina* uwv (escrevemos $uAv \Rightarrow uwv$). Dizemos

que u *deriva* v (escrevemos $u \Rightarrow^* v$) se $u = v$ ou existe uma sequência $u_1, \ldots, u_k$ para $k \geq 0$ em que:

$$u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

A *linguagem associada à gramatica* $G = \langle V, \Sigma, R, S \rangle$, ou simplesmente a linguagem de G é $\{\omega \in \Sigma^* : S \Rightarrow^* \omega\}$.

Se uma linguagem A possui uma gramatica livre de contexto associada a ele então A é chamada *linguagem livre de contexto*.

Exemplo 3.1.1:

Considere a seguinte GLC $G = \langle V, \Sigma, R, S \rangle$:

- $V = \{S\}$
- $\Sigma = \{0, 1\}$
- $R = \{S \to \varepsilon, S \to 0S1\}$

01 pertence a linguagem dessa gramática:

$$\begin{aligned} S &\Rightarrow 0S1 \\ &\Rightarrow 0\varepsilon 1 = 01 \end{aligned}$$

000111 pertence a linguagem dessa gramática:

$$\begin{aligned} S &\Rightarrow 0S1 \\ &\Rightarrow 00S11 \\ &\Rightarrow 000S111 \\ &\Rightarrow 000\varepsilon 111 \\ &\Rightarrow 000111 \end{aligned}$$

Não é difícil notar que a linguagem dessa gramática é:

$$\{0^n 1^n : n \geq 0\}$$

Para apresentar as próximas gramáticas, usaremos a seguinte abreviação:

$$\{A \to w, A \to u, A \to v\}$$

Será substituído simplesmente por:

$$A \to w|u|v$$

Exemplo 3.1.2:

$G = \langle \{S\}, \{0, 1\}, R, S \rangle$ em que R é:

$$S \to SS|0|1$$

01 pertence à linguagem de G:

$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow 0S \\ &\Rightarrow 01 \end{aligned}$$

0101 pertence à linguagem de G:

$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow 0S \\ &\Rightarrow 0SS \\ &\Rightarrow 0S0 \\ &\Rightarrow 010 \end{aligned}$$

Exemplo 3.1.3:

$$G = \langle \{S\}, \{0, 1, \star, \cup, \epsilon, \emptyset\}, R, S \rangle$$
$$S \to 0|1|\epsilon|\emptyset|SS|S \cup S|S^\star$$

Vamos mostrar que $10 \cup 1^\star \in L(G)$

$$\begin{aligned} S &\Rightarrow S \cup S \\ &\Rightarrow SS \cup S \\ &\Rightarrow SS \cup S^\star \\ &\Rightarrow 1S \cup S^\star \\ &\Rightarrow 10 \cup S^\star \\ &\Rightarrow 10 \cup 1^\star \end{aligned}$$

Exemplo 3.1.4:

$$G = \langle V, \Sigma, R, Expr \rangle$$

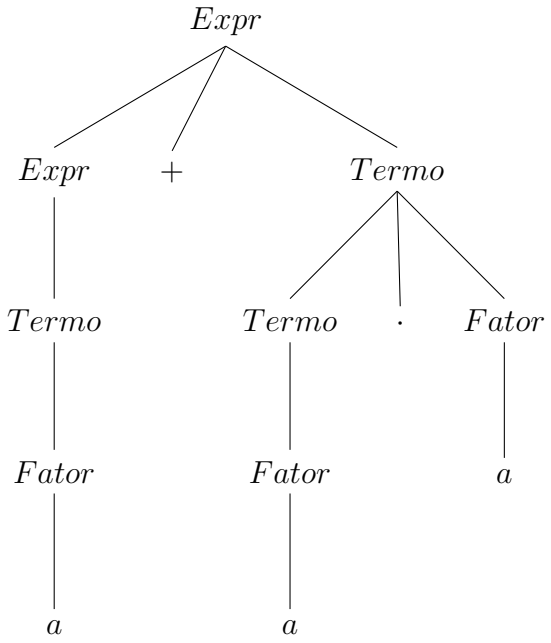
- $V = \{Expr, Termo, Fator\}$
- $\Sigma = \{a, +, \cdot, (,)\}$

$$\begin{aligned} Expr &\rightarrow Expr + Termo | Termo \\ Termo &\rightarrow Termo \cdot Fator | Fator \\ Fator &\rightarrow (Expr) | a \end{aligned}$$

Vamos mostrar que $a + a \cdot a \in L(G)$.

$$\begin{aligned} Expr &\Rightarrow Expr + Termo \\ &\Rightarrow Expr + Termo \cdot Fator \\ &\Rightarrow Termo + Termo \cdot Fator \\ &\Rightarrow Fator + Fator \cdot Fator \\ &\Rightarrow a + a \cdot a \end{aligned}$$

Podemos representar a derivação do último exemplo por meio de uma *árvore sintática*:



Note que uma mesma string pode ser derivada de uma mesma gramática por diferentes árvores sintáticas. Esse fenômeno é chamado *ambiguidade*.

Uma derivação de uma string ω em uma gramática G é uma *derivação mais a esquerda* se a cada passo a variável remanescente mais a esquerda é aquela que será substituída no próximo passo. Uma string é *derivada de maneira ambígua* na gramática G se ela tem mais de uma derivação à esquerda. Uma GLC é dita *ambígua* se ela gera alguma string de maneira ambígua.

