

# **Introdução à Teoria da Computação: Linguagens Formais, Autômatos e Computabilidade**

**Benjamín René Callejas Bedregal**

*Grupo de Lógica, Linguagem, Informação, teoria e Aplicações – LoLITA  
Departamento de Informática e Matemática Aplicada – DIMAp  
Universidade Federal do Rio Grande do Norte – UFRN  
bedregal@dimap.ufrn.br*

**Benedito Melo Acióly**

*Departamento de Ciências Exatas - DCE  
Universidade Estadual do Sudoeste da Bahia - UESB  
bma@uesb.br*

**Aarão Lyra**

*Curso de Engenharia de Computação e Sistemas de Informação  
Universidade Potiguar - UnP  
aarao@unp.br*

Setembro, 2009

# Prefácio

Este livro apresenta um estudo sobre a teoria das linguagens formais e uma breve introdução à teoria da computabilidade, ambos conhecimentos básicos e indispensáveis para a formação de bachareles e engenheiros em computação. De fato atualmente a maioria dos cursos de ciência da computação, principalmente aqueles de instituições sérias, têm uma ou mais disciplinas onde essas teorias são abordadas. Assim, por exemplo, o POSCOMP, que é um exame nacional para avaliar os conhecimentos na área de Computação de candidatos a Programas de Pós-Graduação, tem estas matérias como parte dos conhecimentos avaliados.

Num mundo onde a tecnologia predomina, é fácil reduzir a computação ao desenvolvimento de software e hardware, mas certamente o mundo da computação é muito mais rico e complexo que isso. Principalmente ele se assenta em uma forte fundamentação matemática e teórica. Não é por nada que os primeiros cientistas a introduzirem conceitos e noções ligadas à computação eram oriundos da matemática, principalmente lógicos. É difícil dizer quando exatamente começou a computação, pois desde a Grécia antiga já existiam as noções de procedimentos efetivo e algoritmos (Por exemplo o algoritmo da divisão de Euclídes), mas certamente o ano de 1936 é um marco importante, por ser nesse ano que foram introduzidos os primeiros modelos formais de computador, surgindo assim o inicio da teoria da computabilidade. Já a teoria das linguagens formais surge a fines da década de 1950, num intento de desenvolver teorias para linguagens naturais, e desde então tem sido motivo de extensas pesquisas com aplicações, por exemplo, para processamento de linguagens naturais como para linguagens de computação. Em [Gop06] são apresentadas diversas aplicações de teoria de autômatos tanto para engenharia quanto para ciência da computação. Aqui veremos as principais classes de linguagens formais e suas propriedades, dando assim um embasamento básico desta teoria.

O objetivo deste texto é introduzir ao estudante de computação nestas fascinantes teorias formais que acreditamos lhe serão de muito valia na melhor compreensão da ciência da computação como um todo.



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Conceitos Matemáticos Básicos . . . . .	3
1.1.1	Conjuntos . . . . .	3
1.1.2	Funções e Relações . . . . .	5
1.1.3	Grafos e Árvores . . . . .	7
1.2	Noções de Linguagens, Gramáticas e Autômatos . . . . .	8
1.2.1	Linguagens Formais . . . . .	8
1.2.2	Gramáticas . . . . .	14
1.2.3	Autômatos . . . . .	17
1.3	Exercícios . . . . .	18
<b>2</b>	<b>Autômatos Finitos</b>	<b>21</b>
2.1	Autômatos Finitos Determinísticos . . . . .	21
2.2	Linguagens e AFD's . . . . .	23
2.3	Linguagens Regulares . . . . .	27
2.4	Autômatos Finitos Não-Determinísticos . . . . .	28
2.5	Equivalência entre AFD's e AFN's . . . . .	36
2.6	Algoritmo da Minimização de Estados . . . . .	44
2.7	Exercícios . . . . .	49
<b>3</b>	<b>Expressões e Gramáticas Regulares</b>	<b>57</b>
3.1	Expressões Regulares: Sintaxe . . . . .	57
3.2	Expressões Regulares: Semântica . . . . .	58
3.3	Equivalência entre Expressões Regulares . . . . .	61
3.4	Conexão entre Expressões Regulares e Linguagens Regulares . . . . .	62
3.4.1	As Expressões Regulares Denotam Linguagens Regulares . . . . .	62

## CONTEÚDO

---

3.4.2	Linguagens Regulares para Expressões Regulares . . . . .	65
3.5	Gramáticas Regulares . . . . .	69
3.6	Conexão entre Gramáticas Regulares e Linguagens Regulares . . . . .	72
3.6.1	Gramáticas Lineares à Direita Geram Linguagens Regulares . . . . .	72
3.6.2	Linguagens Regulares são Geradas por Gramáticas Lineares à Direita . . . . .	74
3.6.3	Equivalência entre Linguagens Regulares e Gramáticas Regulares . . . . .	76
3.7	Exercícios . . . . .	78
<b>4</b>	<b>Propriedades das Linguagens Regulares</b>	<b>83</b>
4.1	Propriedades de Fecho de Linguagens Regulares . . . . .	83
4.2	Questões Elementares sobre Linguagens Regulares . . . . .	88
4.3	Identificando Linguagens Não Regulares . . . . .	90
4.3.1	Usando o Princípio da Casa de Pombos . . . . .	90
4.3.2	Lema do Bombeamento para Linguagens Regulares . . . . .	91
4.4	Exercícios . . . . .	96
<b>5</b>	<b>Gramáticas Livres do Contexto</b>	<b>101</b>
5.1	Gramáticas Livres do Contexto . . . . .	101
5.1.1	Derivações Mais à Esquerda e Mais à Direita . . . . .	103
5.1.2	Árvores de Derivação . . . . .	104
5.1.3	Relação entre Formas Sentenciais e Árvores de Derivação . . . . .	105
5.2	Ambigüidade em Gramáticas e Linguagens . . . . .	107
5.3	Algoritmos de Análise de Pertinência . . . . .	110
5.3.1	Método de Análise de Pesquisa Exaustiva . . . . .	111
5.3.2	Métodos de Análise mais Eficientes . . . . .	113
5.4	Gramáticas Livres do Contexto e Linguagens de Programação . . . . .	115
5.5	Simplificações de Gramáticas Livres do Contexto . . . . .	116
5.5.1	Regra Geral de Substituição . . . . .	116
5.5.2	Remoção de Produções Esquerda-Recursivas . . . . .	118
5.5.3	Remoção de Produções Inúteis . . . . .	120
5.5.4	Remoção de $\lambda$ -Produções . . . . .	123
5.5.5	Remoção de Produções Unitárias . . . . .	125
5.6	Formas Normais . . . . .	128
5.6.1	Forma Normal de Chomsky . . . . .	128
5.6.2	Forma Normal de Greibach . . . . .	132
5.7	Exercícios . . . . .	135

<b>6 Autômatos com Pilha</b>	<b>141</b>
6.1 Autômatos com Pilha Determinísticos (APD) . . . . .	141
6.2 Autômato com Pilha Não-Determinístico (APN) . . . . .	144
6.2.1 A Linguagem Aceita por um Autômato com Pilha Não-Determinístico . . . . .	147
6.3 Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto . . . . .	151
6.3.1 Transformando Gramáticas Livre do Contexto para Autômatos com Pilha Não-Determinísticos . . . . .	151
6.3.2 Transformando Autômatos com Pilha Não-Determinísticos em Gramáticas Livre do Contexto . . . . .	155
6.4 $\lambda$ -Transições vs Não-determinismo Explícito . . . . .	160
6.4.1 Medidas de Não-Determinismo Implícito em APN's Livres de Não-determinismo Explícito . . . . .	163
6.5 Exercícios . . . . .	165
<b>7 Propriedades das Linguagens Livres do Contexto</b>	<b>169</b>
7.1 O Lema do Bombeamento para Linguagens Livres do contexto . . . . .	169
7.2 Propriedades de Fecho para Linguagens Livres do Contexto . . . . .	172
7.3 Propriedades Decidíveis de Linguagens Livres do Contexto . . . . .	173
7.4 Exercícios . . . . .	175
<b>8 Máquinas de Turing</b>	<b>177</b>
8.1 A Máquina de Turing Padrão . . . . .	177
8.1.1 Definição de Máquina de Turing . . . . .	178
8.1.2 Máquinas de Turing como Reconhecedoras de Linguagens . . . . .	183
8.1.3 Máquinas de Turing como Tradutores (Computadores) . . . . .	186
8.2 Combinação de Máquinas de Turing para Realizar Tarefas Complicadas . . . . .	189
8.2.1 Diagramas de Blocos . . . . .	190
8.2.2 Macroinstrução . . . . .	191
8.2.3 Subprogramas . . . . .	192
8.3 Procedimentos efetivos e Algoritmos . . . . .	194
8.4 Tese de Turing . . . . .	195
8.5 Ligeiras Variações da Máquina de Turing Padrão . . . . .	197
8.5.1 Máquinas de Turing com uma Opção de Permanência . . . . .	198
8.5.2 Máquinas de Turing com uma Fita Semi-infinita . . . . .	198

## CONTEÚDO

---

8.5.3	Máquinas de Turing Off-line . . . . .	199
8.5.4	Máquinas de Turing com Múltiplas Fitais . . . . .	199
8.5.5	Máquinas de Turing Multidimensional . . . . .	200
8.5.6	Máquinas de Turing Não-Determinísticas . . . . .	201
8.6	Máquina de Turing Universal . . . . .	202
8.7	Algumas Limitações da Tese de Church-Turing . . . . .	205
8.8	Exercícios . . . . .	207
<b>9</b>	<b>Linguagens Recursivamente Enumeráveis, Recursivas e Sensíveis ao Contexto</b>	<b>213</b>
9.1	Linguagens Recursivas e Recursivamente Enumeráveis . . . . .	213
9.2	Linguagens que Não são Recursivamente Enumeráveis . . . . .	215
9.3	Gramáticas Irrestritas . . . . .	218
9.4	Linguagens Sensíveis ao Contexto . . . . .	223
9.4.1	Gramáticas Sensíveis ao Contexto . . . . .	223
9.4.2	Forma Normal Sensível ao Contexto . . . . .	225
9.4.3	Autômatos Limitados Linearmente . . . . .	230
9.4.4	Relação entre Linguagens Recursivas e Linguagens Sensíveis ao Contexto	233
9.4.5	Equivalência entre ALL's e Gramáticas Sensíveis ao Contexto . . . . .	234
9.4.6	Propriedades de Fecho das Linguagens Sensíveis ao Contexto . . . . .	235
9.5	Exercícios . . . . .	237
<b>10</b>	<b>Hierarquia de Chomsky e Classes de Complexidades de Linguagens</b>	<b>239</b>
10.1	A Hierarquia de Chomsky . . . . .	240
10.2	Complexidade Computacional . . . . .	242
10.2.1	Medida e Complexidade . . . . .	242
10.2.2	Famílias e Classes de Complexidade . . . . .	243
10.3	Classificação da Complexidade e a Hierarquia de Chomsky . . . . .	246
10.4	A Classe de Complexidade <i>P</i> e <i>NP</i> . . . . .	248
10.5	Exercícios . . . . .	251

## CONTEÚDO

---

<b>11 Limites da Computação Algorítmica: Problemas Indecidíveis</b>	<b>253</b>
11.1 Computabilidade e Decibilidade . . . . .	253
11.2 O Problema da Parada para Máquinas de Turing . . . . .	254
11.3 Redução de um Problema Indecidível ao Problema da Parada . . . . .	258
11.4 Problemas Indecidíveis para Linguagens Recursivamente Enumeráveis . . . . .	260
11.5 O Problema da Correspondência de Post . . . . .	263
11.6 Problemas Indecidíveis para Linguagens Livres do Contexto . . . . .	268
11.7 Semi-decibilidade, Co-semi-decibilidade e Divergência . . . . .	271
11.8 Exercícios . . . . .	272
<b>A Soluções de Alguns dos Exercícios Propostos</b>	<b>273</b>
A.1 Soluções de Exercícios do Capítulo 1 . . . . .	273
A.2 Soluções de Exercícios do Capítulo 2 . . . . .	274
A.3 Soluções de Exercícios do Capítulo 3 . . . . .	295
A.4 Soluções de Exercícios do Capítulo 4 . . . . .	304
A.5 Soluções de Exercícios do Capítulo 5 . . . . .	315
A.6 Soluções de Exercícios do Capítulo 6 . . . . .	329
A.7 Soluções de Exercícios do Capítulo 7 . . . . .	337
A.8 Soluções de Exercícios do Capítulo 8 . . . . .	343

## **CONTEÚDO**

---

# Listas de Figuras

1.1	O diagrama de um grafo dirigido.	8
1.2	Exemplo de árvore	9
1.3	Representação esquemática de um autômato.	18
2.1	Exemplo de AFD.	22
2.2	Exemplo de AFD.	23
2.3	Exemplo de AFD.	25
2.4	Exemplo de AFD.	26
2.5	Exemplo de AFD.	26
2.6	Exemplo de AFD.	27
2.7	Exemplo de AFD.	28
2.8	Exemplo de AFN.	29
2.9	Exemplo de AFN.	30
2.10	Exemplo de um AFN com $\lambda$ -transições.	30
2.11	Exemplo de um AFN com $\lambda$ -transições.	32
2.12	Exemplo de um AFN com $\lambda$ -transições.	33
2.13	afd equivalente ao AFN do exemplo 2.4.4.	36
2.14	afd equivalente à AFN da figura 2.9	37
2.15	Exemplo de AFN.	38
2.16	afd equivalente ao AFN da figura 2.15.	39
2.17	Exemplo de AFN.	41
2.18	AFD equivalente ao AFN da figura 2.17.	42
2.19	Exemplo de AFN.	43
2.20	AFD equivalente ao AFN da figura 2.19.	44
2.21	AFD com estados equivalentes.	45

## LISTA DE FIGURAS

---

2.22 AFD $M_{/\equiv}$ obtido a partir do AFD da Figura 2.21.	47
2.23 AFN do exercício 20	51
2.24 AFN do exercício 21	52
2.25 AFN do exercício 21	52
2.26 AFN do exercício 21	53
2.27 AFN do exercício 21	54
2.28 AFN do exercício 21	55
2.29 AFD não mínimo do exercício 27	56
2.30 AFD não mínimo que reconhece os números em binário múltiplos de 6.	56
3.1 AFN's que aceitam as linguagens regulares $\emptyset$ , $\{\lambda\}$ e $\{a\}$ , respectivamente.	63
3.2 Representação esquemática de um AFN que reconhece $L(r)$ .	63
3.3 Autômato para $L(r_1 + r_2)$ .	64
3.4 Autômato para $L(r_1 r_2)$ .	64
3.5 Autômato para $L(r_1^*)$ .	64
3.6 $M_1$ reconhece $L(aa)$ e $M_2$ reconhece $L(bb)$ .	66
3.7 $M_3$ reconhece $L(aa + bb)$ .	66
3.8 $M_4$ reconhece $L((aa + bb)^*)$ .	66
3.9 $M_5$ reconhece $L(ba)$ .	66
3.10 $M_6$ reconhece $L((aa + bb)^*ba)$ .	67
3.11 AFD para o exemplo 3.4.4.	68
3.12 Representam $V_i \rightarrow a_1 a_2 \dots a_m V_j$ e $V_i \rightarrow a_1 a_2 \dots a_m$ , respectivamente.	74
3.13 AFN resultante de uma gramática linear à direita.	74
3.14 Autômato finito determinístico do exercício 8.	80
3.15 AFD's do exercício 12	82
4.1 AFD $M_1$	85
4.2 AFD $M_2$	85
4.3 AFD's que reconhecem as linguagens $\mathcal{L}_1 = L((a + b)^*aa(a + b)^*)$ e $\mathcal{L}_2 = L((a + b)^*bbb(a + b)^*)$ .	86
4.4 AFD que reconhece $L((a + b)^*aa(a + b)^*) \cap L((a + b)^*bbb(a + b)^*)$ .	87
4.5 AFD do exercício 2.	96
5.1 Parte da árvore de derivação para a produção $A \rightarrow abABC$ .	104

---

## LISTA DE FIGURAS

5.2 Uma árvore de derivação parcial. . . . .	105
5.3 Uma árvore de derivação. . . . .	106
5.4 Exemplo de árvores de derivação diferentes para uma mesma cadeia. . . . .	107
5.5 Exemplo de árvores de derivação diferentes para uma mesma cadeia. . . . .	108
5.6 Exemplo de árvore de derivação numa gramática, não ambígua, equivalente à gramática ambígua do exemplo anterior. . . . .	109
5.7 Exemplo de grafo de dependência . . . . .	121
5.8 Árvore de derivação parcial . . . . .	122
5.9 Exemplo de grafo de dependência para remoção de produções unitárias. . . . .	127
6.1 Representação esquemática de um autômato com pilha. . . . .	142
6.2 Esquema básico da transformação de um APN em um APN livre de não-determinismo explícito. . . . .	162
8.1 Representação esquemática de uma máquina de Turing. . . . .	178
8.2 A situação (a) antes do movimento e (b) após. . . . .	179
8.3 Parte da execução de uma máquina de Turing. . . . .	180
8.4 Configuração arbitrária de uma máquina de Turing . . . . .	181
8.5 Exemplo de combinação de máquinas de Turing . . . . .	190
8.6 Divisão da fita para simulação de subprogramas em máquinas de Turing. . . . .	193
8.7 Conteúdo da fita para a multiplicação usando a idéia de subprograma. . . . .	193
8.8 Fita ilimitada somente à direita. . . . .	198
8.9 Representação esquemática de uma máquina de Turing off-line. . . . .	199
8.10 Representação esquemática de uma máquina de Turing com duas fitas. . . . .	199
8.11 Esquema de uma máquina de Turing bi-dimensional. . . . .	200
8.12 Representação esquemática de uma máquina de Turing universal. . . . .	203
9.1 procedimento de enumeração das computações de uma máquina de Turing $M$ . .	214
10.1 Hierarquia de Chomsky original. . . . .	240
10.2 Hierarquia de Chomsky Estendida. . . . .	241
10.3 Configuração de uma Máquina de Turing de duas fitas que reconhece a linguagem $a^n b^n$ . . . . .	244
11.1 Diagrama de blocos para uma suposta máquina de Turing que resolve o problema da parada. . . . .	256

## LISTA DE FIGURAS

---

11.2 Diagrama de blocos para uma variante da suposta máquina de Turing que resolve o problema da parada . . . . .	256
11.3 Algoritmo para o problema da parada usando um eventual algoritmo para o problema da parada com a fita em branco. . . . .	259
11.4 Algoritmo para o problema da parada com a fita em branco baseado num algoritmo para computar $f(n)$ . . . . .	261
11.5 Algoritmo de pertinência para linguagens recursivamente enumeráveis. . . . .	262
11.6 Uma solução para o problema da parada baseada num algoritmo que diz se uma linguagem é finita ou não. . . . .	262
11.7 Exemplo de solução-cp . . . . .	264
11.8 Exemplo de solução-CPM (parcial) . . . . .	265
11.9 Exemplo de solução-CPM (continuação) . . . . .	266
11.10 Algoritmo de pertinência para uma gramática irrestrita. . . . .	267
11.11 Algoritmo CPM. . . . .	268
11.12 Algoritmo CP. . . . .	270
 A.1 AFD para o exercício 1. . . . .	275
A.2 AFD para o exercício 3. . . . .	275
A.3 AFD para o exercício (4b). . . . .	276
A.4 AFD para o exercício (4c). . . . .	276
A.5 AFD para o exercício (4d). . . . .	277
A.6 AFD para o exercício (4d). . . . .	278
A.7 AFD para o exercício (5a). . . . .	279
A.8 AFD para o exercício (5c). . . . .	279
A.9 AFD para o exercício (6a). . . . .	280
A.10 AFD para o exercício (6b). . . . .	280
A.11 AFD para o exercício (7a). . . . .	280
A.12 AFD para o exercício (7c). . . . .	281
A.13 AFD para o exercício (7e). . . . .	281
A.14 AFD para o exercício (7g). . . . .	281
A.15 AFD para o exercício (7i). . . . .	282
A.16 AFD para o exercício (7k). . . . .	283
A.17 AFD para o exercício (9a). . . . .	284
A.18 AFD para o exercício (9c). . . . .	285

---

## LISTA DE FIGURAS

A.19 AFD para o exercício (9e).	286
A.20 AFD para o exercício (9g).	286
A.21 AFD para o exercício (9i).	287
A.22 AFD para o exercício 15.	287
A.23 AFN para o exercício 17.	288
A.24 AFN para o exercício 19.	288
A.25 AFD equivalente ao AFN da figura A.24.	289
A.26 AFD equivalente ao AFD da figura A.25.	289
A.27 AFD equivalente ao AFN da figura 2.24.	290
A.28 AFN para o exercício (25a).	291
A.29 AFN para o exercício (25c).	291
A.30 AFN para o exercício (25e).	292
A.31 AFD não mínimo do exercício 27	293
A.32 AFD $M_{/\equiv}$ obtido a partir do AFD da Figura A.31.	294
A.33 AFN que reconhece a linguagem $L(aa^*(a+b))$	297
A.34 AFN que reconhece a linguagem $L((ab+b)^*(a+\lambda))^*$	298
A.35 AFN que reconhece a linguagem $L(aa^*bb^*aa^*)$	298
A.36 AFN que reconhece a linguagem $L(a^*(b^bb)^*aa)^*$	299
A.37 AFN que reconhece a linguagem gerada pela gramática regular do exercício (13a).	302
A.38 AFN que reconhece a linguagem gerada pela gramática regular do exercício (13c).	303
A.39 AFD que reconhecem as linguagens $\mathcal{L}_a$ , $\mathcal{L}_b$ e $\mathcal{L}_c$	351
A.40 Uma árvore de derivação para a cadeia $aabbba$ .	352
A.41 Grafo de dependência entre variáveis de uma gramática livre do contexto.	352

Dedicatória do primeiro autor

**Com todo meu amor aos meus filhos:  
Berta Letícia, Natália Tainá e Pablo Labán,  
pelas muitas alegrias e amor  
que me proporcionam a cada dia.**

# Agradecimentos

Durante vários anos, os autores (e alguns colegas) usaram versões prévias deste texto como livro base para lecionar as disciplinas de “Teoria da Computação” e “Linguagens Formais”, nas universidades: Universidade Federal de Pernambuco, Universidade Federal do Rio Grande do Norte, Universidade Estadual do Sudoeste da Bahia e Potiguar. Agradecemos sinceramente aos alunos dessas turmas, pois com suas dúvidas e questionamentos ajudaram a refinar este texto até chegar à atual versão.