Exemplo 3.1.5:

$$G = \langle \{S\}, \{+, \cdot, a\}, R, S \rangle$$

$$S \to S + S | S \cdot S | a$$

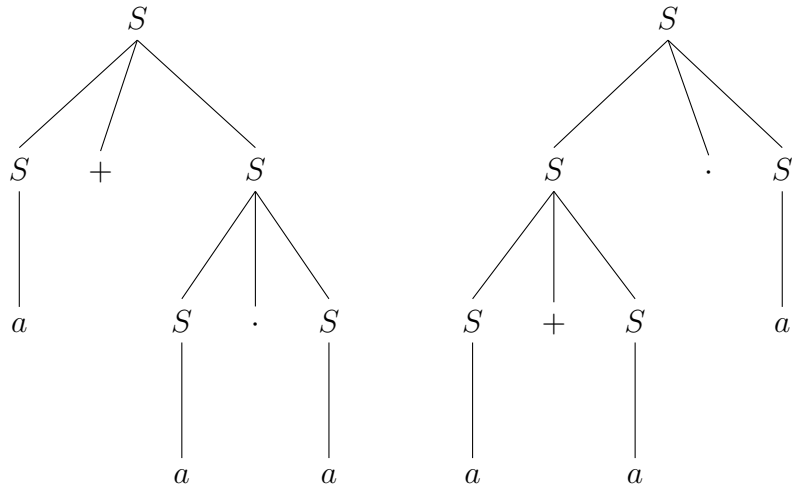
Vamos derivar a esquerda a expressão $a + a \cdot a$:

$$\begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow a + S \\ &\Rightarrow a + S \cdot S \\ &\Rightarrow a + a \cdot a \end{aligned}$$

Alternativamente podemos derivar a mesma expressão à esquerda da seguinte maneira:

$$\begin{aligned} S &\Rightarrow S \cdot S \\ &\Rightarrow S + S \cdot S \\ &\Rightarrow a + S \cdot S \\ &\Rightarrow a + a \cdot a \end{aligned}$$

Essas derivações são representadas pelas seguintes árvores sintáticas.



Uma GLC está na *Forma Normal de Chomsky* (FNC) se toda regra é de uma das seguintes formas:

$S \to \varepsilon$

$A \to BC|a$

Onde $a \in \Sigma$, $A \in V$ e $B, C \in V - \{S\}$.

Teorema 3.1.6. *Toda linguagem livre de contexto é gerada por uma GLC na FNC.*

Demonstração. Essa prova é construtiva.

1. Criamos um novo estado S_0 e uma regra $S_0 \to S$
2. Removemos cada regra da forma $A \to \varepsilon$ e criamos uma nova regra para cada ocorrência de A a direita de uma regra em que A não ocorre (por exemplo $R \to uAvAw$ gera três regras $R \to uvAw$, $R \to uAvw$ e $R \to uvw$).
3. Removemos todas as regras da forma $A \to B$ e criamos uma regra $A \to u$ para cada ocorrência de $B \to u$.
4. Substituímos $A \to u_1 u_2 \dots u_k$ onde $k > 2$ e $u_i \in V \cup \Sigma$ por $A \to u_1 A_1$, $A \to u_2 A_2$, $\dots$, $A_{k-2} \to u_{k-1} u_k$.
5. Substituímos $A \to cB$ (e $A \to Bc$) por $A \to BC$ (ou $A \to CB$) e $C \to c$

□

Exemplo 3.1.7:

Considere por exemplo a seguinte GLC:

$$\begin{aligned} S &\to ASA|aB \\ A &\to B|S \\ B &\to b|\varepsilon \end{aligned}$$

Aplicando o primeiro passo da construção obetmos:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA|aB \\ A &\to B|S \\ B &\to b|\varepsilon \end{aligned}$$

Aplicando o segundo passo primeiro removemos $B \to \varepsilon$:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA|aB|a \\ A &\to B|S|\varepsilon \\ B &\to b \end{aligned}$$

Em seguida removemos $A \to \varepsilon$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA|AS|SA|aB|a \\ A &\to B|S \\ B &\to b \end{aligned}$$

Aplicando terceiro passo ficamos com o seguinte:

$$\begin{aligned} S_0 &\to ASA|AS|SA|aB|a \\ S &\to ASA|AS|SA|aB|a \\ A &\to b|ASA|AS|SA|aB|a \\ B &\to b \end{aligned}$$

O quarto passo consiste em substituir as sequência de mais de duas variáveis não terminais:

$$\begin{aligned} S_0 &\to AA_1|SA|AS|SA|S|aB|a \\ S &\to AA_1|AS|SA|S|aB|a \\ A &\to b|AA_1|AS|SA|S|aB|a \\ A_1 &\to SA \\ B &\to b \end{aligned}$$

Para concluir substituímos os símbolos terminais em regras com um símbolo não terminal:

$$\begin{aligned} S_0 &\to AA_1|SA|AS|SA|UB|a \\ S &\to AA_1|AS|SA|UB|a \\ A &\to b|AA_1|AS|SA|UB|a \\ A_1 &\to SA \\ B &\to b \\ U &\to a \end{aligned}$$

3.2 Autômatos de Pilha

Em um dos exemplos da seção anterior vimos que a linguagem não-regular $\{0^n1^n : n \geq 0\}$ é livre de contexto. Portanto, os autômatos finitos não são adequados para reconhecer LLC. Nesta seção veremos um novo modelo de computação chamado autômato com pilha e mais para frente mostraremos sua relação íntima com as LLCs.

Um *autômato com pilha* (AP) é uma 6-upla $\langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ onde:

- Q é um conjunto finito cujos elementos são chamados *estados*,
- Σ é um alfabeto chamado *alfabeto de entrada*,
- Γ é um alfabeto chamado *alfabeto da pilha*,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon) \to 2^{Q\times(\Gamma\cup\{\varepsilon\})}$ é a *função de transição*,
- $q_0 \in Q$ é o *estado inicial* e
- $F \subseteq Q$ é o conjunto dos *estados finais*.

Um AP $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ *aceita* uma string ω se $\omega = \omega_1\omega_2\ldots\omega_n$ onde $\omega_i \in \Sigma\cup\{\varepsilon\}$ e existe uma sequência de estados $r_0, r_1, \ldots, r_m \in Q$ e uma sequência de strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ tal que:

1. $r_0 = q_0$ e $s_0 = \varepsilon$ (a pilha começa vazia),
2. $\langle r_{i+1}, b\rangle \in \delta(r_i, \omega_{i+1}, a)$ onde $s_i = at$ e $s_{i+1} = bt$ para $t \in \Gamma^*$ (lê um símbolo, vai para o próximo estado e atualiza a pilha) e
3. $r_m \in F$ (termina em um estado final).

A cada passo o AP lê um símbolo $\omega_{i+1} \in \Sigma\cup\{\varepsilon\}$, desempilha um símbolo $a \in \Gamma\cup\{\varepsilon\}$ da pilha, empilha outro $b \in \Gamma\cup\{\varepsilon\}$ e vai para o novo estado r_{i+1}.

Exemplo 3.2.1:

$$\begin{aligned} M &= \langle Q, \Sigma, \Gamma, \delta, q_1, F\rangle \\ Q &= \{q_1, q_2, q_3, q_4\} \\ \Sigma &= \{0, 1\} \\ \Gamma &= \{0, \$\} \\ F &= \{q_1, q_4\} \end{aligned}$$

δ é dado pela seguinte tabela:

	0			1			ε		
	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Na tabela omitimos as células que deveriam ser preenchidas por $\emptyset$ deixando-as vazias.

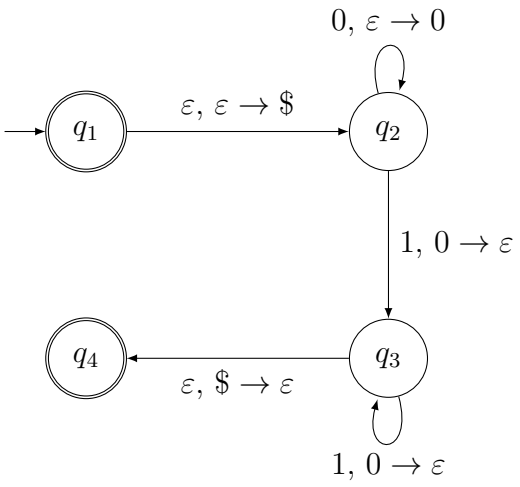
M reconhece a string 01:

1. Começo no estado q_1, emppilho \$ e vou para q_2 (pilha: \$).
2. Leio 0, empilho 0 e fico em q_2 (pilha: 0\$).
3. Leio 1, desempilho 0 e vou para q_3 (pilha: \$).
4. Desempilho \$ e vou para $q_4 \in F$ (pilha: ε).

Podemos representar um AP usando um diagrama de estados. O diagrama de estados de um AP é como um diagrama de AFNs, mas em cada transição, além do símbolo a ser lido, indicamos os símbolos a serem desempilhados e empilhados (exemplo $a \to b$ indica que deve-se desempilhar a e empilhar b).

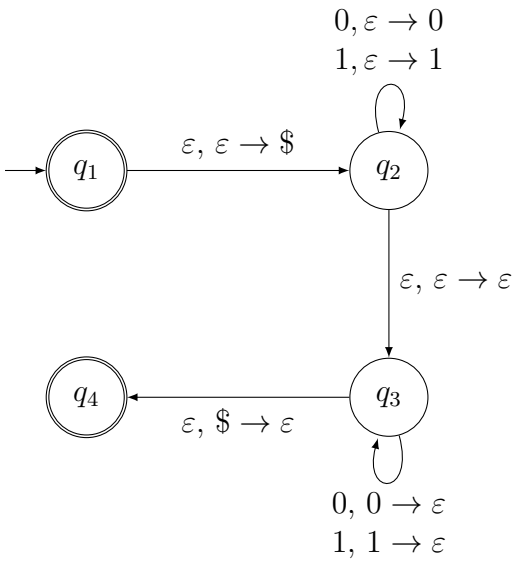
Exemplo 3.2.2:

Vamos ilustrar o autômato do último exemplo:



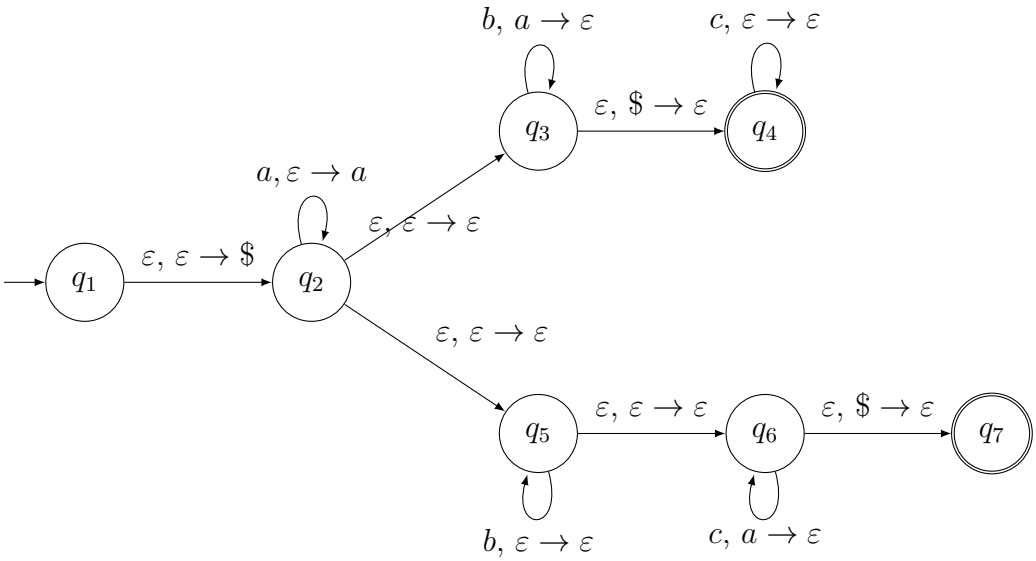
A pilha garante que será reconhecida a mesma quantidade de 0s e de 1s. Portanto $L(G) = \{0^n 1^n : n \geq 0\}$

Exemplo 3.2.3:



Procure verificar com alguns exemplos que $L(G) = \{\omega\omega^R : \omega \in \{0,1\}^*\}$ aonde ω^R é ω escrito de trás para frente.