Nossos agradecimentos ao corpo docente do DIMAp-UFRN, DCE-UESB e da UnP pelo seu constante incentivo para publicar esta obra

Finalmente, agradecemos particularmente ao nosso colega e amigo Regivan Hugo Nunes Santiago pelos seus comentários e correções do texto.

# Capítulo 1

## Introdução

A importância da teoria para a prática, em qualquer ciência, seja factual ou exata, é imensa. Para elucidar a correlação entre ambos aspectos de uma ciência foram formuladas algumas frases consagradas tais como: “A teoria é a luz da prática” e “A prática sem a teoria é cega, mas a teoria sem a prática é estéril”. Como é de supor, a ciência da computação, não poderia ficar alheia a esta interação e, portanto, uma teoria para esta ciência se faz necessário para “iluminar o caminho” dos cientistas da computação (práticos), dos engenheiros em computação, dos analistas de sistemas, enfim de todos os profissionais que usem a computação como objeto de estudo ou de trabalho. Assim, antes de descrever o que seria uma teoria para a ciência da computação, devemos esclarecer o que entendemos por ciência da computação ou, simplesmente, por computação.

Entendendo por computação tudo o que um computador pode realizar, devemos primeiro entender o que um computador é, cuja resposta poderia ser dada em termos de hardware e tecnologia. Mas, devemos de ter cuidado, para não nos limitar à tecnologia do momento, pois nessa definição de computador devem coexistir os primeiros computadores, calculadoras, super computadores e futuros computadores. Ou seja, seria necessário unificar características essenciais e comuns a todos os computadores, de tal modo a distinguir um computador de outros tipos de hardware, tais como elevadores, máquinas de venda de refrigerante, toca CD, etc. Isto nos levará, irremediavelmente, a definir uma noção abstrata de computador, onde ele possa ser melhor descrito em termos do que ele faz.

A teoria nos fornece conceitos e princípios para nos ajudar a entender a natureza geral da *ciência do computador*. O campo dessa disciplina inclui um vasto leque de tópicos especiais, desde projetos de máquinas até programação. O uso dos computadores no mundo real envolve uma riqueza de detalhes específicos que devem ser aprendidos para uma aplicação com sucesso. Isto faz com que a ciência da computação seja diversificada e ampla. Apesar dessa diversidade, existem alguns princípios básicos comuns. Para estudar esses princípios básicos, construiremos modelos abstratos de computadores e computação. Esses modelos contém as características importantes que são comuns tanto ao hardware quanto ao software, essenciais a muitos conceitos especiais e complexos encontrados quando se trabalha com computadores. Mesmo que esses modelos sejam muito simples para serem aplicados imediatamente nas situações do mundo real, o entendimento que ganhamos em estudá-los nos fornece fundamentos sobre os quais o desenvolvimento específico é baseado. Esta abordagem não é exclusividade da ciência da computação. A construção de modelos é essencial em qualquer disciplina científica, e a utilidade de

## Capítulo 1. Introdução

---

uma disciplina depende freqüentemente da teoria e de leis simples, ainda que poderosas. Além do mais, as idéias que discutiremos têm algumas aplicações imediatas e importantes. Os campos de *projetos digitais*, *linguagens de programação* e *compiladores* são os exemplos mais óbvios, existem porém muitos outros (Protocolos de comunicação, projetos de hardwares digitais, esquemas XML, criptografia, processamento digital de imagens, etc.).

Este texto é o resultado de diversos cursos ministrados pelos autores para a disciplina de “teoria da computação” dos cursos de graduação em *Bacharelado em Ciência da Computação* da Universidade Federal do Rio Grande do Norte - UFRN, da Universidade Federal de Pernambuco - UFPE e da Universidade Estadual do Sudoeste da Bahia - UESB e no curso de *Engenharia da Computação* da UFRN. Como esta disciplina é pre-requisito da disciplina “compiladores”, enfatizamos os conceitos de linguagens formais, com suas abordagens através de autômatos e de gramáticas. Mas, como principalmente é um curso de “teoria da computação”, também estudamos a noção de computabilidade e consideramos uma breve discussão de complexidade computacional.

Os tópicos apresentados neste texto são importantes para os estudantes de informática, seja de ciências ou de engenharia, pois os colocam diante de questões profundas de natureza computacional e de conhecimentos que não ficarão rapidamente obsoletos, pois não dependem da tecnologia. Assim, este texto nos permite vislumbrar o poder das ferramentas matemáticas e dos métodos formais para modelar fenômenos da computação. No entanto, é óbvio que este texto não é exaustivo no que tange à teoria da computação, pois só se direciona a modelar linguagens formais e computabilidade clássica, deixando de lado certos aspectos específicos, como por exemplo computação concorrente e distribuída, especificação formal, computações numéricas envolvendo números reais, etc. Mesmo no caso de linguagens formais o texto está num nível de um curso de graduação, mas certamente a área é muito mais ricas e ainda há muitas coisas a serem exploradas. Esse estudo dos fenômenos da computação através de modelos matemáticos ou formais é que ergue a computação ao patamar de uma ciência em lugar de uma mera tecnologia.

Neste texto, estudaremos diversas classes de linguagens (formais), mas daremos ênfase especialmente a quatro delas:

1. Linguagens regulares,
2. Linguagens livres do contexto,
3. Linguagens sensíveis ao contexto e
4. Linguagens recursivamente enumeráveis.

Cada uma dessas classes de linguagens será abordada através de autômatos, que são modelos matemáticos de classes de computadores digitais, e através de gramáticas, que são, basicamente, um conjunto de regras que dizem como construir palavras válidas da linguagem. No entanto, também veremos superficialmente três outras classes de linguagens formais: Linguagens lineares, linguagens livres do contexto determinísticas e linguagens recursivas.

Exploramos o mais complexo desses autômatos, e observamos que ele não só tem capacidade para reconhecer linguagens formais, mas também pode transformar entradas em saídas, ou seja

realizar computações como qualquer computador real faz. Este é o ponto de partida para se introduzir a noção de computabilidade e estabelecer os limites do mundo da computação.

Finalmente, com um conhecimento do que é e o que não é computável, podemos nos preocupar com analisar a qualidade das soluções, isto é, não só é importante saber se um determinado problema admite uma solução implementável num computador, mas se essa solução vai nos ser útil (se vai ser realizada em um tempo razoável, ou ainda se ela ocupa espaço de memória que dispomos). Ou seja, agora podemos nos preocupar com a complexidade computacional das soluções. Para isso, introduzimos algumas medidas de complexidade baseadas no tempo de execução de um algoritmo e no espaço usado por ele.

Neste capítulo, veremos alguns conceitos matemáticos básicos, fundamentais para o entendimento do texto. Além do mais, veremos de modo geral e superficial as três visões (linguagem, autômato e gramática) que adotaremos neste texto, para estudar o poder computacional dos nossos modelos.

## 1.1 Conceitos Matemáticos Básicos

### 1.1.1 Conjuntos

Um **conjunto** é uma coleção de objetos (seus elementos) distintos. Descrevemos um conjunto enumerando um a um seus elementos em alguma ordem adequada ou descrevendo uma propriedade que só os elementos do conjunto possuem ou satisfazem. Em geral, denotamos conjuntos por letras maiúsculas com ou sem índices, por exemplo  $A, B, C, A_1, B_k$ , etc. Se  $a$  é um elemento de um conjunto  $A$ , dizemos que  $a$  pertence a  $A$ , denotado por  $a \in A$ . Se vários elementos, digamos  $a, b$  e  $c$ , pertencem ao conjunto  $A$  denotamos-lo por  $a, b, c \in A$ .

O conjunto que não tem elementos é denominado **conjunto vazio**, denotado por  $\emptyset$ . O conjunto vazio pode ser especificado por uma propriedade que nenhum elemento satisfaz. Por exemplo,  $\emptyset = \{x/x \neq x\}$ . O conjunto cujos elementos são todos objetos físicos ou mentais que existem, existiram ou existirão no universo é denominado **conjunto universo** e o denotamos por  $U$ . Em algumas situações são considerados universos particulares, por exemplo o conjunto dos seres humanos.

Um conjunto  $A$  é um **subconjunto** de um conjunto  $B$  ou  $A$  está contido ou é igual a  $B$ , denotado por  $A \subseteq B$ , se todos os elementos de  $A$  são elementos de  $B$ , isto é, se para todo  $x \in A$ ,  $x \in B$ . Os conjuntos  $A$  e  $B$  são **iguais**, denotado por  $A = B$ , se  $A \subseteq B$  e  $B \subseteq A$ . Dizemos que um conjunto  $A$  é um **subconjunto próprio** de  $B$ , denotado por  $A \subset B$ , se  $A \subseteq B$  e  $A$  não é igual a  $B$ , ou seja se  $A \subseteq B$  e existe um elemento de  $B$  que não está em  $A$ .

**Proposição 1.1.1** *Sejam  $A, B$  e  $C$  conjuntos quaisquer. Então*

1.  $\emptyset \subseteq A$ ,
2. Se  $A \subset B$  então  $A \subseteq B$ ,
3. Se  $A \subseteq B$  e  $B \subseteq C$  então  $A \subseteq C$ ,

## 1.1. Conceitos Matemáticos Básicos

---

4. Se  $A \subset B$  e  $B \subset C$  então  $A \subset C$ , e

5. Se  $A \subseteq B$  e  $B \subset C$  então  $A \subset C$ .

A seguir definiremos algumas operações básicas sobre conjuntos. Sejam  $A$ ,  $B$  e  $C$  três conjuntos quaisquer.

- União:  $A \cup B = \{x / x \in A \text{ ou } x \in B\}$ ,
- Intersecção:  $A \cap B = \{x / x \in A \text{ e } x \in B\}$ ,
- Diferença:  $A - B = \{x / x \in A \text{ e } x \notin B\}$ , e
- Complementação:  $\overline{A} = \{x / x \notin A\}$ .

**Proposição 1.1.2** Sejam  $A$  e  $B$  conjuntos quaisquer. Então

$$1. A \cup B = B \cup A,$$

$$2. A \cap B = B \cap A,$$

$$3. A \cup (B \cap C) = (A \cup B) \cap (A \cup C),$$

$$4. A \cap (B \cup C) = (A \cap B) \cup (A \cap C),$$

$$5. A \cup \emptyset = A - \emptyset = A,$$

$$6. A \cap \emptyset = \emptyset,$$

$$7. \overline{\emptyset} = U,$$

$$8. \overline{\overline{A}} = A,$$

$$9. \overline{A \cup B} = \overline{A} \cap \overline{B},$$

$$10. \overline{A \cap B} = \overline{A} \cup \overline{B},$$

$$11. \text{Se } A \subseteq B \text{ então } A \cup B = B, A \cap B = A \text{ e } \overline{B} \subseteq \overline{A}.$$

Se  $A$  e  $B$  são conjuntos sem nenhum elemento em comum, isto é, se  $A \cap B = \emptyset$ , então  $A$  e  $B$  são denominados **conjuntos disjuntos**.

Um conjunto é **finito** se contém um número finito de elementos. Caso contrário, ele é **infinito**. O tamanho ou **cardinalidade** de um conjunto finito é o número de elementos que ele contém, denotado por  $|A|$ .

Seja  $\{A_i\}_{i \in I}$  uma **família de conjuntos** indexada pelo conjunto de índices  $I$ . Então

1. a união da família é definida por

$$\bigcup_{i \in I} A_i = \{x / \text{ existe } i \in I \text{ tal que } x \in A_i\},$$

2. a intersecção da família é definida por

$$\bigcap_{i \in I} A_i = \{x / \text{ para todo } i \in I \text{ tal que } x \in A_i\},$$

3. se  $I = \{1, 2, \dots, n\}$  então denotamos a união e intersecção da família  $\{A_i\}_{i \in I}$  por

$$\bigcup_{i=1}^n A_i \quad \text{e} \quad \bigcap_{i=1}^n A_i.$$

Um dado conjunto pode ter muitos subconjuntos. O conjunto de todos os subconjuntos de um conjunto  $A$  é chamado o **conjunto potência** ou **conjunto das partes** de  $A$ , denotado por  $2^A$  ou  $\wp(A)$ .

**Exemplo 1.1.3** Se  $A$  é o conjunto  $\{a, b, c\}$ , então seu conjunto potência é

$$2^A = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Observe que neste caso  $|A| = 3$  e  $|2^A| = 2^{|A|} = 2^3 = 8$ . Generalizando, se  $A$  é um conjunto finito então  $|2^A| = 2^{|A|}$ . Esta propriedade deu origem ao nome de conjunto potência.

### 1.1.2 Funções e Relações

Uma **função parcial** ou simplesmente **função** é uma regra que associa a certos elementos de um conjunto um único elemento de outro conjunto. Se  $f$  denota uma função, então o primeiro conjunto é chamado **conjunto de partida** e o segundo conjunto é chamado **conjunto de chegada**. Escreveremos  $f : A \rightarrow B$ , para indicar que  $f$  é uma função cujo conjunto de partida é  $A$  e cujo conjunto de chegada é  $B$ . Eventualmente, teremos casos em que alguns elementos do conjunto de partida não tenham associado nenhum elemento do conjunto de chegada através da função  $f$ . O subconjunto de  $A$  cujos elementos são associados a elementos de  $B$  é chamado **domínio** de  $f$ , denotado por  $\text{dom}(f)$ . Analogamente, o subconjunto de  $B$ , aos quais aqueles elementos de  $A$  foram associados, é chamado **imagem** ou **contradomínio** de  $f$  e o denotamos por  $\text{img}(f)$ . Se o domínio de  $f$  é todo  $A$ , dizemos que  $f$  é uma **função total** sobre  $A$ , caso contrário dizemos que  $f$  é uma **função parcial não total**.

No cotidiano sempre estamos relacionando objetos de um mesmo tipo ou de diferentes tipos. Por exemplo, a relação “pertencer à mesma família”, relaciona pessoas que são de uma mesma família e a relação “ser dono” relaciona pessoas com objetos. Assim, uma relação exige que exista uma conexão entre as coisas que se relacionam. Esta noção de conexão de um objeto  $a$  com um objeto  $b$ , pode ser formalmente colocado introduzindo a noção de *par ordenado*,  $(a, b)$ . A noção de par ordenado tem a seguinte propriedade fundamental:

$$(x_1, x_2) = (y_1, y_2) \text{ se, e somente se, } x_1 = y_1 \text{ e } x_2 = y_2. \quad (1.1)$$

## 1.1. Conceitos Matemáticos Básicos

---

Ou seja nos pares ordenados a ordem em que são dispostos os elementos é importante. Colocado dessa maneira, aparentemente, não podemos pensar num par ordenado como um conjunto, mas poderíamos encarar um **par ordenado**,  $(a, b)$ , como uma abreviação do conjunto  $\{a, \{a, b\}\}$ . Esta visão conjuntista de par ordenado satisfaaz a propriedade fundamental 1.1. O conjunto de todos os pares ordenados possíveis de ser construídos entre um conjunto  $A$  e um conjunto  $B$ , é denominado **produto cartesiano** de  $A$  com  $B$ , cuja denotação é  $A \times B$ . Assim,

$$A \times B = \{(a, b) / a \in A \text{ e } b \in B\}.$$

A noção de par ordenado pode ser estendida à noção de  $n$ -tuplas ordenadas, isto é, em vez de considerar elementos de dois conjuntos, considerar elementos de  $n$  conjuntos. Denotaremos uma  $n$ -tupla por  $(a_1, \dots, a_n)$ . Analogamente, podemos considerar o produto cartesiano dos conjuntos  $A_1, \dots, A_n$ , denotado por  $A_1 \times \dots \times A_n$ , por

$$A_1 \times A_2 \times \dots \times A_n = \{(x_1, x_2, \dots, x_n) / x_i \in A_i, \text{ para cada } 1 \leq i \leq n\}.$$

Usaremos a notação  $A^n$  para descrever o produto cartesiano de  $A$  com ele mesmo  $n$ -vezes, ou seja

$$A^n = \underbrace{A \times \dots \times A}_{n-\text{vezes}}.$$

Assim, uma **relação**  $r$  entre os elementos de um conjunto  $A$  com os elementos de um conjunto  $B$  é, em suma, um subconjunto do produto cartesiano de  $A$  com  $B$ , ou seja,  $r \subseteq A \times B$ . Analogamente, uma função  $f : A \longrightarrow B$  pode ser vista como um subconjunto do produto cartesiano de  $A$  com  $B$ , isto é,  $f \subseteq A \times B$ , onde cada  $x_i$  pode ocorrer no máximo uma vez como o primeiro elemento de um par ordenado em  $f$ . Esta maneira de encarar funções é chamada **gráfico da função**  $f$ . Claramente, as relações são mais gerais que as funções: numa função cada elemento do domínio tem associado a ele exatamente um elemento no contradomínio; numa relação podem existir vários de tais elementos no contradomínio.

Uma relação especialmente importante é uma **relação de equivalência**, que é uma generalização do conceito de igualdade (identidade). Para indicar que um par  $(x, y)$  é equivalente, escrevemos

$$x \equiv y$$

Uma relação, denotada por  $\equiv$ , é considerada uma equivalência se ela satisfaz três regras:

- Reflexividade:  $x \equiv x$  para todo  $x$ ,
- Simetria: se  $x \equiv y$ , então  $y \equiv x$  e
- Transitividade: se  $x \equiv y$  e  $y \equiv z$ , então  $x \equiv z$ .

**Exemplo 1.1.4** Seja  $\mathbb{N} = \{0, 1, 2, \dots\}$  o conjunto dos números naturais e defina

$$x \equiv y \text{ se e somente se } x \pmod{3} = y \pmod{3}$$

Assim,  $2 \equiv 5$ ,  $6 \equiv 9$  e  $10 \equiv 16$ . Isso é claramente uma relação de equivalência.

### Fechos de Conjuntos

Seja  $A$  um conjunto,  $X \subseteq A$  e  $O$  uma operação n-ária sobre  $A$  (isto é  $O : A^n \rightarrow A$ ). Dizemos que  $X$  é fechado sobre a operação  $O$  se para cada  $x_1, \dots, x_n \in X$ ,  $O(x_1, \dots, x_n) \in X$ . O **fecho de um conjunto**  $X$  com respeito à operação  $O$  é o menor subconjunto de  $A$  contendo  $X$  que é fechado sobre  $O$ .

**Exemplo 1.1.5** O fecho do conjunto  $\{2\} \subseteq \mathbb{N}$  sobre a soma, é o conjunto dos números pares diferentes de 0. Já sobre a multiplicação é o conjunto  $\{2^n / n \geq 1\}$ .

Observe que se a operação for parcial nem sempre o fecho de um conjunto existirá, por exemplo não existe nenhum conjunto  $X \subseteq \mathbb{N}$  que seja fechado sobre a subtração, pois nesse conjunto teria que estar  $2 - 2 = 0$  e  $0 - 2$  que não está em  $\mathbb{N}$ . Mas se a operação for total, então sempre existirá o fecho de qualquer subconjunto.

### 1.1.3 Grafos e Árvores

Um **grafo** é um par  $\langle V, A \rangle$  onde  $V$  e  $A$  são conjuntos finitos e  $A$  é um subconjunto de pares não ordenados de vértices. Os elementos de  $V$  são chamados de **vértices** enquanto que os elementos de  $A$  são chamados de **arestas**. Dizer que arestas são pares não ordenados significa que as arestas  $(v_i, v_j)$  e  $(v_j, v_i)$  são as mesmas.

Quando se faz diferenças entre estas arestas, ou seja quando arestas são pares ordenados, então estamos diante de um **grafo dirigido**. Este nome se deve a que associamos uma direção (de  $v_i$  a  $v_j$ ) a cada aresta  $(v_i, v_j)$ . Neste caso, dizemos que a aresta  $(v_i, v_j)$  é uma **aresta de saída** para  $v_i$  e de **chegada** para  $v_j$ .

Grafos podem ser rotulados, sendo o rótulo um nome ou outra informação associada às componentes do grafo. Tanto os vértices quanto as arestas podem ser rotulados.

Uma maneira conveniente de se visualizar grafos é através de **diagramas**, nos quais os vértices são representados por círculos e as arestas por linhas ou setas conectando os vértices, caso o grafo seja dirigido. O grafo dirigido com vértices  $\{v_1, v_2, v_3\}$  e arestas  $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$  é desenhado na figura 1.1.