Exemplo 3.2.4:



Procure verificar com alguns exemplos que:

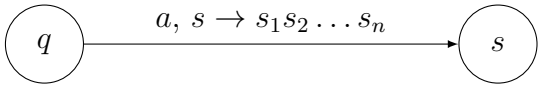
$$L(G) = \{a^i b^j c^k : i = j \text{ ou } i = k\}$$

3.3 LLCs são Reconhecíveis por APs

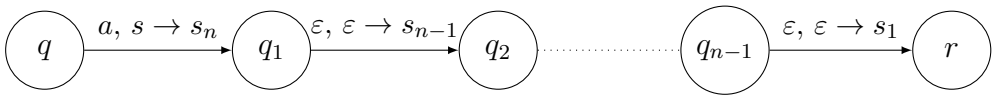
Para esta seção usaremos uma abreviação para descrever o empilhamento de uma sequência de símbolos. Seja $\omega \in \Gamma^*$, $r, q \in Q$, $a \in \Sigma$ e $s \in \Gamma$, escrevemos $\langle r, \omega \rangle \in \Delta(q, a, s)$ para indicar que ao ler a no estado q, desempilhamos s e empilhamos cada um dos símbolos de ω antes de ir para r. Ou seja, se $\omega = s_1 s_2 \ldots s_n \in \Gamma^*$, então $\langle r, \omega \rangle \in \Delta(q, a, s)$ é uma abreviação para:

$$\begin{aligned}
\langle q_1, s_n \rangle &\in \Delta(q, a, s) \\
\{\langle q_2, s_{n-1} \rangle\} &= \Delta(q_1, \varepsilon, \varepsilon) \\
\{\langle q_3, s_{n-2} \rangle\} &= \Delta(q_2, \varepsilon, \varepsilon) \\
&\ldots \\
\{\langle r, s_1 \rangle\} &= \Delta(q_{n-1}, \varepsilon, \varepsilon)
\end{aligned}$$

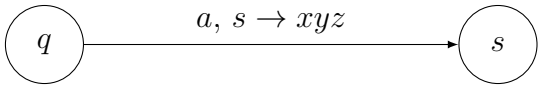
No diagrama de estados, escrevemos:



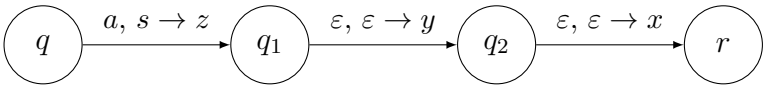
Para abreviar o seguinte:



Exemplo 3.3.1:



É uma abreviação de:



Teorema 3.3.2. *Toda linguagem livre de contexto é reconhecida por um Autômato com Pilha.*

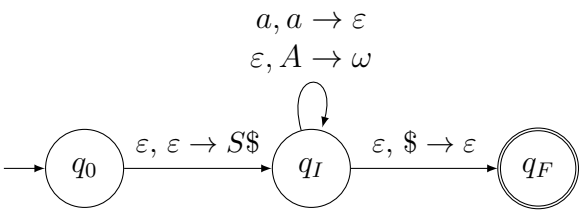
Demonstração. Se A é uma LLC, por definição, existe uma GLC $G = \langle V, \Sigma, R, S\rangle$ associada a A i.e. $L(G) = A$.

Construiremos um AP $P = \langle Q, \Sigma, \Gamma, \Delta, q_0, F\rangle$ que reconhece A i.e. $L(P) = A$.

- $Q = \{q_0, q_I, q_F\} \cup E$ onde E é o conjunto de estados necessários para abreviação que vimos acima.
- $F = \{q_F\}$
- $\Gamma = V \cup \Sigma \cup \{\$\}$
- Δ é apresentado abaixo.

$$\begin{aligned}
\Delta(q_0, \varepsilon, \varepsilon) &= \{\langle q_I, S\$\rangle\} \\
\Delta(q_I, \varepsilon, \$) &= \{\langle q_F, \varepsilon\rangle\} \\
\Delta(q_I, a, a) &= \{\langle q_I, \varepsilon\rangle\} \text{ para todo } a \in \Sigma \\
\Delta(q_I, \varepsilon, A) &= \{\langle q_I, \omega\rangle\} \text{ para todo } A \to \omega \in R
\end{aligned}$$

Diagramaticamente temos:



Em palavras, primeiro inserimos \$ para marcar o fim da pilha e em seguida inserimos a variável inicial S na pilha e seguimos para o estado intermediário q_I. Então, não-deterministicamente empilhamos o corpo de alguma das regras $A \to \omega$ ou desempilhamos um símbolo terminal a e o reconhecemos. Quando a pilha chega no fim (no símbolo \$), desempilhamos e vamos para o estado final. □

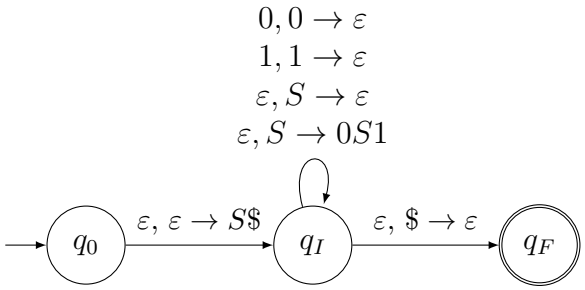
Exemplo 3.3.3:

Considere a gramática $G = \langle V, \Sigma, R, S\rangle$ aonde R possui as seguintes regras:

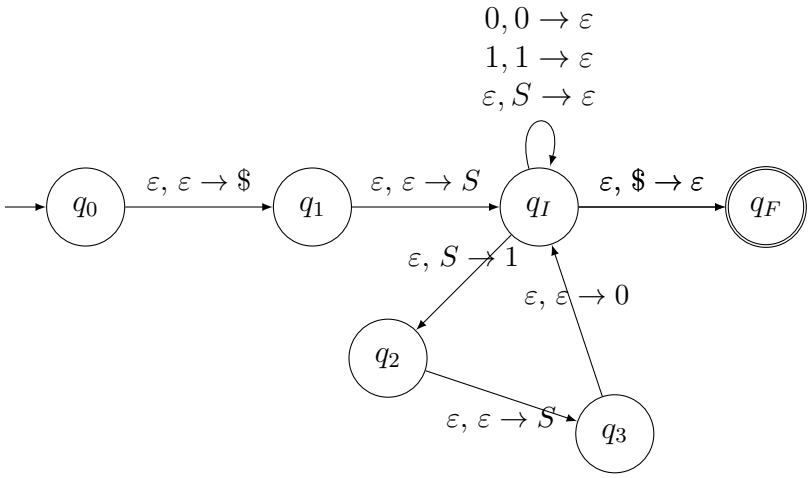
$$S \to 0S1|\varepsilon$$

Como já vimos, $L(G) = \{0^n 1^n : n \geq 0\}$.

Usando a construção do teorema anterior, temos que o seguinte AP reconhece essa linguagem:



O diagrama acima é uma abreviação para o seguinte diagrama:



Exemplo 3.3.4:

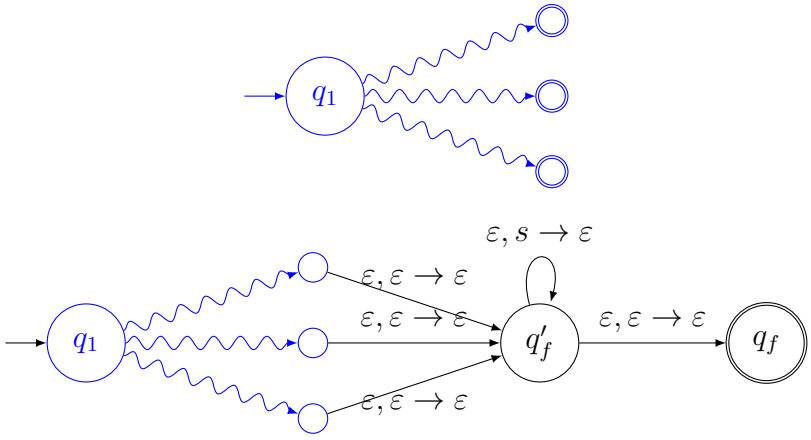
Exemplo 3.3.5:

3.4 Linguagens Reconhecíveis por APs são Livres e Contexto

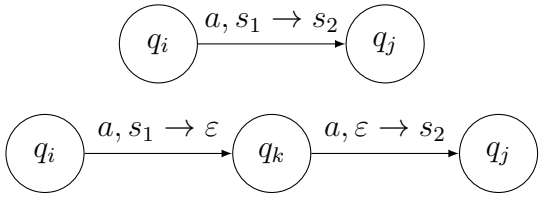
Lema 3.4.1. *Todo AP P é equivalente a outro AP P' em que:*

1. *o conjunto de estados finais possui um único elemento q_f,*
2. *as transações só empilham ou desempilham, mas nunca ambas ao mesmo tempo e*
3. *chega ao estado final com a pilha vazia.*

Demonstração. Para garantir os itens 1 e 3 criamos transições de cada estado final de P para q'_f que não leem nada e não empilham nem desempilham nada (setas com etiqueta $\varepsilon, \varepsilon \to \varepsilon$). Uma transição de q'_f para si mesmo que não le nada e desempilha s para cada $s \in \Gamma$ e uma transição que não lê nada e não mexe na pilha que vai de q'_f para q_f.



Para garantir a condição 2, substituimos toda transição que empilha e desempilha ao mesmo tempo por uma que desempilha seguida por outra que empilha.



□

Teorema 3.4.2. *Toda linguagem reconhecida por APs é livre de contexto.*

Demonstração. Seja $P = \langle Q, \Sigma, \Gamma, \Delta, q_0, F\rangle$. Pelo lema anterior existe P' equivalente a P satisfazendo as três proriedades. Criaremos uma gramática $G = \langle \Sigma, V, R, S\rangle$ que reconhece $L(P) = L(P')$.

- $V = \{A_{pq} : p, q \in Q\}$
- $S = A_{p_0p_f}$
- R é formado por três tipos de regras:
  1. $A_{pq} \to A_{pr}A_{rq} \in R$ para todo $p, r, q \in Q$
  2. se $\langle r, t\rangle \in \Delta(p, a, \varepsilon)$ e $\langle q, \varepsilon\rangle \in \Delta(s, b, t)$ então $A_{pq} \to aA_{rs}b \in R$
  3. $A_{pp} \to \varepsilon \in R$ para todo $p \in Q$

Primeiro demonstraremos por indução no tamanho da derivação o seguinte:

Hipótese de Indução: Se $A_{pq} \Rightarrow^k x$ então P começa no estado p, reconhece x e chega no estado q com a pilha vazia.

Base: As únicas derivações de tamanho 1 são da forma $A_{pp} \to \varepsilon$ e claro que o autômato que começa e termina em q reconhece ε.

Passo de Indução: Precisamos mostrar que se $A_{pq} \Rightarrow^{k+1} x$ então P começa em p, reconhece x e chega em q com a pilha vazia. O primeiro passo dessa derivação deve ser

1. $A_{pq} \Rightarrow aA_{rs}b$ ou
2. $A_{pq} \Rightarrow A_{pr}A_{rq}$

No primeiro caso temos que $x = ayb$ e, portanto, $A_{rs} \Rightarrow^k y$. Pela H.I. P reconhece y indo de r até s e terminando com a pilha vazia. Como a $A_{pq} \to aA_{rs}b \in R$ então $\langle r, t\rangle \in \Delta(p, a, \varepsilon)$ e $\langle q, \varepsilon\rangle \in \Delta(s, b, t)$. Então p vai para r e empilha t ao ler a e desempilha t ao ler b e ir para q.