Uma seqüência de arestas  $(v_i, v_j), (v_j, v_k), (v_k, v_l), \dots, (v_m, v_n)$  diz-se um **caminho** de  $v_i$  a  $v_n$ . O **comprimento de um caminho** é o número total de arestas que ele atravessa indo do vértice inicial ao vértice final. Um caminho no qual nenhuma aresta é repetida diz-se uma **trilha**. Uma trilha é simples se nenhum vértice é repetido, isto é, se não existem duas arestas distintas,  $(v_i, v_j)$  e  $(v_k, v_l)$ , na trilha tal que  $v_i = v_k$  ou  $v_j = v_l$ . Uma trilha de  $v_i$  a ele próprio diz-se um **ciclo com base**  $v_i$ . Se nenhum vértice além de  $v_i$ , a base, é repetido o ciclo com

## 1.2. Noções de Linguagens, Gramáticas e Autômatos

---

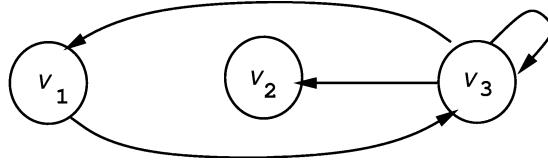


Figura 1.1: O diagrama de um grafo dirigido.

base  $v_i$  diz-se **simples**. No grafo da figura 1.1  $(v_1, v_3), (v_3, v_2)$  é uma trilha simples de  $v_1$  a  $v_2$ . A seqüência de arestas  $(v_1, v_3), (v_3, v_3), (v_3, v_1)$  é um ciclo com base  $v_1$ , mas não simples, pois repete o vértice  $v_3$ , já a seqüência  $(v_1, v_3), (v_3, v_1)$  é um ciclo simples com base  $v_1$ . Se as arestas de um grafo são rotuladas, podemos falar do **rótulo de um caminho**. Este rótulo é a seqüência de rótulos das arestas encontradas quando se percorre a trilha.

Em algumas ocasiões, nos referiremos a um algoritmo para achar todas as trilhas simples entre dois vértices (ou todos os ciclos simples baseados num vértice). Se não nos preocuparmos com eficiência podemos usar o seguinte método. Iniciar do vértice dado, digamos  $v_i$ , liste todas as arestas de saída de  $v_i$ ,  $(v_i, v_k), (v_i, v_l), \dots$ . Neste ponto temos todas as trilhas de comprimento um, começando em  $v_i$ . Para todos os vértices  $v_k, v_l, \dots$  alcançados, liste todas as arestas de saída desde que elas não levem a qualquer vértice já usado na trilha que estamos construindo. Após isto, teremos todas as trilhas simples de comprimento dois iniciando em  $v_i$  e assim por diante. Como só existe um número finito de vértices, mais cedo ou mais tarde teremos listado todas as trilhas simples, com início em  $v_i$ . Desses escolheremos aqueles terminando no vértice desejado.

As **árvores** são tipos particulares de grafos. Uma árvore é um grafo dirigido que não possui ciclos, tendo um vértice distinguido chamado **raiz**, tal que existe exatamente uma trilha da raiz a qualquer outro vértice. Esta definição implica que não existe aresta de chegada na raiz e que existem vértices sem arestas de saída, chamados **folhas** da árvore.

Se existe uma aresta de  $v_i$  para  $v_j$ ,  $v_i$  é dita um **pai** de  $v_j$ , e  $v_j$  é dito um **filho** de  $v_i$ . O **nível** associado a cada vértice é o número de arestas na trilha da raiz ao vértice. A **altura** de uma árvore é o número mais alto do nível de algum vértice.

## 1.2 Noções de Linguagens, Gramáticas e Autômatos

Na primeira parte de nosso curso abordaremos três idéias fundamentais: *linguagem*, *gramática* e *autômato*. Estaremos particularmente interessados nas suas relações.

### 1.2.1 Linguagens Formais

Em Português, existem três tipos de entidades diferentes: letras, palavras e sentenças. Existe um certo paralelismo entre elas, no sentido de que grupos de letras constituem uma palavra, e grupos de palavras uma sentença. Mas, nem toda concatenação de letras forma uma palavra, nem toda seqüência de palavras uma sentença. A analogia pode ser estendida a parágrafos, histórias, e assim por diante. A situação se dá, também, para as linguagens de programação, na

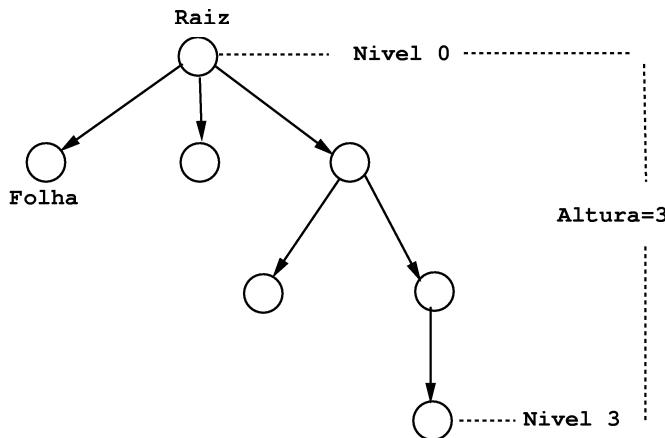


Figura 1.2: Exemplo de árvore

qual certos agrupamentos de palavras chaves são um comando e algumas seqüências de comandos são programas.

Para construir uma teoria geral que unifique todos esses casos, será necessário adotar uma definição para a estrutura de linguagens mais universal, isto é, uma estrutura na qual a decisão de quando uma cadeia de unidades constitui uma unidade maior válida, seja baseada na aplicação de regras explicitamente definidas.

Em linguagens tipo Português e Inglês, e em dialetos tipo Papiamento (dialeto da maior das ilhas na Antilha Holandesa - Curaçao), não podemos dar regras que nos permitam descrever todas as frases coerentes da linguagem (ou dialeto). Por exemplo, em Português, uma frase do tipo *< artigo >< sustantivo >< verbo >* está bem construída, no entanto ela nos permita gerar frases sem sentidos tais como “o gato late”. Além disso, é difícil estabelecer quais frases fazem sentido na língua portuguesa, isto porque depende de nossa habilidade para interpretar metáforas poéticas de sentenças aparentemente sem sentido, como por exemplo “Mãe de pedra, espuma dos condores.” do poema *Alturas do Macchu Picchu* do poeta Pablo Neruda<sup>1</sup>. Por estes motivos este tipo de linguagens são chamadas de linguagens naturais. Isto já não acontece com linguagens de programação, pois elas são “formais”, e o compilador de uma linguagem de programação é um procedimento que analisa a corretude de uma seqüência de símbolos e determina se ela constitui um programa válido ou não. Assim, o compilador não procura saber o que o programa faz, mas se ele está corretamente escrito.

Neste texto, estudaremos alguns aspectos básicos do que se conhece como “teoria de linguagens formais”. A palavra “formal” nos diz que todas as regras para a linguagem são explicitamente declaradas em termos das cadeias de símbolos que podem ocorrer nela. Nenhuma liberdade é tolerada e nenhum “entendimento profundo” é preciso. Sentenças serão consideradas como meros símbolos e não como expressões de idéias na mente humana. Neste modelo básico, as linguagens (formais) não servem para a comunicação entre intelectos, mas como um jogo de símbolos, com regras formais. O termo “formal”, aqui usado, enfatiza que é a *forma* da cadeia de símbolos que nos interessa e não seu significado.

<sup>1</sup>Poema do livro *Canto geral* escrito por Pablo Neruda em 1950.

## 1.2. Noções de Linguagens, Gramáticas e Autômatos

---

Seja  $\Sigma$  um conjunto finito não vazio de símbolos, chamado **alfabeto**. Dos símbolos individuais construiremos **cadeias** (*strings*) que são seqüências finitas de símbolos do alfabeto.

**Exemplo 1.2.1** Se o alfabeto for  $\Sigma = \{a, b\}$ , então  $abab$  e  $aaabba$  são cadeias sobre  $\Sigma$ .

Usaremos como convenção letras minúsculas iniciais do alfabeto português, como  $a$ ,  $b$ ,  $c$ ,  $d$  e  $e$  para representar elementos de um alfabeto  $\Sigma$  e letras minúsculas do final do alfabeto português, como  $u$ ,  $v$ ,  $w$ ,  $x$ ,  $y$  e  $z$ , para indicar nomes de cadeias. Por exemplo, escreveremos  $w = abaaa$  para indicar que a cadeia de nome  $w$  tem o valor específico  $abaaa$ .

A **cadeia vazia**, é aquela sem nenhum símbolo, e será denotado por  $\lambda$ .

A **concatenação** de duas cadeias  $w$  e  $v$ , denotado por  $wv$ , é a cadeia obtida juntando-se os símbolos de  $v$  à direita de  $w$ , isto é, se  $w = a_1 \dots a_n$  e  $v = b_1 \dots b_m$ , então  $wv = a_1 \dots a_n b_1 \dots b_m$ . Note que,  $\lambda w = w\lambda = w$  para qualquer cadeia  $w$ .

A **reversa** de uma cadeia é obtida escrevendo-se os símbolos em ordem reversa. Se  $w$  é a cadeia acima, então a reversa de  $w$ , denotado por  $w^R$ , é a cadeia  $a_n \dots a_1$ . Uma definição recursiva da reversa de uma cadeia é a seguinte:  $\lambda^R = \lambda$  e  $(aw)^R = w^Ra$

**Exemplo 1.2.2** Sejam  $w = bbab$  e  $v = aaab$ . Então  $wv = bbabaaab$ ,  $vw = aaabbab$ ,  $w^R = babb$  e  $v^R = baaa$ .

Se  $w = vu$ , então  $v$  é chamado de **prefixo** e  $u$  de **sufixo** de  $w$ . Formalmente  $v$  é um prefixo de  $w$  se existe uma cadeia  $u$  tal que  $vu = w$ . Analogamente,  $u$  é um sufixo de  $w$  se existe uma cadeia  $v$  tal que  $vu = w$ .

**Exemplo 1.2.3** Seja  $w = abbab$ . O conjunto de todos os prefixos de  $w$  é  $\{\lambda, a, ab, abb, abba, abbab\}$  e de todos os sufixos é  $\{\lambda, b, ab, bab, bbab, abbab\}$ .

O **comprimento** de uma cadeia  $w$ , denotado por  $|w|$ , é o número de símbolos que figuram na cadeia. Por exemplo,  $|abba| = 4$ . Sejam  $u$  e  $v$  cadeias quaisquer sobre um alfabeto  $\Sigma$ . Então

$$|uv| = |u| + |v|. \quad (1.2)$$

De fato, para mostrar esta igualdade precisamos de uma definição mais precisa de comprimento de cadeia. Daremos a seguinte definição recursiva.

- $|\lambda| = 0$  e
- $|wa| = |w| + 1$ ,

para todo  $a \in \Sigma$  e qualquer cadeia  $w$  de  $\Sigma$ . Usaremos a indução no comprimento da cadeia para mostrar a equação 1.2.

Se  $|v| = 0$ , então  $v = \lambda$ . Logo, por definição,

$$\begin{aligned}|uv| &= |u| \\&= |u| + 0 \\&= |u| + |v|.\end{aligned}$$

Suponha que  $|uv| = |u| + |v|$  e calculemos  $|uva|$ :

$$\begin{aligned}|uva| &= |uv| + 1 \\&= |u| + |v| + 1 \\&= |u| + |va|.\end{aligned}$$

Logo, por indução, a igualdade é válida para todo  $v$ .

Se  $w$  é uma cadeia, então  $w^n$  é a cadeia obtida concatenando  $w$  com ela própria  $n$  vezes. Portanto,  $w^{n+1} = ww^n$ . No caso especial de  $n = 0$ ,  $w^0 = \lambda$ .

Seja  $\Sigma$  um alfabeto. O **fecho estrela** de  $\Sigma$ , denotado por  $\Sigma^*$ , é o conjunto de todas as cadeias (finitas) obtidas concatenando zero ou mais símbolos de  $\Sigma$ . Por exemplo, se  $\Sigma = \{a\}$ , então

$$\Sigma^* = \{\lambda, a, aa, aaa, aaaa, \dots\}.$$

Formalmente, podemos definir o fecho estrela de um alfabeto  $\Sigma$ , recursivamente, por

1.  $\lambda \in \Sigma^*$
2. Se  $w \in \Sigma^*$  e  $a \in \Sigma$ , então  $wa \in \Sigma^*$
3. Os únicos elementos de  $\Sigma^*$  são aqueles que podem ser obtidos aplicando uma quantidade finita de vezes as regras 1) e 2), acima.

O conjunto  $\Sigma^*$  sempre contém  $\lambda$ . O fecho estrela de  $\Sigma$  sem a cadeia vazia é denominado **fecho positivo** do alfabeto  $\Sigma$ , e é denotado por  $\Sigma^+$ . Isto é,  $\Sigma^+ = \Sigma^* - \{\lambda\}$ .

Enquanto um alfabeto  $\Sigma$  é um conjunto finito,  $\Sigma^*$  e  $\Sigma^+$  são sempre infinitos, pois  $\Sigma$  é não vazio e não existe limite no comprimento das cadeias nesses dois conjuntos.

Um subconjunto qualquer de  $\Sigma^*$  pode ser visto como uma **linguagem** sobre o alfabeto  $\Sigma$ . Uma cadeia  $w$  numa linguagem  $L$ , isto é  $w \in L$ , será chamada **palavra** ou **sentença** de  $L$ . Na teoria das linguagens formais não há distinção entre palavras e sentenças. Para fornecer mais estrutura a essa definição, bastante ampla, de linguagem, estudaremos métodos pelos quais poderemos definir linguagens particulares.

**Exemplo 1.2.4** Seja  $\Sigma = \{a, b\}$ . Então  $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ . O conjunto  $\{a, aa, aab\}$  é uma linguagem sobre  $\Sigma$ .

## 1.2. Noções de Linguagens, Gramáticas e Autômatos

---

A linguagem do exemplo anterior tem uma quantidade finita de palavras, por isso ela é chamada linguagem finita. O conjunto  $L = \{a^n b^n / n \geq 0\}$  é também uma linguagem sobre  $\Sigma$ . As cadeias  $aabb$  e  $aaaabbbb$  são palavras de  $L$ , mas a cadeia  $abb$  não está em  $L$ . Esta linguagem é infinita.

Como as linguagens são conjuntos, a união, a intersecção e a diferença de duas linguagens são imediatamente definidas. O **complemento** de uma linguagem  $L$ , sobre o alfabeto  $\Sigma$ , é definida com respeito a  $\Sigma^*$ , isto é, o complemento de  $L$  é a linguagem, sobre  $\Sigma$ , definida por

$$\overline{L} = \Sigma^* - L.$$

**Exemplo 1.2.5** Seja  $\mathcal{L} = \{w \in \Sigma^* / |w| \text{ é par}\}$ . Então  $\overline{\mathcal{L}} = \{w \in \Sigma^* / |w| \text{ é ímpar}\}$ .

Mas como linguagens têm características particulares (são conjuntos, mas de cadeias), podemos definir também operações próprias para linguagens, tais como concatenação, fecho estrela, fecho positivo, reverso, prefixos, sufixos, etc.

A **concatenação** de duas linguagens  $L_1$  e  $L_2$ , sobre os alfabetos  $\Sigma_1$  e  $\Sigma_2$ , respectivamente, é a linguagem, sobre o alfabeto  $\Sigma_1 \cup \Sigma_2$ , denotada por  $L_1 L_2$ , consistindo do conjunto de todas as cadeias obtidas concatenando-se qualquer elemento de  $L_1$  com qualquer elemento de  $L_2$ , isto é

$$L_1 L_2 = \{xy / x \in L_1 \text{ e } y \in L_2\}.$$

**Exemplo 1.2.6** Sejam as linguagens  $L_1 = \{aa, ab\}$  e  $L_2 = \{bb, ba, aba\}$ . Então a concatenação de  $L_1$  com  $L_2$  é a linguagem

$$L_1 L_2 = \{aabb, aaba, aaaba, abbb, abba, ababa\}.$$

Definimos  $L^n$  como  $L$  concatenado  $n$  vezes com ele próprio. Nos casos especiais de  $n = 0$  e  $n = 1$  temos que,

$$L^0 = \{\lambda\} \quad \text{e} \quad L^1 = L.$$

**Exemplo 1.2.7** Se  $L = \{a^n b^n / n \geq 0\}$ , então  $L^2 = \{a^m b^m a^n b^n / m \geq 0 \text{ e } n \geq 0\}$ . Observe que  $m$  e  $n$  não estão relacionados. A cadeia  $aabbaaabbb$  está em  $L^2$ .

O **fecho estrela** de uma linguagem  $L$ , sobre um alfabeto  $\Sigma$ , denotado por  $L^*$ , é o fecho de  $L \cup \{\lambda\} \subseteq \Sigma^*$  com respeito à operação (binária) de concatenação de cadeias. Por definição de fecho de conjuntos  $L^* \subseteq \Sigma^*$ , logo  $L^*$  é uma linguagem sobre o próprio  $\Sigma$ .  $L^*$  também pode ser obtido da seguinte maneira:

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$$

O **fecho positivo** da linguagem  $L$ , denotado por  $L^+$ , o fecho de  $L \subseteq \Sigma^*$  com respeito à operação (binária) de concatenação de cadeias. Também pode ser definido por

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cup \dots$$

Observe que não necessariamente é verdade que  $L^+ = L^* - L^0 = L^* - \{\lambda\}$ , pois se  $\lambda \in L$  então,  $L^+ = L^*$ .

**Exemplo 1.2.8** Seja  $L$  a linguagem  $L = \{aa, bb, aba\}$ . Então os fechos estrela e positivo de  $L$  são definidos por:

$$L^* = \{\lambda, aa, bb, aba, aaaa, aabb, aaaba, bbaa, bbbb, bbaba, abaaa, ababb, abaaba, aaaaaa, aaaabb, \dots\} \quad \text{e}$$

$$L^+ = \{aa, bb, aba, aaaa, aabb, aaaba, bbaa, bbbb, bbaba, abaaa, ababb, abaaba, aaaaaa, aaaabb, \dots\}.$$

A **linguagem reversa** de uma linguagem  $L$ , sobre um alfabeto  $\Sigma$ , é a linguagem  $L^R$ , sobre o próprio alfabeto  $\Sigma$ , consistindo de todas as cadeias em  $L$  revertidas, isto é

$$w \in L \text{ se e somente se } w^R \in L^R.$$

Como  $(w^R)^R = w$  para qualquer cadeia  $w$ , temos que  $L^R$  é definida por

$$L^R = \{w \in \Sigma^* / w^R \in L\}.$$

**Exemplo 1.2.9** Seja a linguagem  $L = \{aa, baa, aba, abbb\}$ . A linguagem reversa de  $L$  é:

$$L^R = \{aa, aab, aba, bbba\}.$$

A **linguagem de prefixos** de uma linguagem  $L$ , sobre um alfabeto  $\Sigma$ , é a linguagem  $L^P$ , sobre o próprio  $\Sigma$ , definida por

$$L^P = \{v \in \Sigma^* / vw \in L \text{ para algum } w \in \Sigma^*\}.$$

Isto é,  $L^P$  é a linguagem consistindo de todas as cadeias que são prefixos de alguma cadeia em  $L$ .

Analogamente, A **linguagem de sufixos** de uma linguagem  $L$ , sobre um alfabeto  $\Sigma$ , é a linguagem  $L^S$ , sobre o próprio  $\Sigma$ , definida por

## 1.2. Noções de Linguagens, Gramáticas e Autômatos

---

$$L^S = \{w \in \Sigma^* / vw \in L \text{ para algum } v \in \Sigma^*\}.$$

Isto é,  $L^S$  é a linguagem consistindo de todas as cadeias que são sufixos de alguma cadeia em  $L$ .

Observe que  $L \subseteq L^R$  e  $L \subseteq L^S$ .

**Exemplo 1.2.10** Seja a linguagem  $L = \{aa, ba, baa\}$ . Então

$$L^P = \{\lambda, a, b, aa, ba, baa\} \quad \text{e} \quad L^S = \{\lambda, a, ba, aa, baa\}.$$

### 1.2.2 Gramáticas

Para estudar linguagens, matematicamente, precisamos de um mecanismo para descrevê-las. A linguagem do dia a dia é imprecisa e ambígua, portanto, a descrição em português, em geral, é inadequada. A notação de conjuntos é mais adequada, porém limitada. Aqui, introduziremos a noção de *gramática*, uma ferramenta comum e poderosa para definir linguagens.

Uma gramática para a língua portuguesa nos diz se uma sentença, em particular, é bem formada ou não. Uma regra típica da gramática portuguesa é “uma sentença pode consistir de um substantivo, seguido por um predicado”. Mais precisamente, podemos escrever isso como

$$\langle \text{sentença} \rangle \longrightarrow \langle \text{frase} - \text{substantivo} \rangle \langle \text{predicado} \rangle$$

com a interpretação óbvia. Isto é bastante para se lidar com sentenças. Devemos também fornecer definições dos conceitos introduzidos na definição de sentença  $\langle \text{frase} - \text{substantivo} \rangle$  e  $\langle \text{predicado} \rangle$ . Portanto,

$$\langle \text{frase} - \text{substantivo} \rangle \longrightarrow \langle \text{artigo} \rangle \langle \text{substantivo} \rangle$$

$$\langle \text{predicado} \rangle \longrightarrow \langle \text{verbo} \rangle$$

Se associarmos a cada sentença “um” ou “uma” como  $\langle \text{artigo} \rangle$ , “cão” e “menina” como  $\langle \text{substantivo} \rangle$  e “corre” e “caminha” como  $\langle \text{verbo} \rangle$ , então a gramática nos diz que as sentenças “um cão corre” e “uma menina caminha” são sentenças bem-formadas.

**Definição 1.2.11** Uma **gramática**,  $G$ , é definida como uma quádrupla  $G = \langle V, T, S, P \rangle$ , onde

- $V$  é um conjunto finito de objetos, chamados **variáveis**,
- $T$  é um conjunto finito de objetos, disjunto de  $V$ , chamados **símbolos terminais**,
- $S \in V$  é um símbolo especial, chamado **variável de início**, e
- $P$  é um conjunto finito de **produções**.