No segundo caso, temos que $x = yz$ e $A_{pr} \Rightarrow^* y$ e $A_{rq} \Rightarrow z$. Ambas derivações devem ter comprimento menor que $k + 1$ e, logo, pela H.I. P reconhece y indo de p para r e reconhece z indo de r até q.

Por fim, resta provar por indução no número de passos de computação de P que:

Hipótese de Indução: Se P reconhece x indo de p para q em k passos então $A_{pq} \Rightarrow^* x$

Base: Em 0 passos não sai do estado p e reconhece ε. Pela regra $A_{pp} \to \varepsilon$ geramos ε.

Passo de Indução: Suponha que P vai de p até q em $k+1$ passos e reconhece x.

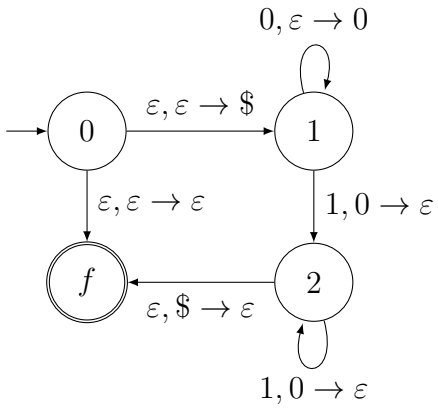
Suponhamos primeiro que em nenhum momento no processo a pilha fique vazia. Neste caso, o símbolo t empilhado no começo é desempilhado no fim. Se a é o símbolo lido no começo, b o símbolo lido no fim, então r o estado seguinte a p e s é o anterior a q. Ou seja, se $\langle r, t\rangle \in \Delta(p, a, \varepsilon)$ e $\langle q, \varepsilon\rangle \in \Delta(s, b, t)$ então $A_{pq} \to aA_{rs}b \in R$. Seja $x = ayb$, pela H.I., $A_{rs} \Rightarrow^* y$ e logo $A_{pq} \Rightarrow^* x$.

Por outro lado, se a pilha chega esvaziar entçai ela reconhece uma string y até ficar vazia e z até q e $x = yz$. Seja r o estado em P quando a pilha está vazia. Pela H.I. $A_{pr} \Rightarrow^* y$ e $A_{rp} \Rightarrow^* z$. Como $A_{pq} \to A_{pr}A_{rq} \in R$ então $A_{pq} \Rightarrow^* yz = x$. □

Corolario 3.4.3. *Toda linguagem regular é livre de contexto.*

Demonstração. Não é difícil notar que todo AFD é um AP aonde a pilha nunca é usada. Vimos que toda linguagem regular é reconhecida por um AFD, portanto toda linguagem regular é reconhecida por um AP e portanto é livre de contexto.2 □

Exemplo 3.4.4:



$$\left.\begin{array}{rcl}\langle 1,0\rangle & \in & \Delta(1,0,\varepsilon)\\ \langle 2,\varepsilon\rangle & \in & \Delta(2,1,0)\end{array}\right\} A_{12} \to 0A_{12}1$$

$$\left.\begin{array}{rcl}\langle 2,0\rangle & \in & \Delta(1,0,\varepsilon)\\ \langle 2,\varepsilon\rangle & \in & \Delta(2,1,0)\end{array}\right\} A_{12} \to 0A_{22}1$$

$$\left.\begin{array}{rcl}\langle 1,\$\rangle & \in & \Delta(0,\varepsilon,\varepsilon)\\ \langle f,\varepsilon\rangle & \in & \Delta(2,\varepsilon,\$)\end{array}\right\} A_{0f} \to \varepsilon A_{12}\varepsilon$$

P reconhece 0011, portanto essa string deve estar em $L(G)$ onde G é a gramática acima.

$$\begin{array}{rcl} A_{0f} & \Rightarrow & \varepsilon A_{12}\varepsilon \\ & \Rightarrow & 0A_{12}1 \\ & \Rightarrow & 00A_{12}11 \\ & \Rightarrow & 00\varepsilon 11 = 0011 \end{array}$$

3.5 Linguagens que não são Livres de Contexto

Lema 3.5.1 (Bombeamento para LLCs)**.** *Se A é uma LLC então existe p (comprimento do bombeamento) tal que se $\omega \in A$ e $|\omega| \geq p$ então $\omega = uvxyz$ e:*

1. *$uv^ixy^iz \in A$ para todo $i \geq 0$*
2. *$|vy| > 0$ e*
3. *$|vxy| \leq p$*

Demonstração. Se A é uma LLC, então por definição existe uma GLC $G = \langle \Sigma, V, R, S\rangle$ tal que $L(G) = A$. Seja b o número máximo de símbolos a direita em uma regra em R. Se partirmos de uma varíavel qualquer em G, em h passos o comprimento máximo da string que é possível produzir é b^h (se

desenharmos a árvore sintática da string produzida desta forma, h é a altura desta árvore).

O comprimento do bombeamento será $p = b^{|V|+1}$. Se $\omega \in A$ e $|\omega| \geq p$, como na hipótese, existe k tal que $S \Rightarrow^k \omega$. Vamos supor que k seja o menor valor em que S deriva ω. Note que necessariamente $k \geq |V| + 1$. É claro que esse caminho possui $|V| + 1$ símbolos não-terminais, logo, pelo princípio da casa dos pombos, pelo menos uma variável ocorre mais de uma de uma vez neste caminho. Seja $T \in V$ a última variável que ocorre mais de uma vez.

Dividimos ω em 5 partes $\omega = uvxyz$ de forma que a penúltima ocorrência de T gera vxy e a última gera x (i.e. $S \Rightarrow^* uTz \Rightarrow^* uvTyz \Rightarrow uvxyz$).

Note que $S \Rightarrow^* uxz$ se substituirmos a penúltima ocorrência de T pela última (i.e. $S \Rightarrow^* uTz \Rightarrow^* uxz$).

Da mesma forma, $S \Rightarrow^* uv^ixy^iz$ para qualquer $i > 1$ bastando repetir i vezes a última ocorrência de T pela penúltima.

Se $|vy| = 0$ então $v = y = \varepsilon$ e, portanto, $S \Rightarrow^l uxz = \omega$ como substituindo a penúltima ocorrência de T pela última e $l < k$ contrariando a suposição.

Se $|vxy| > p$ então, pelo princípio da casa dos pombos, na derivação da penúltima ocorrência de T até vxy alguma variável deveria repetir contrariando a suposição de que T era a última variável que se repetia. □

Exemplo 3.5.2:

$B = \{a^nb^nc^n : n \geq 0\}$ não é livre de contexto.

Seja p o comprimento do bombeamento, e $\omega = a^pb^pc^p \in B$. Se B fosse uma LLC então, pelo lema, $\omega = uvxyz$, $|vy| > 0$ e $uv^ixy^iz \in B$ para todo $i \geq 0$. Temos duas possibilidades:

1. se v e y um único tipo de símbolo cada, então uv^2xy^2z não conterá a mesma quantidade de as, bs e cs e, portanto, $uv^2xy^2z \notin B$.
2. se v ou y contém mais de um símbolo distinto então uv^2xy^2z contém símbolos na ordem errada e, portanto, $uv^2xy^2z \notin B$.

Concluímos que B não é livre de contexto.

Exemplo 3.5.3:

$C = \{a^i b^j c^k : 0 \leq i \leq j \leq k\}$ não é livre de contexto.

Seja p o comprimento do bombeamento e $\omega = a^p b^p c^p \in C$. Pelo lema, se C fosse livre de contexto, teríamos $\omega = uvxyz$ com $|vy| > 0$ e $uv^i xy^i z \in C$ para todo $i > 0$. Considere os dois possíveis casos:

1. y e v só contém um tipo de símbolo cada:
   - se a não ocorre em vy então $uxz \notin C$;
   - se b não ocorre em vy, mas a ocorre, então $uv^2xy^2z \notin C$, pois possui mais as do que bs e se c ocorre $uv^2xy^2z \notin C$ por motivo análogo;
   - se c não ocorre em vy então $uv^2xy^2z \notin C$, pois a string possuiria mais as ou mais bs do que c.
2. se y ou z possuem mais de um tipo de símbolo então $uv^2xy^2z \notin C$, pois possui símbolos na ordem errada.

Concluímos que C não é livre de contexto.

Na seção anterior, vimos que todas as linguagens regulares são livres de contexto, mas anteriormente havíamos mostrado que exitem linguagens livres de contexto ($\{0^n 1^n : n \geq 0\}$ por exemplo) que não são regulares.

$$\text{Ling. Reg.} \subset \text{LLCs}$$

Nesta seção vimos que esxistem linguagens formais que não são livres de contexto:

$$\text{LLCs} \subset \text{Ling. Formais}$$

Além disso, vimos que linguagens regulares coincidem com as linguagens reconhecíveis por autômatos finitos e que as livres de contexto coincidem com as reconhecíveis por autômatos com pilha.

No próximo capítulo passaremos à questão central do curso, a saber, a existência de problemas que não possuem solução computacional. Para tanto, precisamos de um modelo de computação capaz de dar conta de qualquer dispositivo mecânico.

Capítulo 4

Máquinas de Turing

Estudamos até aqui modelos de computação de expressividade crescente. Começamos com autômatos finitos, vimos que existem linguagens que não conseguimos reconhecer com esse tipo de autômatos. Identificamos exatamente a classe de linguagens que esse tipo de modelo é capaz de reconhecer, a saber, as linguagens regulares. Passamos então para os autômatos com pilha que são mais expressivos, reconhecem todas as liguagens regulares e mais algumas não-regulares. Novamente encontramos limitações, linguagens que não são reconhecíveis por autômatos com pilha.

Neste capítulo estudaremos um modelo ainda mais expressivo, as Máquinas de Turing (MTs). Temos dois grandes objetivos neste capítulo. O primeiro é convencer que este é o modelo definitivo de computação, ou seja, que não existe modelo de computação mais expressivo que as Máquinas de Turing. Esse resultado, que não é nem pode ser um teorema, é chamado de Tese de Church-Turing. O argumento para esta tese serão três: provaremos que as MTs são mais expressivas que os autômatos com pilha, em seguida mostraremos uma serie de variantes das MTs e provaremos que todas são equivalentes (i.e. tem a mesma expressividade) e, por fim, provaremos que toda MT pode ser simulada por uma MT específica chamada de Máquina de Turing Universal. O segundo objetivo deste capítulo é provar que, mesmo sendo o modelo de computação mais completo, as MTs possuem limitações. Ou seja, existem problemas computacionais que não podem ser resolvidos por MTs.

4.1 Máquinas de Turing Determinísticas

Uma MT consiste de uma fita formada por células em sequência, potencialmente infinita em ambas as direções e uma cabeça que lê o conteúdo de cada célula e guarda o estado atual. Uma função de transição indica, dado o estado atual e o símbolo sendo lido qual é a próxima operação: ir para esquerda ou ir para a direita e qual o novo símbolo na célula atual.

Formalmente temos que uma *Máquina de Turing Determinística*, ou simplesmente uma MT, é uma 7-upla $\langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ em que:

Q é um conjunto finito de *estados*,

Σ é o *alfabeto da entrada*,

Γ é o *alfabeto da fita* e $\Sigma \cup \{\textvisiblespace\} \subseteq \Gamma$,

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{E, D\}$ é a *função de transição*,

$q_0 \in Q$ é o *estado inicial*,

$q_a \in Q$ é o *estado de aceitação* e

$q_r \in Q$ é o *estado de rejeição*.