O principal na gramática são as regras de produção, pois são elas que especificam como a gramática transforma uma cadeia em outra, o qual permite definir uma linguagem associada à gramática. Assumiremos que todas as produções são da forma

$$x \longrightarrow y,$$

onde  $x$  é um elemento de  $(V \cup T)^+$  e  $y$  está em  $(V \cup T)^*$ . As produções são aplicadas como segue: dado uma cadeia da forma

$$w = uxv,$$

dizemos que a produção é aplicável a esta cadeia, e podemos usá-la para trocar uma ocorrência de  $x$  por  $y$ , obtendo, assim, a nova cadeia

$$z = u y v.$$

Neste caso, dizemos que  $w$  **deriva**  $z$  ou  $z$  é derivada de  $w$ , denotado por  $w \xrightarrow{*} z$ . Cadeias sucessivas são derivadas aplicando regras de produção da gramática, numa ordem arbitrária. Uma produção pode ser usada se ela é aplicável e pode ser aplicada quantas vezes se quiser. Se  $w_1 \xrightarrow{*} w_2 \xrightarrow{*} \dots \xrightarrow{*} w_n$ , dizemos que  $w_1$  deriva  $w_n$  e escrevemos

$$w_1 \xrightarrow{*} w_n.$$

A estrela  $*$  indica que foi considerado um número não especificado de etapas (incluindo zero) para derivar  $w_n$  de  $w_1$ . Portanto,  $w \xrightarrow{*} w$  sempre se dá. Para indicar que, pelo menos uma produção foi aplicada escreveremos

$$w \xrightarrow{+} v.$$

Aplicando as regras de produção em ordens diferentes, uma gramática pode gerar muitas cadeias. O conjunto de todas cadeias é a linguagem definida ou gerada pela gramática.

**Definição 1.2.12** *Seja  $G = \langle V, T, S, P \rangle$  uma gramática. O conjunto*

$$L(G) = \{w \in T^* / S \xrightarrow{*} w\}$$

*é denominado linguagem gerada pela gramática  $G$ . Os elementos de  $L(G)$  são chamados sentenças ou palavras.*

Se  $w \in L(G)$ , então a seqüência

$$S \xrightarrow{*} w_1 \xrightarrow{*} w_2 \xrightarrow{*} \dots \xrightarrow{*} w_n \xrightarrow{*} w$$

é uma **derivação** da sentença  $w$ . As cadeias  $S, w_1, w_2, \dots, w_n$  que contém variáveis e objetos terminais, são chamados **formas sentenciais** da derivação.

## 1.2. Noções de Linguagens, Gramáticas e Autômatos

---

**Exemplo 1.2.13** Considere a gramática  $G = \langle \{S\}, \{a, b\}, S, P \rangle$ , onde  $P$  é dado por

$$S \longrightarrow aSb$$

$$S \longrightarrow \lambda.$$

Então,  $S \implies aSb \implies aaSbb \implies aabb$ . Logo, podemos escrever  $S \xrightarrow{*} aabb$ .

A cadeia  $aabb$  é uma sentença na linguagem gerada por  $G$ , enquanto  $aaSbb$  é uma forma sentencial.

Uma gramática  $G$  define completamente a linguagem  $L(G)$ , porém pode não ser fácil obter uma descrição explícita da linguagem a partir da gramática. Nesse exemplo, no entanto, não é difícil conjecturar que

$$L(G) = \{a^n b^n / n \geq 0\}$$

e é fácil prová-la.

Se observarmos que a regra  $S \longrightarrow aSb$  é recursiva, teremos uma prova por indução.

Mostremos que todas as formas sentenciais tem a forma  $w_i = a^i S b^i$ .

1. Se  $i = 0$ ,  $w_0 = S$
2. Suponha que  $w_i = a^i S b^i$ . Mostremos que  $w_{i+1} = a^{i+1} S b^{i+1}$ . Mas, de  $a^i S b^i$ , aplicando a regra  $S \longrightarrow aSb$ , obtemos  $a^{i+1} S b^{i+1}$ . Observe que se aplicarmos a produção  $S \longrightarrow \lambda$ , obteremos uma sentença e não uma forma sentencial. Logo, toda forma sentencial tem aquela forma.

Agora, aplicando a regra  $S \longrightarrow \lambda$ , temos que todas as sentenças tem a forma  $a^n b^n$ , para  $n \geq 0$ .

**Exemplo 1.2.14** Achar a gramática que gera  $L = \{a^n b^{n+1} / n \leq 0\}$ .

A idéia por trás do exemplo anterior pode ser estendida a este caso. Tudo que precisamos fazer é gerar um  $b$  extra. Isto pode ser feito com a produção  $S \longrightarrow Ab$ , com outras produções escolhidas tais que  $A$  possa derivar a linguagem do exemplo anterior. Com esse raciocínio, obtemos a gramática  $G = \langle \{S, A\}, \{a, b\}, S, P \rangle$ , com produções  $P$ :

$$S \longrightarrow Ab$$

$$A \longrightarrow aAb$$

$$A \longrightarrow \lambda$$

Para se convencer que essa é a gramática pedida derive algumas sentenças.

Os exemplos são relativamente fáceis, de modo que os argumentos rigorosos podem parecer supérfluos. Mas, em geral, não é fácil achar a gramática para uma linguagem descrita informalmente ou dar uma caracterização intuitiva da linguagem definida pela gramática. Para mostrar que uma dada linguagem é, de fato, gerada por uma gramática,  $G$ , devemos mostrar:

- Que todo  $w \in L$  pode ser derivado de  $S$ , usando  $G$ .
- Toda cadeia derivada, usando  $G$ , está em  $L$ .

**Exemplo 1.2.15** Considere a gramática  $G_1 = \langle \{A, S\}, \{a, b\}, S, P_1 \rangle$ , com  $P_1$  consistindo das produções:

$$\begin{aligned} S &\longrightarrow aAb \mid \lambda \\ A &\longrightarrow aAb \mid \lambda \end{aligned}$$

Introduzimos, aqui, uma abreviação no qual várias produções com o mesmo lado esquerdo são escritos numa única linha, com lado direito alternativo separado por “|”. Nessa notação,  $S \longrightarrow aAb \mid \lambda$  substitui as duas produções,  $S \longrightarrow aAb$  e  $S \longrightarrow \lambda$ .

Essa gramática é equivalente à gramática  $G$  do exemplo 1.2.13, no sentido que gera a mesma linguagem. A equivalência é fácil de mostrar provando que  $L(G_1) = \{a^n b^n / n \geq 0\}$ .

### 1.2.3 Autômatos

Um autômato é um modelo abstrato de um computador digital. Como tal, todo autômato inclui algumas características essenciais. Ele possui um mecanismo para ler entradas. Assume-se que as entradas são cadeias sobre um dado alfabeto, escritos numa *fita de entrada*, a qual o autômato pode ler, mas não alterar. A fita de entrada é dividida em quadrados ou células, cada um dos quais pode conter um símbolo. O mecanismo de entrada pode ler a fita de entrada da esquerda para a direita, um símbolo de cada vez. O mecanismo de entrada pode detectar o fim da cadeia de entrada (percebendo uma condição de término da cadeia). O autômato pode produzir saídas, de alguma forma. Ele pode ter um dispositivo temporário de *armazenamento*, consistindo de um número ilimitado de células, cada uma capaz de manter um único símbolo do alfabeto (não necessariamente o mesmo do alfabeto de entrada). O autômato pode ler e alterar o conteúdo das células de armazenamento. Finalmente, o autômato possui uma *unidade de controle*, que pode estar em qualquer um de um número finito de *estados internos*, podendo trocar os estados de um modo bem definido. A figura 1.3 mostra a representação esquemática de um autômato.

Assumimos que um autômato opera num tempo discreto. Em qualquer tempo dado, a unidade de controle está em algum estado interno e o mecanismo de entrada está apontando para um símbolo particular na fita de entrada. O estado interno da unidade de controle na próxima etapa de tempo é determinado pelo *próximo-estado* ou *função de transição*. Esta função de transição fornece o próximo estado, em função do estado corrente, o símbolo de entrada corrente e a informação corrente na área de armazenamento temporária. Durante a transição de um intervalo de tempo ao próximo, a saída pode ser produzida, ou trocada informação na área de armazenamento temporário. O termo *configuração* será usado para referenciar um estado particular da unidade de controle, a fita de entrada e o armazenamento temporário. A transição do autômato de uma configuração para a próxima será um *movimento*.

Esse modelo geral cobre todos os autômatos que serão discutidos neste texto.

### 1.3. Exercícios

---

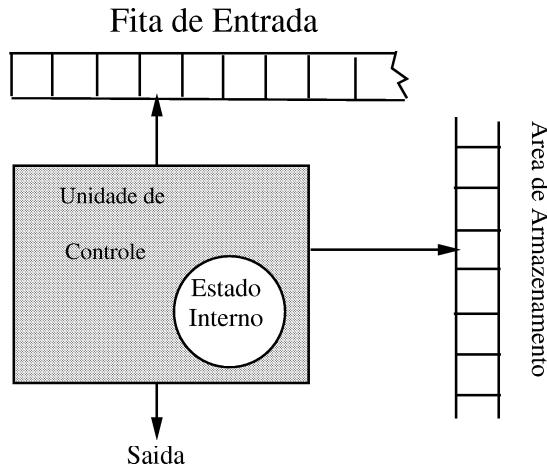


Figura 1.3: Representação esquemática de um autômato.

### 1.3 Exercícios

1. Seja a cadeia  $w = 010010001$  no alfabeto  $\Sigma = \{0, 1\}$ . Determine  $w^R$ ,  $|w|$ ,  $w^0$ ,  $w^3$ , o conjunto de todos os prefixos e de todos os sufixos de  $w$ .
2. Demonstre que para quaisquer cadeias  $w$  e  $u$  e para todo  $n \geq 0$  temos que
  - (a) se  $u$  é um prefixo de  $w$  então  $|u| \leq |w|$ ,
  - (b)  $|w^n| = n|w|$ ,
  - (c)  $(w^R)^R = w$ , e
  - (d)  $(wu)^R = u^R w^R$ .
3. Seja  $\mathcal{L} = \{abba, abb, a, ba\}$  no alfabeto  $\Sigma = \{a, b\}$ . Determine  $\mathcal{L}^R$ ,  $\mathcal{L}^P$ ,  $\mathcal{L}^S$ ,  $\mathcal{L}^2$  e oito cadeias de  $\overline{\mathcal{L}}$  e de  $\mathcal{L}^*$ .
4. Sejam  $\mathcal{L}_1 = \{abb, a, ba\}$  e  $\mathcal{L}_2 = \{\lambda, bb, a\}$ . Determine  $\mathcal{L}_1 \circ \mathcal{L}_2$  e  $\mathcal{L}_2 \circ \mathcal{L}_1$ .
5. Mostre que para qualquer  $n, m \geq 0$  e linguagem  $\mathcal{L}$  temos que
  - (a)  $(\mathcal{L}^n)^m = \mathcal{L}^{nm}$ ,
  - (b)  $\mathcal{L}^n \circ \mathcal{L}^m = \mathcal{L}^{n+m}$ ,
  - (c)  $(\mathcal{L}^R)^n = (\mathcal{L}^n)^R$ ,
  - (d)  $\overline{\mathcal{L}^R} = \overline{\mathcal{L}}^R$ , e
  - (e)  $\mathcal{L}^P = ((\mathcal{L})^R)^S$ .
6. Seja  $\mathcal{L}$  uma linguagem qualquer. Então podemos afirmar que  $\overline{\mathcal{L}^n} = \overline{\mathcal{L}}^n$ , para todo  $n \geq 0$ ?
7. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  duas linguagens tais que  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ . Demonstre que

- (a)  $\mathcal{L}_1^P \subseteq \mathcal{L}_2^P$ ,
  - (b)  $\mathcal{L}_1^R \subseteq \mathcal{L}_2^R$ ,
  - (c)  $\overline{\mathcal{L}_2} \subseteq \overline{\mathcal{L}_1}$ ,
  - (d) Para toda linguagem  $\mathcal{L}$ ,  $\mathcal{L}_1 \circ \mathcal{L} \subseteq \mathcal{L}_2 \circ \mathcal{L}$ .
8. Seja  $\mathcal{L}$  uma linguagem. Demonstre que  $\overline{\mathcal{L}}^R = \overline{\mathcal{L}^R}$ .
9. Descreva de maneira genérica em que situações  $\mathcal{L}^P = (\mathcal{L}^S)^R$ .

### **1.3. Exercícios**

---

## Capítulo 2

# Autômatos Finitos

Neste capítulo, veremos a classe mais simples de autômatos, denominados **autômatos finitos**, os quais podem ser classificados em determinísticos e não-determinísticos.

### 2.1 Autômatos Finitos Determinísticos

Um autômato reconhecedor é aquele que para cada cadeia na fita de entrada diz se ela é aceita ou não. Ou seja, diz se uma cadeia de símbolos faz parte ou não de uma linguagem. Um autômato finito é um autômato reconhecedor que não usa área de armazenamento e que sempre pára (para isto, lê uma célula uma única vez).

**Definição 2.1.1** Um Autômato finito determinístico (AFD) é uma quíntupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , onde

- $Q$  é um conjunto finito de **estados internos**,
- $\Sigma$  é um conjunto finito de símbolos, chamado **alfabeto de entrada**,
- $\delta : Q \times \Sigma \rightarrow Q$  é uma função total, chamada **função de transição**,
- $q_0 \in Q$  é o **estado inicial**,
- $F \subseteq Q$  é o conjunto de **estados finais** ou **estados de aceitação**.

Um AFD opera como segue: no início assumimos que ele está no estado inicial  $q_0$ , com seu mecanismo de entrada apontando para o símbolo mais à esquerda da cadeia de entrada. Durante cada movimento do autômato, o mecanismo de entrada avança uma posição para a direita, consumindo um símbolo de entrada. Quando for atingido o fim da cadeia, esta será **aceita** ou **reconhecida** se o autômato estiver num de seus estados finais. Caso contrário, a **cadeia é rejeitada**. O mecanismo de entrada pode se mover, somente, da esquerda para a direita, lendo exatamente, um símbolo em cada etapa. Observe que quando a execução de um autômato atinge um estado final, isso não significa que a execução finalizou, pois ela só termina quando o autômato lê toda a entrada na fita.

## 2.1. Autômatos Finitos Determinísticos

---

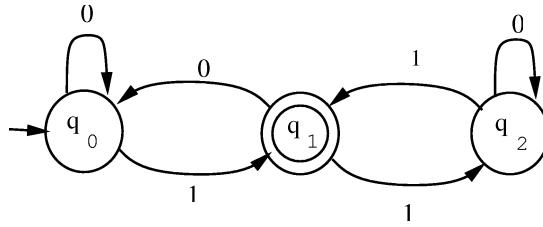


Figura 2.1: Exemplo de AFD.

Para visualizar e representar um autômato finito usamos grafos dirigidos rotulados também chamados de *grafos de transição*, nos quais os vértices representam os estados e as arestas representam as transições. Os rótulos dos vértices são os nomes dos estados, enquanto os rótulos das arestas são os valores correntes dos símbolos de entrada. Por exemplo, se  $q_0$  e  $q_1$  são estados de algum AFD  $M$ , o grafo associado a  $M$  terá um vértice rotulado com  $q_0$  e outro rotulado com  $q_1$ . Uma aresta  $(q_0, q_1)$ , rotulada por  $a$ , representa a transição  $\delta(q_0, a) = q_1$ . O estado inicial será identificado por uma seta não-rotulada chegando ao vértice. Os estados finais serão identificados por um círculo duplo.

Mais formalmente, se  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  é um AFD, então a ele está associado um grafo de transição,  $G_M$ , tendo exatamente  $|Q|$  vértices, cada um rotulado com um  $q_i \in Q$  diferente. Para cada regra de transição  $\delta(q_i, a) = q_j$ , o grafo tem uma aresta  $(q_i, q_j)$ , rotulada  $a$ . O vértice associado a  $q_0$  é chamado **vértice inicial**, enquanto que aqueles rotulados por algum  $q_f \in F$  são os **vértices finais**. É muito simples converter a definição de um AFD,  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , no seu grafo de transição e vice-versa.

**Exemplo 2.1.2** O grafo da figura 2.1 representa o AFD  $M = \langle \{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\} \rangle$ , onde  $\delta$  é dado por

$$\begin{aligned}\delta(q_0, 0) &= q_0, & \delta(q_0, 1) &= q_1 \\ \delta(q_1, 0) &= q_0, & \delta(q_1, 1) &= q_2 \\ \delta(q_2, 0) &= q_2, & \delta(q_2, 1) &= q_1.\end{aligned}$$

Este AFD aceita a cadeia 01. Inicia no estado  $q_0$  e aponta para o símbolo 0. Este último fato é representado pela aresta do grafo rotulada com 0. Observamos que o autômato permanece no estado  $q_0$ . Após o símbolo 1 ter sido lido o autômato entra no estado  $q_1$ . Estamos agora no fim da cadeia e, ao mesmo tempo, no estado final  $q_1$ . Portanto, a cadeia 01 é aceita. O AFD não aceita a cadeia 00, porque após ler dois 0's consecutivos ele não estará no estado  $q_1$ , mas em  $q_0$ . Raciocinando de modo análogo, vemos que o autômato aceitará as cadeias 101, 00111 e 11001, mas não aceitará 100 nem 1100.

É conveniente introduzir a **função de transição estendida**  $\delta^* : Q \times \Sigma^* \rightarrow Q$ . O segundo argumento de  $\delta^*$  é uma cadeia de  $\Sigma^*$ , em vez de um único símbolo, e seu valor é o estado do autômato após ler aquela cadeia. Por exemplo, no caso do AFD do exemplo 2.1.2,  $\delta^*(q_0, 001) = q_1$  e  $\delta^*(q_1, 1110) = q_2$ . Formalmente, podemos definir  $\delta^*$ , recursivamente, por

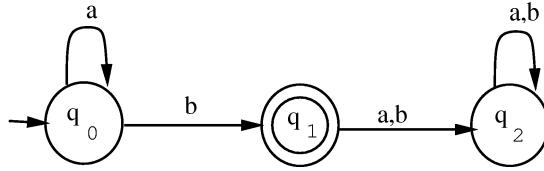


Figura 2.2: Exemplo de AFD.

1.  $\delta^*(q, \lambda) = q$
2.  $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

Para todo  $q \in Q$ ,  $w \in \Sigma^*$  e  $a \in \Sigma$ . Para ver como essa notação é adequada, vamos aplicar essas definições ao exemplo anterior.

$$\begin{aligned}
 \delta^*(q_0, 01) &= \delta(\delta^*(q_0, 0), 1) && \text{aplicando 2)} \\
 &= \delta(\delta(\delta^*(q_0, \lambda), 0), 1) && \text{aplicando 2) e } 0 = \lambda 0 \\
 &= \delta(\delta(q_0, 0), 1) && \text{aplicando 1)} \\
 &= \delta(q_0, 1) && \text{aplicando } \delta \\
 &= q_1 && \text{aplicando } \delta
 \end{aligned}$$

## 2.2 Linguagens e AFD's

Tendo definido precisamente um AFD podemos, agora, definir formalmente o que entendemos por uma linguagem reconhecida ou aceita por um AFD. A associação é óbvia: a linguagem é o conjunto de todas as cadeias reconhecidas pelo autômato.

**Definição 2.2.1** A linguagem aceita ou reconhecida por um AFD,  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , denotado por  $L(M)$ , é o conjunto de todas as cadeias, sobre  $\Sigma$ , aceitas por  $M$ . Formalmente,

$$L(M) = \{w \in \Sigma^* / \delta^*(q_0, w) \in F\}$$

Observe a exigência de que  $\delta$ , e consequentemente  $\delta^*$ , seja uma função total. Em cada etapa, um movimento único é definido, isto justifica o nome **determinístico**, para esse autômato.

Um AFD processará toda cadeia em  $\Sigma^*$ , aceitando-a ou não. Não aceitação significa que o AFD pára, num estado não final. Portanto, o complemento da linguagem aceita por um autômato  $M$  é a linguagem

$$\overline{L(M)} = \Sigma^* - L(M) = \{w \in \Sigma^* / \delta^*(q_0, w) \notin F\}.$$

## 2.2. Linguagens e AFD's

---

	$a$	$b$
$q_0$	$q_0$	$q_1$
$q_1$	$q_2$	$q_2$
$q_2$	$q_2$	$q_2$

Tabela 2.1: Representação por tabela do AFD do exemplo.

**Exemplo 2.2.2** Considere o AFD, na figura 2.2.

Ao desenhar a figura 2.2, permitimos o uso de dois rótulos numa mesma aresta. Tal multiplicidade de arestas é uma abreviação para duas ou mais transições distintas: a transição é tomada se o símbolo de entrada combina com qualquer dos rótulos da aresta.

O autômato da figura 2.2 permanece no seu estado inicial,  $q_0$ , até o primeiro  $b$  ser lido. Se este é também o último símbolo da entrada, então a cadeia é aceita. Se não, o AFD vai para o estado  $q_2$ , do qual nunca escapa. Tal estado é chamado um **estado de morte**. Como podemos ver no grafo, o autômato aceita todas as cadeias que têm exatamente um  $b$ , localizado na posição mais à direita da cadeia. Todas as outras cadeias são rejeitadas. Em notação de conjunto, a linguagem aceita por este autômato é

$$\mathcal{L} = \{a^n b / n \geq 0\}$$

**Teorema 2.2.3** Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD, e seja  $G_M$  seu grafo associado. Então para todo  $q_i, q_j \in Q$  e  $w \in \Sigma^+$ ,  $\delta^*(q_i, w) = q_j$  se e somente se existe em  $G_M$  um caminho com rótulo  $w$  de  $q_i$  a  $q_j$ .

**DEMONSTRAÇÃO:** Este fato é óbvio, examinando os exemplos 2.1.2 e 2.2.2. Podemos provar isso, rigorosamente, usando a indução no comprimento de  $w$ . Suponha que o teorema é verdadeiro para todas as cadeias  $v$ , com comprimento  $|v| \leq n$ . Considere uma cadeia  $w$  de comprimento  $|w| = n + 1$  e escreva como  $w = va$ , para algum  $a \in \Sigma$ . Suponha, agora, que  $\delta^*(q_i, v) = q_k$ . Como  $|v| = n$ , deve existir um caminho em  $G_M$ , rotulado por  $v$ , de  $q_i$  a  $q_k$ . Mas se  $\delta^*(q_i, w) = q_j$ , então  $M$  deve ter uma transição  $\delta(q_k, a) = q_j$ , tal que por construção  $G_M$  tem uma aresta  $(q_k, q_j)$ , com rótulo  $a$ . Portanto, existe um caminho em  $G_M$ , rotulado  $va = w$ , de  $q_i$  a  $q_j$ . Como o resultado é obviamente verdadeiro para  $n = 1$ , podemos argumentar, por indução, que para todo  $w \in \Sigma^+$ ,

$$\delta^*(q_i, w) = q_j \tag{2.1}$$

o que implica que existe um caminho em  $G_M$  de  $q_i$  a  $q_j$ , rotulado  $w$ . Um argumento simples mostra que a existência de tal caminho implica na igualdade (2.1), completando a prova. ■

Uma outra maneira de representar um autômato é através de uma matriz (tabela). A tabela 2.1 e o grafo da figura 2.2 representam o mesmo AFD. Aqui, os rótulos da linha representam os estados correntes e os das colunas representam os símbolos de entrada corrente.

**Exemplo 2.2.4** Achar um AFD que reconheça todas as cadeias, sobre o alfabeto  $\Sigma = \{a, b\}$ , com prefixo ab.

A única exigência aqui, é que os dois primeiros símbolos na cadeia sejam a e b, respectivamente. Após eles terem sido lidos não resta nenhuma decisão mais a ser tomada. Podemos, portanto, resolver o problema com um autômato que tem quatro estados: um estado inicial, dois estados para reconhecer ab, um estado de morte final e outro não final. Se o primeiro símbolo é a e o segundo b, o autômato vai para o estado final de morte, onde ele ficará, pois as entradas restantes não trazem problemas. Por outro lado, se o primeiro símbolo não é a ou o segundo não é b, o autômato entra num estado não final de morte. A solução é mostrada na figura 2.3.

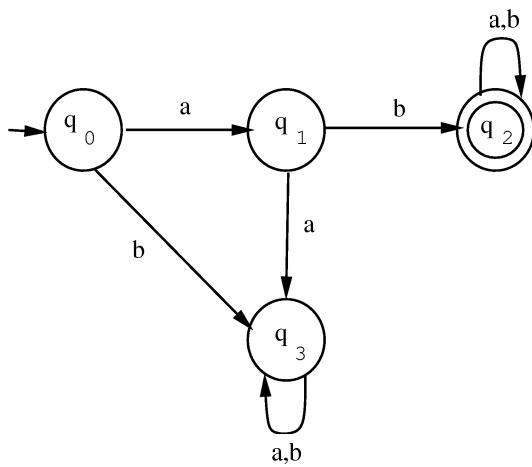


Figura 2.3: Exemplo de AFD.

**Exemplo 2.2.5** Achar um AFD que aceite todas as cadeias, sobre  $\{0, 1\}$ , exceto aquelas contendo a subcadeia 001.

Para decidir se a subcadeia 001 ocorreu, precisamos conhecer não somente o símbolo de entrada corrente, mas também relembrar se ele foi precedido por dois 0's ou não. Podemos controlar isso colocando o autômato num estado específico e rotulando-o adequadamente. Assim, como os nomes de variáveis em linguagens de programação são mnemônicos, isto é, nomes que denotam o papel da variável no programa, os nomes de estados são arbitrários e, portanto, também, podem ser escolhidos por razões mnemônicas. Por exemplo, o estado no qual dois 0's imediatamente precedem um símbolo pode ser rotulado, simplesmente, por 00.

Se a cadeia começa com 001, então ela é rejeitada. Isso implica que deve existir um caminho, rotulado 001, do estado inicial ao estado não final. Por conveniência, esse estado não final é rotulado 001. Esse estado deve ser de morte, pois os símbolos que vierem após não trazem problemas. Todos os outros estados são estados finais ou aceitadores.

Isso nos dá a estrutura básica da solução, mas devemos, ainda, acrescentar condições para o caso da subcadeia 001 ocorrer no meio da cadeia de entrada. Devemos definir  $Q$  e  $\delta$  de tal modo que sempre que precisarmos de uma decisão correta ela seja relembrada pelo autômato. Nesse caso, quando um símbolo for lido, precisamos conhecer alguma parte da cadeia para a esquerda,

## 2.2. Linguagens e AFD's

---

por exemplo, se ou não os dois símbolos anteriores foram 00. Se rotularmos os estados com símbolos relevantes, é fácil ver quais transições devem ser. Por exemplo,  $\delta(00, 0) = 00$ , pois esta situação aparece somente se existem três 0's consecutivos. Estamos, somente, interessados nos dois últimos, o que o deixa no mesmo estado 00. A solução completa é mostrada na figura 2.4, abaixo.

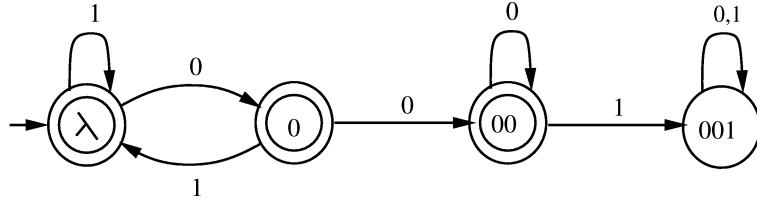


Figura 2.4: Exemplo de AFD.

**Exemplo 2.2.6** Desenhar um AFD que reconheça a linguagem,  $\mathcal{L}$ , de todas as cadeias, sobre o alfabeto  $\Sigma = \{a, b\}$ , com uma quantidade par de a's e de b's. Isto é,

$$\mathcal{L} = \{w \in \Sigma \mid N_a(w) \text{ e } N_b(w) \text{ são pares}\},$$

onde,  $N_a(w)$  ( $N_b(w)$ ) é uma função que devolve o número de símbolos a's (b's), que ocorrem em  $w$ .

Um autômato que reconheça esta linguagem deve permitir que em qualquer momento da computação possamos saber a quantidade de a's e de b's que foi lida. Mas, na verdade, só é necessário “memorizar” se até esse momento as quantidades de a's e de b's, são ambas pares, ambas ímpares, os a's são pares e os b's ímpares ou se os a's são ímpares e os b's pares. Se considerarmos 0 como par, então quando começa a computação do autômato, ele se encontra no estado no qual ambos são pares, se logo em seguida lê um a, então altera para o estado no qual a quantidade de a's são ímpares e as de b's pares, etc. Assim, o autômato deve considerar somente quatro estados, simbolizando cada uma dessas possíveis combinações. O estado par-par, é ambos, inicial e final. A figura 2.5 ilustra o grafo do autômato descrito aqui.

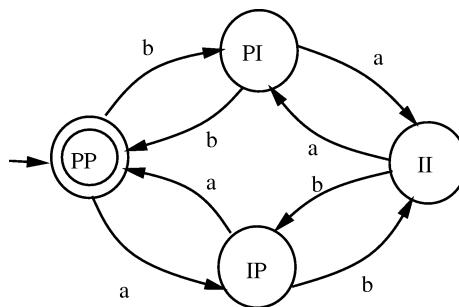


Figura 2.5: Exemplo de AFD.

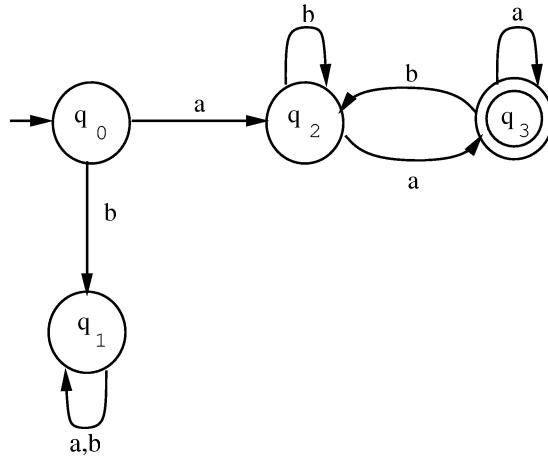


Figura 2.6: Exemplo de AFD.

Neste autômato os nomes dos estados não são os tradicionais  $q_i$ , mas  $PP$ ,  $PI$ ,  $IP$  e  $II$ . Estes nomes são mais mnemônicos, isto é, deixam em evidência o estado da máquina em termos da situação parcial da paridade de  $a$ 's e de  $b$ 's. Assim, o estado  $PP$ , simboliza a situação de uma quantidade par de  $a$ 's e de  $b$ 's, o estado  $PI$  indica uma quantidade par de  $a$ 's e ímpar de  $b$ 's, etc.

## 2.3 Linguagens Regulares

**Definição 2.3.1** Uma linguagem,  $\mathcal{L}$ , diz-se **regular** se e somente se existe algum AFD  $M$  tal que

$$\mathcal{L} = L(M).$$

**Exemplo 2.3.2** Mostre que a linguagem  $\mathcal{L} = \{awa / w \in \{a,b\}^*\}$  é regular.

Devemos achar uma AFD que reconheça a linguagem  $\mathcal{L}$ . Assim, o AFD deve checar se uma cadeia, começando com  $a$ , também termina com  $a$ . O que está no meio não é importante. A solução é complicada pelo fato de não existir um modo explícito de testar o final da cadeia. Essa dificuldade pode ser superada pondo o AFD num estado final se o segundo  $a$  for encontrado. Se este não for o último, e se depois dele vier um  $b$  então ele tirará o autômato do estado final. A pesquisa continuará dessa maneira, cada  $a$  fazendo, o autômato ir (ou continuar, se for o caso) para o estado final. A solução completa está mostrada na figura 2.6. Novamente, faça alguns exemplos para testar seu resultado.

**Exemplo 2.3.3** Seja  $\mathcal{L}$  a linguagem do exemplo anterior. Mostrar que  $\mathcal{L}^2$  é regular.

Novamente, mostraremos que a linguagem é regular construindo um AFD para ela. Podemos escrever, explicitamente, uma expressão para  $\mathcal{L}^2$ , qual seja

## 2.4. Autômatos Finitos Não-Determinísticos

---

$$\mathcal{L}^2 = \{aw_1aaw_2a / w_1, w_2 \in \{a, b\}^*\}$$

Portanto, precisamos de um AFD para reconhecer duas cadeias consecutivas de formas essencialmente iguais (mas não necessariamente idênticas em valores). O diagrama na figura 2.6 pode ser usado como ponto de início, mas o vértice  $q_3$  tem de ser modificado. Este estado não pode mais ser final pois, neste ponto, devemos começar a olhar para a segunda subcadeia da forma awa. Para reconhecer a segunda subcadeia, tomamos uma réplica dos estados na primeira parte (com novos nomes), com  $q_3$  no início da segunda parte. Como a cadeia completa pode ser quebrada em suas partes constituintes se aa ocorrer, deixamos a primeira ocorrência de aa disparar o autômato para a segunda parte. Podemos obter isso fazendo  $\delta(q_3, a) = q_4$ . A solução completa está na figura 2.7

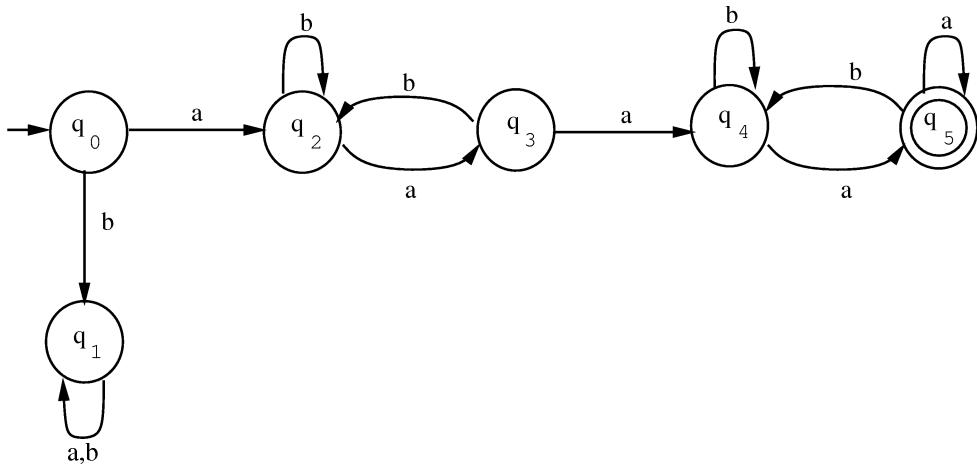


Figura 2.7: Exemplo de AFD.

Este exemplo sugere a conjectura de que se  $\mathcal{L}$  é regular, então  $\mathcal{L}^2$ ,  $\mathcal{L}^3, \dots$  também são.

## 2.4 Autômatos Finitos Não-Determinísticos

Não-determinismo significa uma escolha de movimentos para um autômato. Em vez de prescrever um único movimento em cada situação, permitimos um conjunto de movimentos possíveis. Formalmente, conseguiremos isso definindo a função de transição tal que sua imagem seja um conjunto de estados possíveis.

**Definição 2.4.1** Um autômato finito não-determinístico (AFN) é definido como a quíntupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

onde  $Q, \Sigma, q_0$  e  $F$  são definidos como para AFD, mas

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow 2^Q.$$

Observe que existem duas diferenças principais entre esta definição e a de AFD. Num autômato finito não-determinístico, a imagem de  $\delta$  é  $2^Q$  (o conjunto das partes de  $Q$ ). Logo, seu  $\delta(q, a)$  não-determiniza um único elemento de  $Q$ , mas um subconjunto dele. Este subconjunto define o conjunto de todos os estados possíveis que podem ser alcançados pela transição  $\delta(q, a)$ . Se, por exemplo, o estado corrente é  $q_1$ , está sendo lido o símbolo  $a$  e

$$\delta(q_1, a) = \{q_0, q_2\},$$

então tanto  $q_0$  quanto  $q_2$  pode ser o próximo estado do AFN. Permitimos, também,  $\lambda$  ser o segundo argumento de  $\delta$ . Isto significa que o AFN pode fazer uma transição sem consumir um símbolo de entrada. Este tipo de transição é chamada  **$\lambda$ -transição**.

Assim, embora assumimos que o mecanismo de entrada pode somente se deslocar da esquerda para a direita, é possível, aplicando  $\lambda$ -transições, que ele fique estacionado sobre alguns movimentos.

Analogamente aos AFD's, os autômatos não-determinísticos podem ser representados por grafos de transição.

Os vértices são determinados por  $Q$ , enquanto uma aresta  $(q_i, q_j)$ , com rótulo  $a \in \Sigma \cup \{\lambda\}$ , está no grafo se, e somente se,  $q_j \in \delta(q_i, a)$ . Observe que se  $a$  é a cadeia vazia, então a aresta  $(q_i, q_j)$  terá como rótulo  $\lambda$  e será chamada de  **$\lambda$ -aresta**. *lambda*-arestas são as representações gráficas de  $\lambda$ -transições e portanto muitas vezes nos referiremos a  $\lambda$ -arestas por  $\lambda$ -transições. Note que se houver no AFN uma  $\lambda$ -transição de um estado nele mesmo, essa transição seria inútil (nem consome um símbolo e nem muda de estado) e portanto parece razoável não incluí-la no grafo.

Uma **cadeia é aceita por um AFN** se existir uma possível sequência de movimentos que levará a máquina a um estado final no fim da cadeia.

**Exemplo 2.4.2** Considere o grafo de transição na figura 2.8. Ele descreve um AFN com duas transições rotuladas  $a$  saindo de  $q_0$ . Este AFN reconhece cadeias de  $a$ 's de tamanho par e a cadeia  $aab$ .

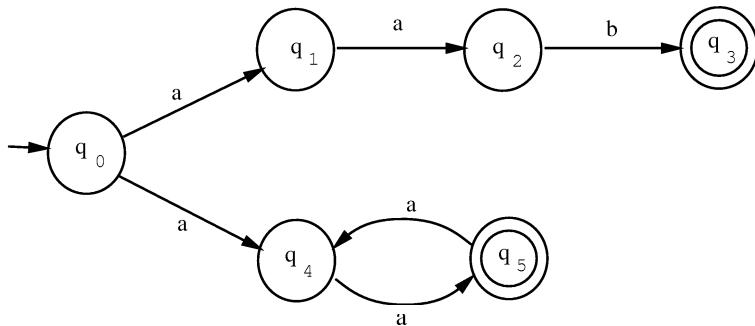


Figura 2.8: Exemplo de AFN.

## 2.4. Autômatos Finitos Não-Determinísticos

---

Observe que para qualquer cadeia começando com o símbolo  $b$ , o autômato não o reconheceria, pois no estado inicial  $q_0$  ele só admite transições rotuladas por  $a$ .

**Exemplo 2.4.3** No AFN da figura 2.9 algumas transições, tais como  $\delta(q_1, 0)$ , não são especificadas no grafo e podem ser interpretadas como uma transição ao conjunto vazio, isto é,  $\delta(q_1, 0) = \emptyset$ . O autômato reconhece as cadeias  $0001010$  e  $0101010$ , mas não  $110$ , nem  $010100$ . Observe que para  $00$  existem dois caminhos alternativos, um levando a  $q_0$  e outro a  $q_1$ . Embora  $q_0$  não seja um estado final, a cadeia é reconhecida, porque é suficiente que um caminho leve a um estado final.

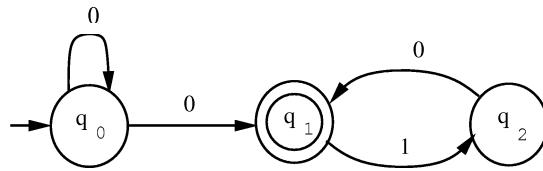


Figura 2.9: Exemplo de AFN.

**Exemplo 2.4.4** A figura 2.10 representa um AFN com  $\lambda$ -transições. Este AFN reconhece cadeias do tipo  $0^n1^m2^p$  com  $n, m, p \geq 0$ . Observe que ele aceita a cadeia  $11222$ , a qual é equivalente à cadeia  $\lambda 11\lambda 222$ . Assim, antes de consumir o primeiro  $1$ , o autômato faz uma  $\lambda$ -transição e “pula” para o estado  $q_1$ , nele consome o primeiro e o segundo  $1$ , depois novamente realiza uma  $\lambda$ -transição indo, imediatamente, para o estado  $q_2$  no qual consome os três  $2$  restantes. Como  $q_2$  é um estado final ele aceita esta cadeia.

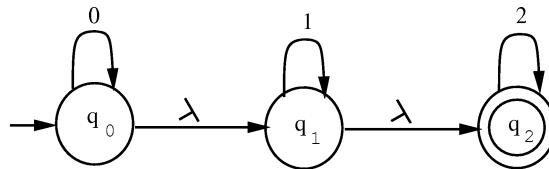


Figura 2.10: Exemplo de um AFN com  $\lambda$ -transições.

Como já tivemos oportunidade de observar para AFD's, a função de transição pode ser estendida de modo que seu segundo argumento seja uma cadeia. Porém, como dado um estado e um símbolo o autômato pode ir a mais (ou nenhum) estados,  $\delta^*(q_i, w)$  resultará no conjunto de todos os estados possíveis que o autômato não-determinístico pode chegar, tendo iniciado em  $q_i$  e tendo lido  $w$  completamente, ou seja, sem interromper o caminho no grafo por falta de definição para um símbolo de entrada em  $w$ . Por exemplo, para o AFN da figura 2.8,  $\delta^*(q_0, aa) = \{q_2, q_5\}$  e  $\delta^*(q_1, abb) = \emptyset$ .

**Definição 2.4.5** Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFN. A função de transição estendida  $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$  é uma função tal que para todo  $q_i \in Q$  e  $w \in \Sigma^*$ ,  $\delta^*(q_i, w)$  contém  $q_j \in Q$  se existe, no grafo de transição, um caminho de  $q_i$  a  $q_j$  rotulado por  $w$ .