A cada passo, a MT está e, uma certa *configuração*. A configuração indica a sequência de símbolos antes da cabeça na fita e a sequência de símbolos depois da cabeça. Uma configuração pode ser representada por uma string da seguinte forma:

$$C = \omega_1 q \omega_2$$

As strings $\omega_1 \in \Gamma^*$ e $\omega_2 \in \Gamma^*$ indicam as sequências antes e depois da cabeça. O estado q indica o estado atual e o primeiro símbolo de ω dois é o símbolo sendo lido. Uma configuração em que $q = q_a$ é dita de *aceitação* e em que $q = q_r$ é de *rejeição*. Configurações de aceitação ou de rejeição são ditas *configurações de parada*. A *função de transição* define para cada configuração C_i qual o próxima configuração C_{i+1}.

Exemplo 4.1.1:

1. $uaq_ibv \Rightarrow uq_jacv$ se $\delta(q_i, b) = \langle q_j, c, E \rangle$
2. $uaq_ibv \Rightarrow uacq_jv$ se $\delta(q_i, b) = \langle q_j, c, D \rangle$
3. $uaq_ib \Rightarrow uabq_j\textvisiblespace$ se $\delta(q_i, b) = \langle q_j, b, D \rangle$
4. $q_iuab \Rightarrow q_j\textvisiblespace uab$ se $\delta(q_i, u) = \langle q_j, u, E \rangle$

4.2 Variantes de Máquinas de Turing

4.3 Máquinas de Acesso Aleatório (RAM)

4.4 O Problema da Parada

Capítulo 5

Complexidade Computacional

5.1 Complexidade de Tempo

5.2 NP-completude

5.3 Complexidade de Espaço

Apêndice A

Exercícios

A.1 Exercícios do Capítulo 2

Exercício 1: Para cada uma das seguintes expressões regulares dê uma string na linguagem representada por ela e uma string que não está nessa linguagem.

a) $(ab \cup \epsilon)b^*$

b) $(ab)^\star bb$

c) $(a \cup b)ba^\star$

d) $(aa)^\star(bb)^\star bb$

Exercício 2: Dê o diagrama de estado **e** a descrição formal de AFDs que reconheçam as seguintes linguagens:

a) $\{\omega \in \{0,1\}^* : \omega$ começa com 1 e termina com $0\}$

b) $\{\omega \in \{0,1\}^* : \omega$ contém a substring $000\}$

c) $\{0,1\}^* - \{\varepsilon\}$

d) $\{\omega \in \{0,1\}^* : \omega$ começa com 1 e tem comprimento par $\}$

Exercício 3: Dê o diagrama de estados de AFNs que reconheçam a linguagem:

a) $0^\star 1^\star$ com dois estados.

b) $(01)^\star$ com três estados.

c) $(0 \cup 1)$ com três estados.

d) $\{\omega \in \{0,1\}^\star : \omega$ começa com 0 e tem comprimento par ou começa com 1 e tem comprimento ímpar $\}$

Exercício 4: Seja $A = \{\omega \in \{0,1\}^* : \omega$ começa com 1 e termina com $0\}$ e $B = \{\omega \in \{0,1\}^* : \omega$ começa com 0 e tem comprimento par ou começa com 1 e tem comprimento ímpar $\}$. Desenhe o diagrama de estados para AFN que reconheça:

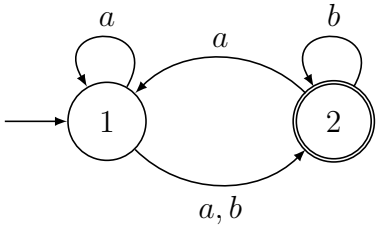
a) $A \circ B$

b) $B \circ A$

c) $A \cup B$

d) B^*

Exercício 5: Use o método visto em sala para desenhar o diagrama de estados AFD que reconheça a mesma linguagem que o seguinte diagrama AFN reconhece. Em seguida desenhe o mesmo AFD omitindo os estados supérfluos.



Exercício 6: Use o método visto em aula para encontrar uma expressão regular que reconhece a linguagem reconhecida pelo segundo AFD desenhado acima.

A.2 Exercícos do Capítulo 3

Exercício 7: O que é uma gramática ambígüa? A seguinte gramática $G = \langle V, \Sigma, R, E \rangle$, cujas regras R estão descritas a seguir, é ambígüa?

$$E \rightarrow E \wedge E | E \vee E | p | \neg p$$

Exercício 8: Desenhe o diagrama de estados de um autômato com pilha que reconhece a seguinte linguagem[1]:

$$A = \{\omega.\omega^R : \omega \in \{0,1\}^*\}$$

Exercício 9: Mostre que a linguagem do exercício anterior não é regular.

Exercício 10: Mostre uma GLC associada a cada uma das linguagens abaixo:

a) $\{\omega \in \{0,1\}^* : \omega$ possui pelo menos dois 1s$\}$

b) $\{\omega.\omega^R : \omega \in \{0,1\}^*\}$

c) $\{0^n 1^n : n \geq 0\}$

Exercício 11: Use o teorema visto em aula para construir um autômato com pilha a partir da gramática $G = \langle V, \Sigma, R, E \rangle$, cujas regras R estão descritas a seguir:

$$\begin{aligned} E &\rightarrow C \wedge C | C \\ C &\rightarrow L \vee L | L \\ L &\rightarrow p | \neg p \end{aligned}$$

Exercício 12: Mostre que a seguinte linguagem não é livre de contexto:

$$\{0^n 1^n 0^n 1^n : n \geq 0\}$$

[1] Lembre-se que ω^R é ω com os símbolos invertidos