A definição recursiva, neste caso, é bem mais complexa que aquela descrita para AFD's. A primeira dificuldade é com as  $\lambda$ -transições, pois se  $q_j \in \delta^*(q_i, w)$  e há uma  $\lambda$ -transição de  $q_j$  a  $q_k$ , então  $q_k \in \delta^*(q_i, w)$ . A noção de  **$\lambda$ -fecho** de um estado  $q$  fornece todos os estados no autômato, que podemos atingir a partir de  $q$ , sem consumir nenhuma entrada, ou seja, no máximo só usando  $\lambda$ -transições. Isto é o  $\lambda$ -fecho de um estado  $q \in Q$  é o fecho de  $\{q\}$  com respeito à operação unária  $\delta_\lambda : Q \longrightarrow \wp(Q)$  definida por  $\delta_\lambda(q) = \delta(q, \lambda)$ . Uma caracterização do  $\lambda$ -fecho de um estado é a seguinte:

$$\lambda\text{-fecho}(q) = \bigcup_{i=0}^n \lambda\text{-fecho}^i(q),$$

onde  $n = |Q| - 1$  e

$$\lambda\text{-fecho}^0(q) = \{q\}$$

$$\lambda\text{-fecho}^{i+1}(q) = \bigcup_{q' \in \lambda\text{-fecho}^i(q)} \delta(q', \lambda).$$

**Exemplo 2.4.6** Seja o AFN do exemplo 2.4.4. Então:

$$\begin{aligned} \lambda\text{-fecho}^0(q_0) &= \{q_0\} \\ \lambda\text{-fecho}^1(q_0) &= \bigcup_{q' \in \lambda\text{-fecho}^0(q_0)} \delta(q', \lambda) \\ &= \bigcup_{q' \in \{q_0\}} \delta(q', \lambda) \\ &= \delta(q_0, \lambda) = \{q_1\} \\ \lambda\text{-fecho}^2(q_0) &= \bigcup_{q' \in \lambda\text{-fecho}^1(q_0)} \delta(q', \lambda) \\ &= \bigcup_{q' \in \{q_1\}} \delta(q', \lambda) \\ &= \delta(q_1, \lambda) = \{q_2\} \end{aligned}$$

Assim, o  $\lambda$ -fecho de  $q_0$  é:

$$\begin{aligned} \lambda\text{-fecho}(q_0) &= \bigcup_{i=0}^2 \lambda\text{-fecho}^i(q_0) \\ &= \lambda\text{-fecho}^0(q_0) \cup \lambda\text{-fecho}^1(q_0) \cup \lambda\text{-fecho}^2(q_0) \\ &= \{q_0\} \cup \{q_1\} \cup \{q_2\} = \{q_0, q_1, q_2\} \end{aligned}$$

**Exemplo 2.4.7** Considere o AFN ilustrado na figura 2.11.

Então o  $\lambda$ -fecho do estado  $q_1$  é:

$$\begin{aligned} \lambda\text{-fecho}^0(q_1) &= \{q_1\} \\ \lambda\text{-fecho}^1(q_1) &= \bigcup_{q' \in \lambda\text{-fecho}^0(q_1)} \delta(q', \lambda) \\ &= \bigcup_{q' \in \{q_1\}} \delta(q', \lambda) \\ &= \delta(q_1, \lambda) = \{q_2, q_5\} \\ \lambda\text{-fecho}^2(q_1) &= \bigcup_{q' \in \lambda\text{-fecho}^1(q_1)} \delta(q', \lambda) \\ &= \bigcup_{q' \in \{q_2, q_5\}} \delta(q', \lambda) \\ &= \delta(q_2, \lambda) \cup \delta(q_5, \lambda) \\ &= \{q_3, q_5\} \cup \emptyset = \{q_3, q_5\} \end{aligned}$$

## 2.4. Autômatos Finitos Não-Determinísticos

---

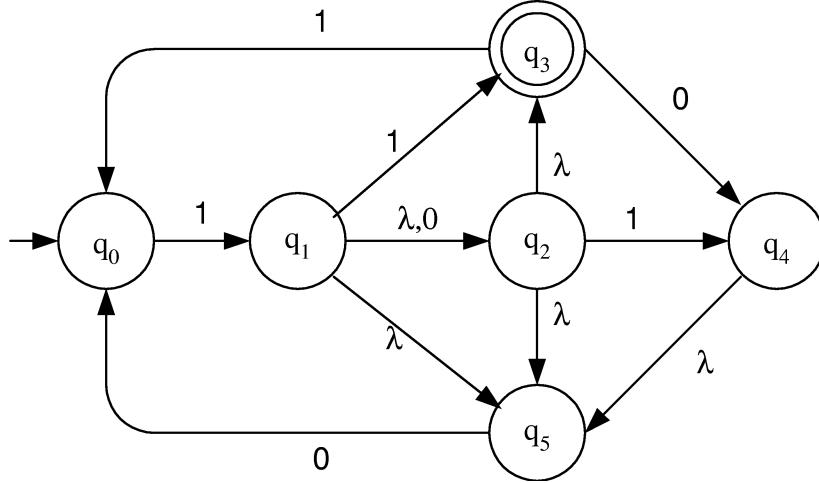


Figura 2.11: Exemplo de um AFN com  $\lambda$ -transições.

$$\begin{aligned}
 \lambda - fecho^3(q_1) &= \bigcup_{q' \in \lambda - fecho^2(q_1)} \delta(q', \lambda) \\
 &= \bigcup_{q' \in \{q_3, q_5\}} \delta(q', \lambda) \\
 &= \delta(q_3, \lambda) \cup \delta(q_5, \lambda) \\
 &= \emptyset \cup \emptyset = \emptyset
 \end{aligned}$$

$$\begin{aligned}
 \lambda - fecho^4(q_1) &= \bigcup_{q' \in \lambda - fecho^3(q_1)} \delta(q', \lambda) \\
 &= \bigcup_{q' \in \emptyset} \delta(q', \lambda) \\
 &= \emptyset
 \end{aligned}$$

$$\begin{aligned}
 \lambda - fecho^5(q_1) &= \bigcup_{q' \in \lambda - fecho^4(q_1)} \delta(q', \lambda) \\
 &= \bigcup_{q' \in \emptyset} \delta(q', \lambda) \\
 &= \emptyset
 \end{aligned}$$

Assim, o  $\lambda$ -fecho de  $q_1$  é:

$$\begin{aligned}
 \lambda - fecho(q_1) &= \bigcup_{i=0}^5 \lambda - fecho^i(q_1) \\
 &= \lambda - fecho^0(q_1) \cup \lambda - fecho^1(q_1) \cup \lambda - fecho^2(q_1) \cup \\
 &\quad \lambda - fecho^3(q_1) \cup \lambda - fecho^4(q_1) \cup \lambda - fecho^5(q_1) \\
 &= \{q_1\} \cup \{q_2, q_5\} \cup \{q_3, q_5\} \cup \emptyset \cup \emptyset \cup \emptyset \\
 &= \{q_1, q_2, q_3, q_5\}
 \end{aligned}$$

Claramente, o  $\lambda - fecho$  é uma função de  $Q$  em  $\wp(Q)$  e pode ser trivialmente estendida para  $\lambda - Fecho : \wp(Q) \rightarrow \wp(Q)$  da seguinte forma:

$$\lambda - Fecho(X) = \bigcup_{q \in X} \lambda - fecho(q)$$

Assim, agora, podemos definir recursivamente  $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$  como,

$$\delta^*(q, \lambda) = \lambda - fecho(q) \tag{2.2}$$

$$\delta^*(q, wa) = \bigcup_{q_i \in \delta^*(q, w)} \lambda\text{-Fecho}(\delta(q_i, a)). \quad (2.3)$$

Observe que caso o AFN não possua  $\lambda$ -transições, estas equações podem ser simplificadas para

$$\delta^*(q, \lambda) = q \quad (2.4)$$

$$\delta^*(q, wa) = \bigcup_{q_i \in \delta^*(q, w)} \delta(q_i, a). \quad (2.5)$$

**Exemplo 2.4.8** Considere o grafo de transição na figura 2.12.

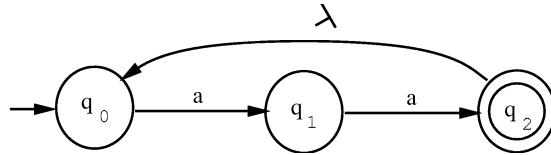


Figura 2.12: Exemplo de um AFN com  $\lambda$ -transições.

Suponha que queremos achar  $\delta^*(q_0, aa)$  e  $\delta^*(q_2, \lambda)$ . Assim, se sairmos de  $q_0$  consumindo o primeiro “a” chegaremos a  $q_1$ , consumindo o outro “a” chegaremos a  $q_2$ , mas como existe uma  $\lambda$ -transição de  $q_2$  a  $q_0$ , também, podemos ir ao estado  $q_0$ , sem necessidade de consumir nada. Portanto,

$$\delta^*(q_0, aa) = \{q_0, q_2\}.$$

Como existe um  $\lambda$ -aresta entre  $q_2$  a  $q_0$ , temos imediatamente que  $\delta^*(q_2, \lambda)$  contém  $q_0$ . Além disso, embora não mostramos explicitamente, existe sempre uma  $\lambda$ -transição implícita de um estado nele mesmo, representando o fato que num autômato quando não se consume um símbolo de entrada sempre existe a possibilidade de se permanecer no mesmo estado. Portanto,

$$\delta^*(q_2, \lambda) = \{q_0, q_2\}.$$

Agora, determinaremos  $\delta^*(q_0, aa)$  seguindo a definição dada pelas equações 2.2 e 2.3.

## 2.4. Autômatos Finitos Não-Determinísticos

---

$$\begin{aligned}
\delta^*(q_0, \lambda) &= \{q_0\} \\
\delta^*(q_0, a) &= \bigcup_{q_i \in \delta^*(q_0, \lambda)} \lambda - \text{Fecho}(\delta(q_i, a)) \\
&= \bigcup_{q_i \in \{q_0\}} \lambda - \text{Fecho}(\delta(q_i, a)) \\
&= \lambda - \text{Fecho}(\delta(q_0, a)) \\
&= \lambda - \text{Fecho}(\{q_1\}) \\
&= \lambda - \text{fecho}(q_1) \\
&= \{q_1\} \\
\delta^*(q_0, aa) &= \bigcup_{q_i \in \delta^*(q_0, a)} \lambda - \text{Fecho}(\delta(q_i, a)) \\
&= \bigcup_{q_i \in \{q_1\}} \lambda - \text{Fecho}(\delta(q_i, a)) \\
&= \lambda - \text{Fecho}(\delta(q_1, a)) \\
&= \lambda - \text{Fecho}(\{q_2\}) \\
&= \lambda - \text{fecho}(q_2) \\
&= \{q_0, q_2\}
\end{aligned}$$

**Exemplo 2.4.9** Considere o AFN do exemplo 2.4.7 (figura 2.11). Determinaremos  $\delta^*(q_0, 10)$

$$\begin{aligned}
\delta^*(q_0, \lambda) &= \{q_0\} \\
\delta^*(q_0, 1) &= \bigcup_{q_i \in \delta^*(q_0, \lambda)} \lambda - \text{Fecho}(\delta(q_i, 1)) \\
&= \bigcup_{q_i \in \{q_0\}} \lambda - \text{Fecho}(\delta(q_i, 1)) \\
&= \lambda - \text{Fecho}(\delta(q_0, 1)) \\
&= \lambda - \text{Fecho}(\{q_1\}) \\
&= \lambda - \text{fecho}(q_1) \\
&= \{q_1, q_2, q_3, q_5\} \\
\delta^*(q_0, 10) &= \bigcup_{q_i \in \delta^*(q_0, 1)} \lambda - \text{Fecho}(\delta(q_i, 0)) \\
&= \bigcup_{q_i \in \{q_1, q_2, q_3, q_5\}} \lambda - \text{Fecho}(\delta(q_i, 0)) \\
&= \lambda - \text{Fecho}(\delta(q_1, 0)) \cup \lambda - \text{Fecho}(\delta(q_2, 0)) \cup \lambda - \text{Fecho}(\delta(q_3, 0)) \cup \lambda - \text{Fecho}(\delta(q_5, 0)) \\
&= \lambda - \text{Fecho}(\{q_2\}) \cup \lambda - \text{Fecho}(\emptyset) \cup \lambda - \text{Fecho}(\{q_4\}) \cup \lambda - \text{Fecho}(\{q_0\}) \\
&= \lambda - \text{fecho}(q_2) \cup \lambda - \text{fecho}(q_4) \cup \lambda - \text{fecho}(q_0) \\
&= \{q_3, q_5\} \cup \{q_5\} \cup \emptyset \\
&= \{q_3, q_5\}
\end{aligned}$$

**Definição 2.4.10** A linguagem,  $\mathcal{L}$ , reconhecida ou aceita por um AFN,  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , é definida como o conjunto de todas as cadeias reconhecidas por  $M$ , no sentido dado anteriormente. Formalmente,

$$L(M) = \{w \in \Sigma^* / \delta^*(q_0, w) \cap F \neq \emptyset\}.$$

Em palavras, a linguagem contém todas as cadeias,  $w$ , para as quais existe um caminho, rotulado  $w$ , do vértice inicial do grafo de transição a algum vértice final.

**Exemplo 2.4.11** Qual é a linguagem reconhecida pelo autômato na figura 2.9?

É fácil ver que a única maneira para o AFN parar no estado final é quando a entrada é uma repetição de 0's (com pelo menos um 0) seguido por sucessivas repetições (possivelmente nenhuma) da cadeia 10. Assim, o autômato reconhece a linguagem  $\mathcal{L} = \{0^m(10)^n / m > 0 \text{ e } n \geq 0\}$ .

O que acontece quando a cadeia  $w = 0100$  é apresentada ao autômato? Após ler o prefixo  $010$ , o autômato se encontra no estado  $q_1$ , com a transição  $\delta(q_1, 0)$  indefinida. Chamamos tal situação de uma **configuração de morte**, e podemos visualizá-la como o autômato, simplesmente, parando sem outra ação. Mas, devemos ter em mente que tais visualizações são imprecisas e podem levar a uma interpretação errada. O que queremos dizer, precisamente, é que

$$\delta^*(q_0, 0100) = \emptyset.$$

Portanto, nenhum estado final pode ser atingido em se processando  $w = 0100$ , e consequentemente a cadeia não é reconhecida.

### Por que não-determinismo?

Quando raciocinamos sobre máquinas não-determinísticas devemos ter cautela com o uso de noções intuitivas, pois a nossa intuição pode nos enganar. Devemos ser capazes de fornecer argumentos precisos para fundamentar as conclusões.

Não determinismo é um conceito difícil. Os computadores digitais são completamente determinísticos, seu estado em qualquer tempo é previsto unicamente da entrada e do estado inicial. Portanto, é natural se indagar porque estudamos máquinas não-determinísticas. Estamos tentando modelar sistemas reais, porque, então, incluir tais características, não mecânicas? Poderemos responder essa questão de várias maneiras.

Muitos algoritmos determinísticos requerem que se faça uma escolha em algum estágio. Um exemplo típico é um programa para “jogar”. Em general, o melhor movimento não é conhecido, mas pode ser encontrado usando-se uma pesquisa exaustiva com *backtracking*. Segundo várias alternativas sejam possíveis, escolhemos uma e seguimos-a até tornar-se claro se ela é a melhor alternativa ou não. Se não voltaremos ao ponto da última decisão e exploraremos as outras escolhas. Um algoritmo não-determinístico que pode fazer a melhor escolha seria capaz de resolver o problema sem *backtracking*, mas um determinístico pode simular não-determinismo com algum trabalho extra. Por essa razão, máquinas não-determinísticas podem servir como modelos de algoritmos de pesquisa *backtracking*.

Não determinismo é, às vezes, útil para se resolver facilmente um problema. Compare o exemplo 2.4.4 e a solução determinista “mais simples” para reconhecer a mesma linguagem, isto é, as cadeias da forma  $0^m 1^n 2^p$ , mostrada na figura 2.13.

É preciso fazer um escolha no AFN da figura 2.8. A primeira alternativa leva ao reconhecimento da cadeia  $aab$ , enquanto a segunda reconhece todas as cadeias, com um número par de  $a$ 's. Assim, a linguagem reconhecida por este AFN é  $\{aab\} \cup \{a^{2n} / n \geq 1\}$ . Embora seja possível achar um AFD que reconheça essa linguagem, o não-determinismo é bastante natural. A linguagem é a união de dois conjuntos, bastante diferentes, e o não-determinismo nos permite decidir no conjunto de saída, qual caso queremos. A solução determinística não está tão obviamente relacionada à definição. Assim, não-determinismo nós permite soluções naturais e “enxutas” para reconhecer certas linguagens.

## 2.5. Equivalência entre AFD's e AFN's

---

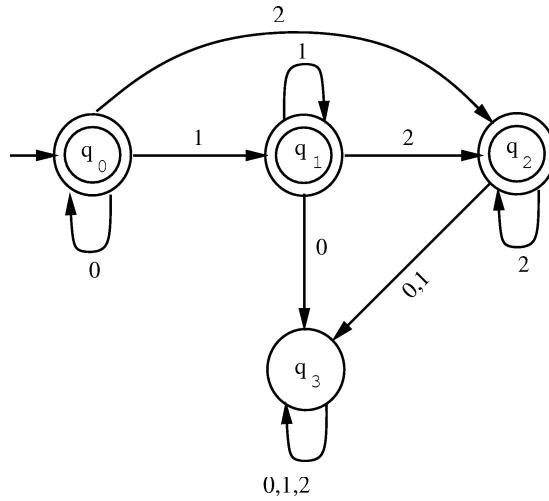


Figura 2.13: afd equivalente ao AFN do exemplo 2.4.4.

Também, o não-determinismo é um mecanismo efetivo para descrever, concisamente, algumas linguagens complicadas. Note que a definição de uma gramática envolve um elemento não-determinístico. Por exemplo, nas produções

$$S \longrightarrow aSb \mid \lambda,$$

podemos em qualquer substituir em uma cadeia contendo pelo menos uma ocorrência da subcadeia  $S$  essa ocorrência de  $S$  por  $aSb$  ou por  $\lambda$ , ou seja usar a primeira ou a segunda produção, o que nos permite especificar muitas cadeias diferentes usando somente duas regras.

Finalmente, existe uma razão técnica para introduzir não-determinismo. Como veremos adiante, certos resultados são mais facilmente estabelecidos para AFN's do que para AFD's. No que segue veremos que não existe diferença essencial entre esses dois tipos de autômatos (determinísticos e não-determinísticos). Conseqüentemente, permitindo não-determinismo, muitas vezes, conseguimos simplificar argumentos formais sem afetar a generalidade da conclusão.

## 2.5 Equivalência entre AFD's e AFN's

Voltemos, agora, à questão fundamental. Em que sentido os AFD's e os AFN's são diferentes? É claro que existe uma diferença em sua definição, mas isso não implica que exista distinção essencial entre eles. Para explorar essa questão, introduziremos a noção de equivalência entre dois autômatos. Dizemos que dois reconhecedores são *equivalentes* se eles reconhecem a mesma linguagem. Para uma dada linguagem podemos, usualmente, achar um número ilimitado de reconhecedores equivalentes, determinísticos ou não.

**Exemplo 2.5.1** O AFD mostrado na figura 2.14 é equivalente ao AFN da figura 2.9, pois ambos aceitam a linguagem  $\{0^m(10)^n \mid m \geq 1 \text{ e } n \geq 0\}$ .

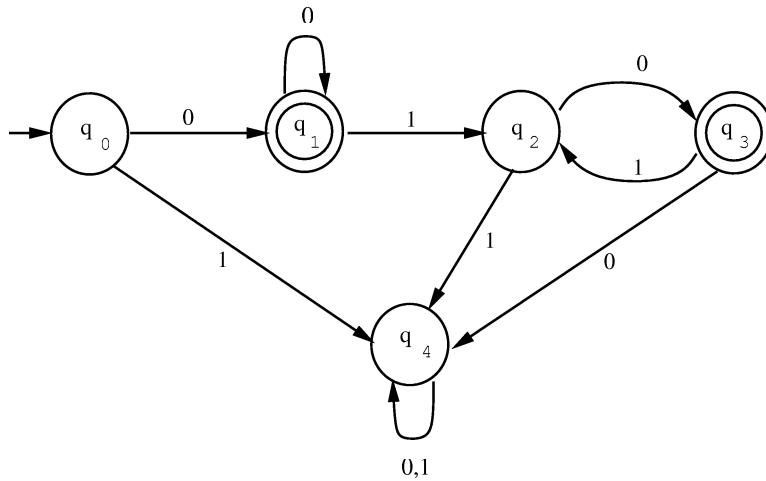


Figura 2.14: afd equivalente à AFN da figura 2.9

Quando comparamos diferentes classes de autômatos, sempre nos questionamos se uma classe é mais potente do que outra ou não. Por mais potente queremos significar que existe um autômato da espécie  $\mathcal{C}$  que pode realizar algo que não pode ser feito por qualquer autômato da outra espécie, digamos  $\mathcal{D}$ , e que todo o que é realizado por um autômato da espécie  $\mathcal{D}$  sempre pode ser realizado por alguém da espécie  $\mathcal{C}$ . Olhemos essa questão para o caso de autômatos finitos. Um AFD é, essencialmente um AFN, pois basta definir  $\delta_N(q, a) = \{\delta_D(q, a)\}$  e  $\delta_N(q, \lambda) = \emptyset$ , para todo  $a \in \Sigma$  e  $q \in Q$ , onde  $\delta_D$  e  $\delta_N$  são as funções de transição do AFD e AFN, respectivamente, permanecendo o resto inalterado (de fato o diagrama de transição continua o mesmo). É claro que qualquer linguagem aceita por um AFD também é aceita por algum AFN. Mas, a recíproca não é tão óbvia. Adicionamos não-determinismo, de modo que é concebível que exista uma linguagem aceita por AFN para a qual não possamos achar um AFD. Mas acontece que isso não é o caso. A classe dos AFD's e AFN's são igualmente poderosas: para qualquer linguagem aceita por um AFN existe um AFD que aceita a mesma linguagem.

Esse resultado não é óbvio e, portanto, tem que ser demonstrado. O argumento será construtivo. Isso significa que devemos dar uma maneira de converter qualquer AFN num AFD equivalente. O argumento é essencialmente o seguinte: após um AFN ter lido uma cadeia  $w$ , podemos não saber exatamente em que estado ele entrará, porém podemos dizer que ele deve ser um dos estados possíveis de um conjunto de estados, digamos  $X = \{q_i / i \in I\}$  para algum  $I \subseteq \{0, 1, \dots, |Q|\}$ . Um AFD equivalente lendo a mesma cadeia deve entrar em algum estado bem definido. O que podemos fazer para que essas duas situações se correspondam? A resposta é a seguinte: rotule os estados do AFD com um conjunto de estados de tal modo que, após ler  $w$ , o AFD equivalente entrará num único estado rotulado  $X$ . Como para um conjunto de  $|Q|$  estados existem exatamente  $2^{|Q|}$  subconjuntos, o AFD correspondente terá no máximo  $2^{|Q|}$  estados.

**Exemplo 2.5.2** Converter o AFN da figura 2.15 (o qual é equivalente ao AFN da figura 2.9) num AFD equivalente. Considere que a função de transição do AFN é  $\delta_N$ .

O AFN inicia no estado  $q_0$ , mas como ele tem uma  $\lambda$ -transição de  $q_0$  a  $q_1$ , ele pode também se comportar como se iniciasse no estado  $q_1$ . Assim, o estado inicial do AFD é  $\{q_0, q_1\}$ , isto é

## 2.5. Equivalência entre AFD's e AFN's

---

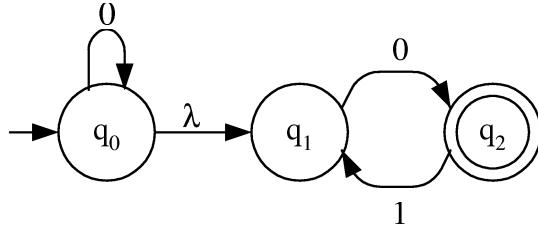


Figura 2.15: Exemplo de AFN.

*o  $\lambda$ -fecho de  $q_0$ . Se o autômato lê um 0 e considerarmos como estado corrente  $q_0$ , o AFN pode permanecer em  $q_0$ , ou produto da  $\lambda$ -transição, pode ir (sem consumir nenhum símbolo) para o estado  $q_1$ . Se considerarmos que o autômato estava em  $q_1$  o AFD muda de estado para  $q_2$ . Portanto, estando no estado  $\{q_0, q_1\}$  o AFD, após ler um 0, deve ir para o estado  $\{q_0, q_1, q_2\}$ . Logo, o AFD deve ter um estado rotulado por  $\{q_0, q_1, q_2\}$  e uma transição*

$$\delta(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

*Nem no estado  $q_0$  nem no  $q_1$  o AFN tem transição especificada quando a entrada é 1, assim*

$$\delta(\{q_0, q_1\}, 1) = \emptyset.$$

*Um estado rotulado  $\emptyset$  representa um movimento impossível para o AFN e, portanto significa não reconhecimento da cadeia. Conseqüentemente, este estado no AFD deve ser um estado não-final de morte.*

*Agora, devemos analisar o comportamento da AFD para o novo estado  $\{q_0, q_1, q_2\}$ . Assim, precisamos achar as transições saindo deste estado. Lembre que este estado do AFD corresponde a três estados no AFN, portanto para determinar o comportamento do AFD para este estado devemos considerar o comportamento do AFN para cada um desses estados ( $q_0$ ,  $q_1$  e  $q_2$ ). Se o AFN está no estado  $q_0$  e lê um 0, ele pode ficar em  $q_0$ , ou ir para o estado  $q_1$  ou  $q_2$ . Se o AFN está no estado  $q_1$  e lê um 0, muda para o estado  $q_2$  e se o AFN está no estado  $q_2$  e lê um 0, o AFN não faz nada. Lembre também que devemos considerar as  $\lambda$ -transições. Portanto,*

$$\begin{aligned}
 \delta(\{q_0, q_1, q_2\}, 0) &= \lambda - \text{Fecho}(\delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)) \\
 &= \lambda - \text{Fecho}(\{q_0\} \cup \{q_2\} \cup \emptyset) \\
 &= \lambda - \text{Fecho}(\{q_0, q_2\}) \\
 &= \lambda - \text{fecho}(q_0) \cup \lambda - \text{fecho}(q_2) \\
 &= \{q_0, q_1, q_2\}.
 \end{aligned}$$

*Analogamente,*

$$\begin{aligned}
 \delta(\{q_0, q_1, q_2\}, 1) &= \lambda - \text{Fecho}(\delta_N(q_0, 1) \cup \delta_N(q_1, 1) \cup \delta_N(q_2, 1)) \\
 &= \lambda - \text{Fecho}(\emptyset \cup \emptyset \cup \{q_1\}) \\
 &= \lambda - \text{fecho}(q_1) \\
 &= \{q_1\}.
 \end{aligned}$$

De  $q_1$  se entra um 0 ele vai para  $q_2$  e se entra um 1 não tem transição especificada, portanto ele vai para um estado de morte  $\emptyset$ . Logo,

$$\delta(\{q_1\}, 0) = \lambda - Fecho(\delta_N(q_1, 0)) = \lambda - fecho(q_2) = \{q_2\}$$

$$\delta(\{q_1\}, 1) = \lambda - Fecho(\delta_N(q_1, 0)) = \lambda - Fecho(\emptyset) = \emptyset.$$

Analogamente, para o estado  $\{q_2\}$  temos as seguintes transições:

$$\delta(\{q_2\}, 0) = \lambda - Fecho(\delta_N(q_2, 0)) = \lambda - Fecho(\emptyset) = \emptyset$$

$$\delta(\{q_2\}, 1) = \lambda - Fecho(\delta_N(q_2, 1)) = \lambda - fecho(q_1) = \{q_1\}.$$

Neste ponto, todo estado tem todas as transições definidas. O resultado é mostrado na figura 2.16, o qual é um AFD equivalente ao AFN, com o qual iniciamos. O AFN da figura 2.15 aceita (reconhece) qualquer cadeia  $w$  para o qual  $\delta^*(q_0, w)$  contém  $q_2$ . Para o correspondente AFD reconhecer tais  $w$ , qualquer estado cujo rótulo inclui  $q_2$  deve se tornar um estado final.

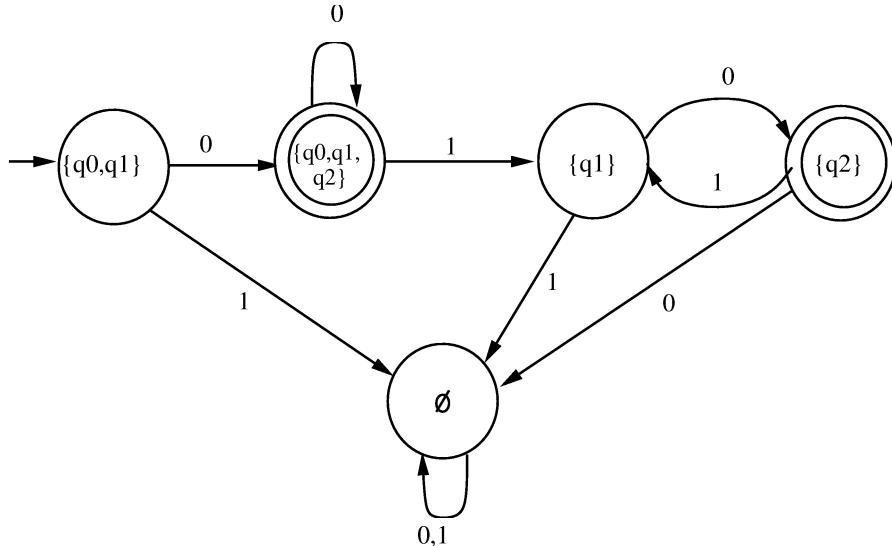


Figura 2.16: afd equivalente ao AFN da figura 2.15.

Observe, que o autômato da figura 2.16 é idêntico, a menos dos rótulos dos estados, ao AFD da figura 2.14. Isto porque o AFN do qual partimos (figura 2.15) é equivalente ao AFN da figura 2.9 que, por sua vez, é equivalente ao AFD da figura 2.14.

**Teorema 2.5.3** Seja  $L(M_N)$  a linguagem reconhecida pelo AFN  $M_N = \langle Q_N, \Sigma, \delta_N, q_0, F_N \rangle$ . Então existe um AFD  $M_D = \langle Q_D, \Sigma, \delta_D, q'_0, F_D \rangle$  tal que  $L(M_N) = L(M_D)$ .

## 2.5. Equivalência entre AFD's e AFN's

---

DEMONSTRAÇÃO: Seja  $M_D = \langle Q_D, \Sigma, \delta_D, q'_0, F_D \rangle$ , onde

- $Q_D = \{X / X \subseteq Q\}$
- $\delta_D : 2^Q \times \Sigma \longrightarrow 2^Q$  é definida por

$$\delta_D(X, a) = \bigcup_{q \in X} \lambda\text{-}Fecho(\delta_N(q, a))$$

- $q'_0 = \lambda\text{-}fecho(q_0)$
- $F_D = \{X \subseteq Q / X \cap F_N \neq \emptyset\}$

Este  $M_D$  é equivalente ao AFN  $M_N$ . No entanto, ele pode incluir estados que não podem ser atingidos a partir do estado inicial de  $M_D$ . Para considerar somente os estados relevantes ao AFD equivalente, usaremos o procedimento AFN-para-afd, para construir o grafo de transição  $G_D$ , para  $M_D$ . Para entender a construção lembre que  $G_D$  tem de ter certas propriedades. Todo vértice deve ter exatamente  $|\Sigma|$  arestas saindo dele, cada um rotulado com um elemento diferente de  $\Sigma$ . Durante a construção, algumas das arestas podem ser perdidas, mas o procedimento continua até todas elas serem encontradas.

**Procedimento:** AFN-para-afd.

1. Crie um grafo  $G_D$  com vértice  $\lambda\text{-}fecho(q_0)$ . Identifique este vértice como o vértice inicial.
  2. Repita as seguintes etapas até não faltar mais arestas.
    - (a) Tome qualquer vértice  $X$ , de  $G_D$ , que não tenha uma aresta com rótulo  $a$  saindo dele, para algum  $a \in \Sigma$ .
    - (b) Compute  $\delta_N(q, a)$ , para todo  $q \in X$ , e gere a união de todos esses conjuntos de estados, em seguida faça o  $\lambda\text{-}fecho$  desse conjunto, gerando o conjunto  $Y \subseteq Q_N$ , isto é,
- $$Y = \lambda\text{-}Fecho(\bigcup_{q \in X} \delta_N(q, a)).$$
- (c) Crie um vértice para  $G_D$ , rotulado com  $Y$  se ele já não existir.
  - (d) Adicione uma aresta de  $X$  para  $Y$  e rotule-a com  $a$ .
3. Todo estado de  $G_D$  cujo rótulo contém qualquer  $q \in F_N$  é identificado como um vértice final. ■

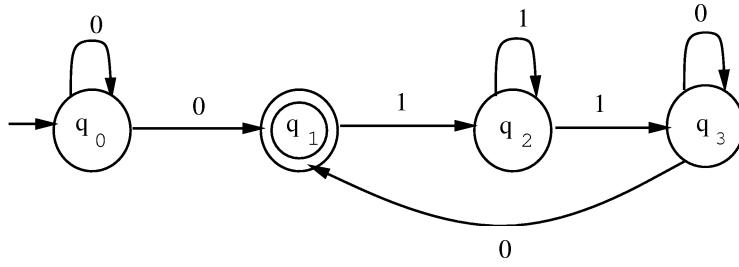


Figura 2.17: Exemplo de AFN.

**Exemplo 2.5.4** Converter o AFN na figura 2.17 num AFD equivalente. Como não temos  $\lambda$ -transições, o  $\lambda$ -fecho de  $q_0$  é  $\{q_0\}$ . Assim, introduziremos primeiro, o estado inicial  $\{q_0\}$  em  $G_D$ . Como  $\delta_N(q_0, 0) = \{q_0, q_1\}$ , introduzimos o estado  $\{q_0, q_1\}$  em  $G_D$  e adicionamos uma aresta rotulada 0 entre  $\{q_0\}$  e  $\{q_0, q_1\}$ , isto é,  $\delta_D(\{q_0\}, 0) = \{q_0, q_1\}$ . Da mesma maneira, como  $\delta_N(q_0, 1) = \emptyset$ , incorporamos o novo estado de morte  $\emptyset$  e uma aresta, rotulada 1, entre  $\{q_0\}$  e ele.

Existem, agora, algumas arestas faltando, portanto continuemos usando o algoritmo do teorema 2.5.3. Assim, calculamos

$$\begin{aligned}\delta_D(\{q_0, q_1\}, 0) &= \lambda - \text{Fecho}(\delta_N(q_0, 0) \cup \delta_N(q_1, 0)) \\ &= \lambda - \text{Fecho}(\{q_0, q_1\} \cup \emptyset) \\ &= \{q_0, q_1\}\end{aligned}$$

$$\begin{aligned}\delta_D(\{q_0, q_1\}, 1) &= \lambda - \text{Fecho}(\delta_N(q_0, 1) \cup \delta_N(q_1, 1)) \\ &= \lambda - \text{Fecho}(\emptyset \cup \{q_2\}) \\ &= \{q_2\}\end{aligned}$$

Isso nos dá o novo estado  $\{q_2\}$ . Como o novo estado  $\{q_2\}$  não tem arestas saindo dele, aplicamos novamente o algoritmo.

$$\delta_D(\{q_2\}, 0) = \lambda - \text{Fecho}(\delta_N(q_2, 0)) = \lambda - \text{Fecho}(\emptyset) = \emptyset$$

e

$$\delta_D(\{q_2\}, 1) = \lambda - \text{Fecho}(\delta_N(q_2, 1)) = \lambda - \text{Fecho}(\{q_2, q_3\}) = \{q_2, q_3\}$$

Aplicando o algoritmo para o novo estado  $\{q_2, q_3\}$ , temos que adicionar as arestas

$$\begin{aligned}\delta_D(\{q_2, q_3\}, 0) &= \lambda - \text{Fecho}(\delta_N(q_2, 0) \cup \delta_N(q_3, 0)) \\ &= \lambda - \text{Fecho}(\emptyset \cup \{q_1, q_3\}) \\ &= \{q_1, q_3\}\end{aligned}$$

e

## 2.5. Equivalência entre AFD's e AFN's

---

$$\begin{aligned}\delta_D(\{q_2, q_3\}, 1) &= \lambda - Fecho(\delta_N(q_2, 1) \cup \delta_N(q_3, 1)) \\ &= \lambda - Fecho(\{q_2, q_3\} \cup \emptyset) \\ &= \{q_2, q_3\}\end{aligned}$$

Aplicando o algoritmo para o novo estado  $\{q_1, q_3\}$ , temos que adicionar as arestas

$$\begin{aligned}\delta_D(\{q_1, q_3\}, 0) &= \lambda - Fecho(\delta_N(q_1, 0) \cup \delta_N(q_3, 0)) \\ &= \lambda - Fecho(\emptyset \cup \{q_1, q_3\}) \\ &= \{q_1, q_3\}\end{aligned}$$

e

$$\begin{aligned}\delta_D(\{q_1, q_3\}, 1) &= \lambda - Fecho(\delta_N(q_1, 1) \cup \delta_N(q_3, 1)) \\ &= \lambda - fecho(\{q_2\} \cup \emptyset) \\ &= \{q_2\}\end{aligned}$$

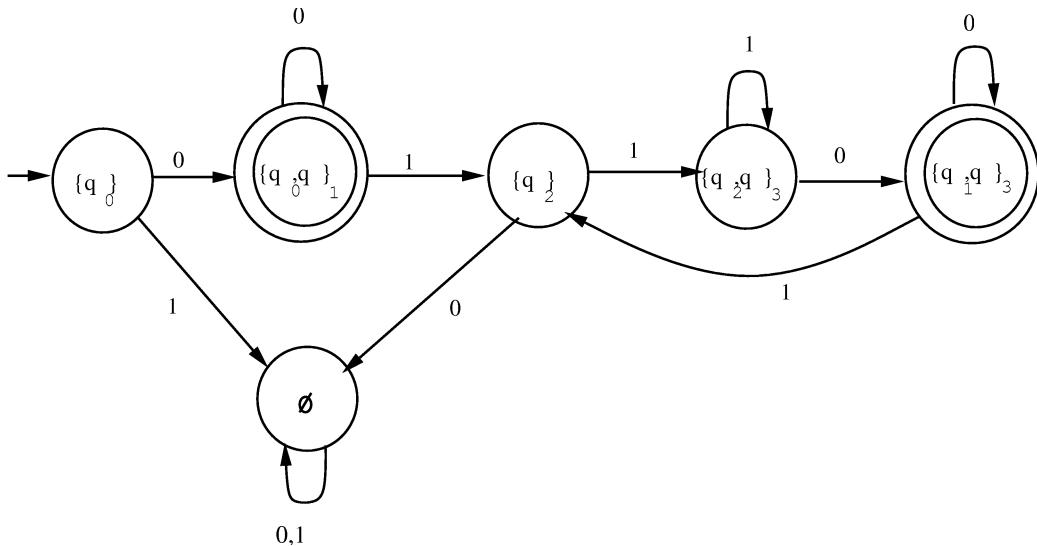


Figura 2.18: AFD equivalente ao AFN da figura 2.17.

Como não se gerou nenhum novo estado e portanto todos os vértices em  $G_D$  têm arestas saindo rotuladas por 0 e 1, paramos o processo de geração de arestas e estados. O último ponto é identificar os estados finais. Como o único estado final em 2.17 é o estado  $q_1$ , em  $G_D$  teremos como estados finais os estados  $\{q_0, q_1\}$  e  $\{q_1, q_3\}$ . A figura 2.18 mostra o AFD resultante da aplicação do algoritmo AFN-para-AFD.

**Exemplo 2.5.5** Converter o AFN da figura 2.19 num AFD equivalente. Como temos uma  $\lambda$ -transição de  $q_0$  a  $q_1$  e outra  $\lambda$ -transição de  $q_1$  a  $q_2$ , então nesse autômato podemos atingir  $q_1$  e  $q_2$  a partir de  $q_0$  sem consumir nenhum símbolo. Assim, o  $\lambda$ -fecho de  $q_0$  é  $\{q_0, q_1, q_2\}$  e portanto introduziremos o estado inicial  $\{q_0, q_1, q_2\}$  em  $G_D$ . Agora iremos completando o AFD usando o algoritmo AFN-para-AFD.

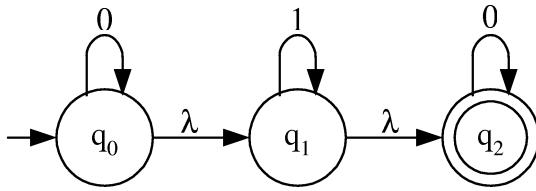


Figura 2.19: Exemplo de AFN.

$$\begin{aligned}
 \delta_D(\{q_0, q_1, q_2\}, 0) &= \lambda\text{-}Echo(\delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)) \\
 &= \lambda\text{-}Echo(\{q_0\} \cup \emptyset \cup \{q_2\}) \\
 &= \lambda\text{-}Echo(\{q_0, q_2\}) \\
 &= \{q_0, q_1, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta_D(\{q_0, q_1, q_2\}, 1) &= \lambda\text{-}Echo(\delta_N(q_0, 1) \cup \delta_N(q_1, 1) \cup \delta_N(q_2, 1)) \\
 &= \lambda\text{-}Echo(\emptyset \cup \{q_1\} \cup \emptyset) \\
 &= \lambda\text{-}Echo(\{q_1\}) \\
 &= \{q_1, q_2\}
 \end{aligned}$$

Este passo do algoritmo gerou o novo estado  $\{q_1, q_2\}$ . Portanto, repetimos o passo 2 do algoritmo.

$$\begin{aligned}
 \delta_D(\{q_1, q_2\}, 0) &= \lambda\text{-}Echo(\delta_N(q_1, 0) \cup \delta_N(q_2, 0)) \\
 &= \lambda\text{-}Echo(\emptyset \cup \{q_2\}) \\
 &= \lambda\text{-}Echo(\{q_2\}) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta_D(\{q_1, q_2\}, 1) &= \lambda\text{-}Echo(\delta_N(q_1, 1) \cup \delta_N(q_2, 1)) \\
 &= \lambda\text{-}Echo(\{q_1\} \cup \emptyset) \\
 &= \lambda\text{-}Echo(\{q_1\}) \\
 &= \{q_1, q_2\}
 \end{aligned}$$

Este passo gerou outro estado novo:  $\{q_2\}$ . Portanto, devemos continuar aplicando o passo 2 do algoritmo.

$$\begin{aligned}
 \delta_D(\{q_2\}, 0) &= \lambda\text{-}Echo(\delta_N(q_2, 0)) \\
 &= \lambda\text{-}Echo(\{q_2\}) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta_D(\{q_2\}, 1) &= \lambda\text{-}Echo(\delta_N(q_2, 1)) \\
 &= \lambda\text{-}Echo(\emptyset) \\
 &= \emptyset
 \end{aligned}$$

Este passo gerou outro estado novo:  $\emptyset$ . Portanto, devemos continuar aplicando o passo 2 do algoritmo.

## 2.6. Algoritmo da Minimização de Estados

---

$$\begin{aligned}\delta_D(\emptyset, 0) &= \emptyset \\ \delta_D(\emptyset, 1) &= \emptyset\end{aligned}$$

Como não foi gerado nenhum novo estado, então o passo 2 do algoritmo está concluído. No passo 3, só devemos identificar os estados finais, isto é, todos aqueles estados que contém algum estado final do autômato original, ou seja  $q_2$ . Portanto, os estados finais do novo autômato serão todos os estados a menos do estado  $\emptyset$ . A figura 2.20 mostra o autômato finito determinístico resultante deste algoritmo.

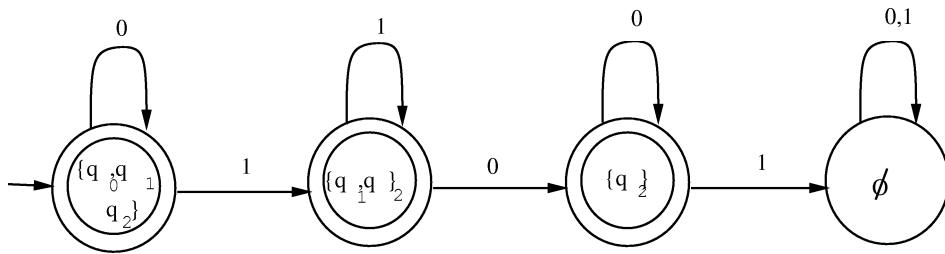


Figura 2.20: AFD equivalente ao AFN da figura 2.19.

## 2.6 Algoritmo da Minimização de Estados

Dizemos que um estado num AFD é acessível se existe um caminho no diagrama de transição de estados do estado inicial a ele. Mais formalmente, um estado  $q$  num AFD é **acessível** se existe  $w \in \Sigma^*$  tal que  $\delta^*(q_0, w) = q$ .

Claramente, sempre é possível eliminar estados não acessíveis de um autômato finito sem alterar a linguagem aceita pelo autômato.

Dizemos que dois estados  $q_i$  e  $q_j$  são do **mesmo tipo** num AFD se ambos são finais ou ambos são não finais.

Dois estados,  $q_i$  e  $q_j$ , num AFD  $M$  são **equivalentes**, denotado por  $q_i \equiv q_j$ , se para qualquer que seja  $w \in \Sigma^*$ ,  $\delta^*(q_i, w)$  e  $\delta^*(q_j, w)$  forem do mesmo tipo. Disto se desprende que dois estados que não são do mesmo tipo também não são equivalentes. Porém podem existir estados do mesmo tipo que não são equivalentes.

A seguir descreveremos um algoritmo para determinar estados equivalentes num AFD sem estados inacessíveis. Primeiro escreva uma tabela (matriz) com  $|Q|$  linhas e colunas e associe a cada linha e coluna um estado diferente. Assim, cada posição na tabela terá associada um par de estados. Cada posição da tabela será marcada com  $\times$ , caso os estados correspondentes sejam equivalentes e em branco caso contrário. O preenchimento dessa tabela seguirá os seguintes passos:

Passo 1: Marque com  $\times$  cada posição relativa a estados de diferentes tipos

$q_1$	$\times$				
$q_2$		$\times$			
$q_3$	$\times$		$\times$		
$q_4$		$\times$		$\times$	
$q_5$		$\times$		$\times$	
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$

Tabela 2.2: Estados que não são do mesmo tipo no AFD da Figura 2.21.

Passo 2: Marque com  $\times$  cada posição correspondente a um par de estados  $(q, q')$  para os quais existe  $a \in \Sigma$  tal que a posição  $(\delta(q, a), \delta(q', a))$  na tabela estiver marcada com  $\times$ . Aplique este passo até não ser possível marcar uma nova posição.

Como  $\equiv$  é de fato uma **relação de equivalência**, ou seja é reflexiva, simétrica e transitiva, podemos omitir da tabela todas as posições acima da diagonal assim como a própria diagonal e portanto também a primeira fila e ultima coluna.

**Exemplo 2.6.1** Seja o AFD ilustrado na figura 2.21. Aplicando o passo 1 do algoritmo acima obtemos a tabela 2.2

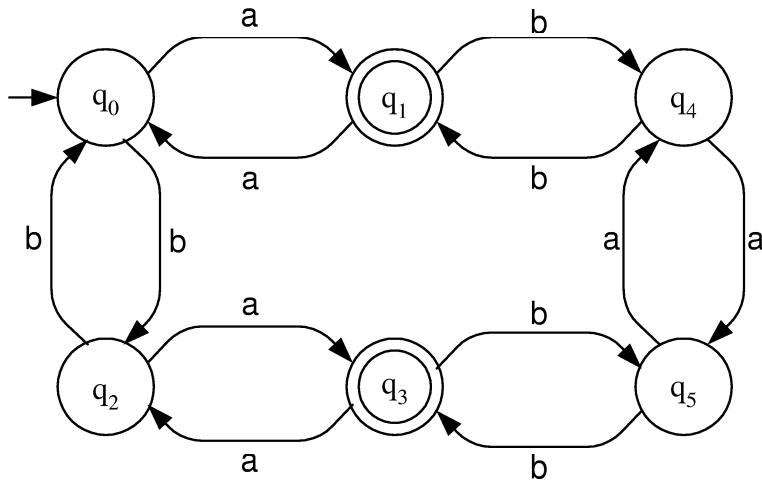


Figura 2.21: AFD com estados equivalentes.

Agora aplicando o passo 2:

- como  $\delta(q_0, a) = q_1$ ,  $\delta(q_4, a) = q_5$ , e  $(q_1, q_5)$  está marcado, então colocamos uma marca na posição  $(q_0, q_4)$ .
- como  $\delta(q_0, a) = q_1$ ,  $\delta(q_5, a) = q_4$ , e  $(q_1, q_4)$  está marcado, então colocamos uma marca na posição  $(q_0, q_5)$ .

## 2.6. Algoritmo da Minimização de Estados

---

$q_1$	$\times$				
$q_2$		$\times$			
$q_3$	$\times$		$\times$		
$q_4$	$\times$	$\times$	$\times$	$\times$	
$q_5$	$\times$	$\times$	$\times$	$\times$	
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$

Tabela 2.3: Estados que são equivalentes no AFD da Figura 2.21.

- como  $\delta(q_2, a) = q_3$ ,  $\delta(q_4, a) = q_5$ , e  $(q_3, q_5)$  está marcado, então colocamos uma marca na posição  $(q_2, q_4)$ .
- como  $\delta(q_2, a) = q_3$ ,  $\delta(q_5, a) = q_4$ , e  $(q_3, q_4)$  está marcado, então colocamos uma marca na posição  $(q_2, q_5)$ .

A tabela 2.3 apresenta o resultado de marcar as posições acima na tabela 2.2.

Observe que as outras posições não podem ser marcadas pois, por exemplo,  $\delta(q_0, a) = q_1$ ,  $\delta(q_2, a) = q_3$ , e  $(q_1, q_3)$  não está marcado e  $\delta(q_0, b) = q_2$ ,  $\delta(q_2, b) = q_0$ , e  $(q_0, q_2)$  não está marcado.

Assim, a tabela 2.3 apresenta os estados que são equivalentes no AFD da Figura 2.21.

Observe que  $q_i \equiv q_j$  se e somente se  $\{w \in \Sigma^* / \delta^*(q_i, w) \in F\} = \{w \in \Sigma^* / \delta^*(q_j, w) \in F\}$ . Assim dois estados equivalentes têm o mesmo comportamento de aceitação e por tanto são essencialmente os mesmos. Assim, como as classes de equivalência de  $\equiv$  são constituídas por estados com um mesmo comportamento, podemos reduzir o número de estados de um AFD unificando estados equivalentes.

Para cada  $q \in Q$  defina a **classe de equivalência** de  $q$  como sendo o conjunto  $[q] = \{q' \in Q / q \equiv q'\}$ . Observe que se  $q_i \equiv q_j$  então  $[q_i] = [q_j]$ . Denotemos por  $Q_{/\equiv}$  o conjunto das classes de equivalências de  $Q$ , ou seja  $Q_{/\equiv} = \{[q] / q \in Q\}$ .

No exemplo 2.6.1, da tabela 2.3 se desprende que  $q_0 \equiv q_2$ ,  $q_1 \equiv q_3$  e  $q_4 \equiv q_5$  (incluindo o fecho reflexivo, simétrico e transitivo destes pares). Assim, temos as seguintes classes de equivalências:

- $[q_0] = \{q_0, q_2\}$
- $[q_1] = \{q_1, q_3\}$
- $[q_4] = \{q_4, q_5\}$

Por lado, seja  $F_{/\equiv} = \{[q] / q \in F\}$ . Claramente  $F_{/\equiv} \subseteq Q_{/\equiv}$  e pelo fato de equivalência implicar em ser do mesmo tipo, temos que se  $q \in F$  se e somente se  $[q] \subseteq F$ .

Assim, a partir de um AFD  $M$  podemos construir o novo AFD  $M_{/\equiv} = \langle Q_{/\equiv}, \Sigma, [q_0], \delta_{/\equiv}, F_{/\equiv} \rangle$  onde  $\delta_{/\equiv} : Q_{/\equiv} \times \Sigma \rightarrow Q_{/\equiv}$ , definido por:

$$\delta_{/\equiv}([q], a) = [\delta(q, a)] \quad (2.6)$$

Observe que se  $q_i \equiv q_j$  então  $\delta_{/\equiv}([q_i], a) = \delta_{/\equiv}([q_j], a)$ . Noutras palavras não importa qual representante da classe for escolhido. Assim, no caso do exemplo 2.6.1, temos que

- $\delta_{/\equiv}([q_0], a) = [\delta(q_0, a)] = [q_1]$
- $\delta_{/\equiv}([q_0], b) = [\delta(q_0, b)] = [q_2] = [q_0]$
- $\delta_{/\equiv}([q_1], a) = [\delta(q_1, a)] = [q_0]$
- $\delta_{/\equiv}([q_1], b) = [\delta(q_1, b)] = [q_4]$
- $\delta_{/\equiv}([q_4], a) = [\delta(q_4, a)] = [q_5] = [q_4]$
- $\delta_{/\equiv}([q_4], b) = [\delta(q_4, b)] = [q_1]$

Dessas transições obtemos o AFD da figura 2.22 o qual é o AFD mínimo do AFD da figura 2.21.

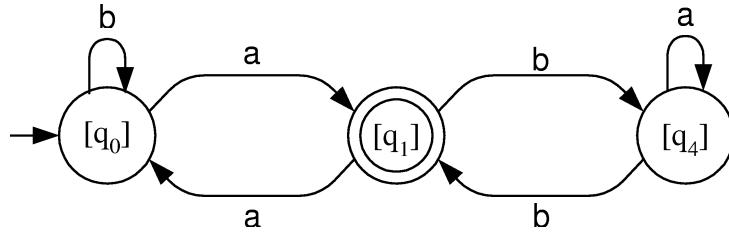


Figura 2.22: AFD  $M_{/\equiv}$  obtido a partir do AFD da Figura 2.21.

Dizemos que um AFD  $M$  é **mínimo**, se para qualquer outro AFD  $M'$  que reconheça a mesma linguagem, ou seja tal que  $L(M) = L(M')$ , temos que  $|Q| \leq |Q'|$ .

**Teorema 2.6.2** *Seja  $M$  um AFD sem estados inacessíveis. Então  $M_{/\equiv}$  é um AFD mínimo tal que  $L(M) = L(M_{/\equiv})$ .*

**DEMONSTRAÇÃO:** Da equação (2.6) e da definição de função de transição estendida (final da seção 2.1), resulta fácil mostrar que

$$\delta_{/\equiv}^*([q], w) = [\delta^*(q, w)]$$

e por tanto  $L(M) = \{w \in \Sigma^* / \delta^*(q_0, w) \in F\} = \{w \in \Sigma^* / \delta^*([q_0], w) \in F_{/\equiv}\} = L(M_{/\equiv})$ .

A demonstração da minimalidade de  $M_{/\equiv}$  é bem mais complexa, mas pode ser encontrada em [HU79, Mar91] entre outros. ■

Sejam  $M = \langle Q, \Sigma, q_0, \delta, F \rangle$  e  $M' = \langle Q', \Sigma, q'_0, \delta', F' \rangle$  dois AFD. Uma função bijetiva  $\varphi : Q \rightarrow Q'$  é um **isomorfismo** entre  $M$  e  $M'$  se

## 2.6. Algoritmo da Minimização de Estados

---

- $\varphi(q_0) = q'_0$ ;
- $F' = \{\varphi(q) / q \in F\}$ ;
- para cada  $q \in Q$  e  $a \in \Sigma$ ,  $\varphi(\delta(q, a)) = \delta'(\varphi(q), a)$ .

Neste caso, dizemos que  $M$  e  $M'$  são isomorfos e o denotamos por  $M \cong M'$ .

Claramente  $\cong$  é uma relação de equivalência para o conjunto de todos os AFD sobre um alfabeto  $\Sigma$ .

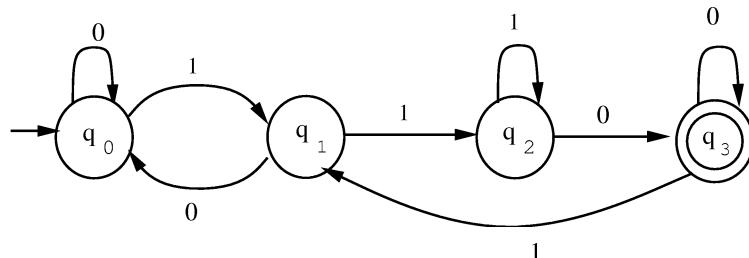
Assim, dois AFD são isomorfos se eles são similares no sentido que só muda o nome dado aos estados. Note que trivialmente se  $M \cong M'$  então  $L(M) = L(M')$ , mas o contrário não é verdade, ou seja dois AFD que reconhecem uma mesma linguagem não necessariamente são isomorfos.

**Proposição 2.6.3** *Se  $M$  e  $M'$  são mínimos e reconhecem a mesma linguagem então  $M \cong M'$ .*

DEMONSTRAÇÃO: Veja [HU79, Theorem 3.10]. ■

## 2.7 Exercícios

1. Construir um autômato para reconhecer somente a seqüência 10110.
  2. Quais das seguintes cadeias 0001, 010011, 0000110 são reconhecidas pelo autômato abaixo.



3. Construir um autômato finito determinístico que reconheça qualquer cadeia contendo um número qualquer de cópias de 001, seguido por 1 e nenhuma outra cadeia. Isto é o AFD deve reconhecer a linguagem:

$$\{w1 \in \{0,1\}^* \mid w = (001)^n \text{ para algum } n > 0\}$$

As cadeias reconhecidas por este autômato são do tipo: 1, 0011, 0010011, 0010010011, etc.

4. Para cada um dos seguintes casos, construir um AFD que com os símbolos de entrada  $a, b, c, d$  reconheça a descrição das cadeias dadas e não outras.

- (a) Qualquer cadeia consistindo somente de  $a$ 's.
  - (b) Qualquer cadeia terminando com um  $a$ .
  - (c) Qualquer cadeia consistindo de um número par de cópias de  $abb$ .
  - (d) Qualquer cadeia em  $\{a, bcd\}^*$ . Isto é a linguagem

$$\{\lambda, a, bcd, aa, abcd, bcdbcd, bcda, aaa, aabcd, \dots\}.$$

- (e) Qualquer cadeia contendo exatamente uma única cópia da cadeia  $abbb$ .  
 (f) Qualquer cadeia com uma quantidade par de  $a$ 's e ímpar de  $b$ 's e de  $c$ 's.

5. Desenhe AFD's que reconheçam os subconjuntos de  $\{a, b\}^*$  consistindo de

- (a) Todas as cadeias com exatamente um  $a$ .
  - (b) Todas as cadeias com no mínimo um  $a$ .
  - (c) Todas as cadeias com mais do que três  $a$ 's.
  - (d) todas as cadeias da forma  $ab^5wb^4$ , onde  $w \in \{a, b\}^*$

6. Dar uma AFD que aceite os números, no sistema binário, que:

## 2.7. Exercícios

---

- (a) São múltiplos de 4.  
 (b) São múltiplos de 3.  
 (c) São múltiplos de 5.  
 (d) São múltiplos de 6.
7. Encontre AFD's para as seguintes linguagens sobre  $\Sigma = \{a, b\}$ .
- $\mathcal{L} = \{w \in \Sigma^* / |w| \bmod 3 = 0\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / |w| \bmod 5 \neq 0\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \bmod 3 > 1\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) = 2 \text{ e } \mathcal{N}_b(w) \text{ é par}\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / 2 \leq \mathcal{N}_a(w) \leq 4\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \text{ é ímpar e, } \mathcal{N}_a(w) = 1 \text{ ou } \mathcal{N}_a(w) = 2\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \notin \{1, 2, 3\}\}$ ,
  - $\mathcal{L} = \{a^m b^n / m \text{ ou } n \text{ não é divisível por } 3\}$ ,
  - $\mathcal{L} = \{a^m b^n / mn \text{ é ímpar}\}$ ,
  - $\mathcal{L} = \{aa\bar{w}aa / w \in \Sigma^*\} \cup \{ab\bar{w}ba / w \in \Sigma^*\}$ ,
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \text{ é ímpar e } |w| \text{ é par}\}$ ,
- onde  $\mathcal{N}_a(w)$  é o número de  $a$ 's que ocorrem em  $w$  e  $|w|$  é o tamanho da cadeia  $w$ .
8. Desenhe um AFD que reconheça as seguinte linguagens sobre o alfabeto  $\Sigma = \{a, b, c\}$ :
- $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \text{ e } \mathcal{N}_c(w) \text{ é par}\}$
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) + \mathcal{N}_b(w) \text{ é ímpar e } \mathcal{N}_c(w) \text{ é par}\}$
  - $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \text{ é par, } \mathcal{N}_b(w) \text{ é ímpar e } \mathcal{N}_c(w) \text{ é par}\}$
9. Considere o conjunto das cadeias, sobre  $\{0, 1\}$ , definido pela condição abaixo. Exiba um AFD para cada um desses conjuntos.
- O primeiro e último símbolo sejam iguais.
  - O símbolo mais à esquerda difere do mais à direita.
  - Que reconheça a linguagem  $\mathcal{L} = \{vwv / v, w \in \{a, b\}^*\text{ e }|v| = 2\}$ .
  - Os dois primeiros símbolos sejam iguais aos dois últimos símbolos.
  - Todo 00 é seguido imediatamente por 1. Por exemplo, as cadeias  $\lambda$ , 101, 0010, 0010111001 estão na linguagem, mas 0001 e 00100, não estão.
  - Contenha pelo menos uma ocorrência da subcadeia 11010 e seus dois últimos símbolos sejam 1.
  - Toda subcadeia de quatro símbolos tem, quando muito, dois zeros. Por exemplo, 0011100, 001101 e 1010101 estão na linguagem, mas 10100110 não está, pois uma de suas subcadeias de quatro símbolos, 0100, contém três zeros.

- (h) Que não contenha qualquer ocorrência da subcadeia 000 ou 111.
- (i) Qualquer ocorrência de dois zeros próximos (isto é sem qualquer 0 no meio), esteja separado por uma quantidade ímpar de uns. Por exemplo,  $\lambda, 111, 11011, 011101011$  e  $110101110101$  fazem parte desta linguagem, mas as cadeias 0010 e 111010110 não.
- (j) O número de ocorrências de dois zeros consecutivos seja par. Observe que 000 é considerado como duas ocorrências de dois zeros consecutivos (o primeiro com o segundo e o segundo com terceiro). Assim, as cadeias  $\lambda, 111, 101011, 11100100101, 0000, 10000110100$  e  $01010010011$  fazem parte da linguagem, mas as cadeias 00101 e 0000 não.
10. Mostre que a linguagem  $\mathcal{L} = \{vwv/v, w \in \{a,b\}^*\text{ e }|v|=2\}$  é regular.
11. Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD. Mostre que  $\delta^*(q, vw) = \delta^*(\delta^*(q, v), w)$  para todo  $v, w \in \Sigma^*$ .
12. Na figura 2.12, ache  $\lambda\text{-fecho}(q_1)$ .
13. Para o AFN na figura 2.9, ache  $\delta^*(q_1, 100)$  e  $\delta^*(q_0, 00101)$ .
14. Na figura 2.15, ache  $\delta^*(q_0, 00)$ .
15. Seja  $\mathcal{L}$  a linguagem aceita pelo autômato da figura 2.2. Achar um AFD que aceite  $\mathcal{L}^2$ .
16. Construa um AFN com três estados que aceite a linguagem  $\{ab, abc\}^*$ .
17. Projete um AFN, com não mais que cinco estados, para o conjunto
- $$\{abab^n / n \geq 0\} \cup \{aba^n / n \geq 0\}$$
18. Achar um AFN, sem  $\lambda$ -transições, com um único estado final que aceita o conjunto  $\{a\} \cup \{b^n / n \geq 1\}$ .
19. Construa um AFN que aceite a linguagem de todas as cadeias sobre o alfabeto  $\{0, 1\}$  com um número par de 1's ou um número ímpar de 0's.
20. Considere o autômato da figura 2.23.
- (a) Quais das cadeias 00, 0100010, 10010, 000, 0000 são aceitas por ele?
- (b) De uma descrição “simples” da linguagem aceita por ele.

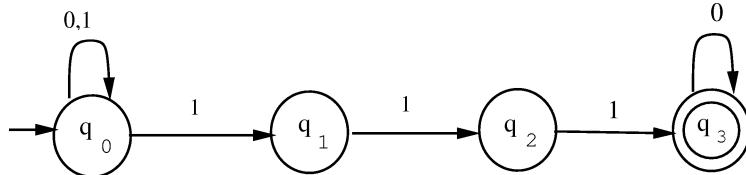


Figura 2.23: AFN do exercício 20

## 2.7. Exercícios

---

21. Use a construção do teorema 2.5.3 para converter os AFN's das figuras 2.11, 2.15, 2.24, 2.25, 2.26, 2.27 e 2.28, e do exercício 19 para AFD's equivalentes. É possível respostas mais simples se fizermos os AFD's diretamente?

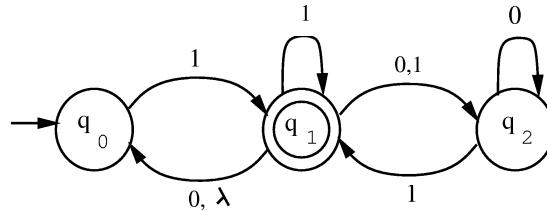


Figura 2.24: AFN do exercício 21

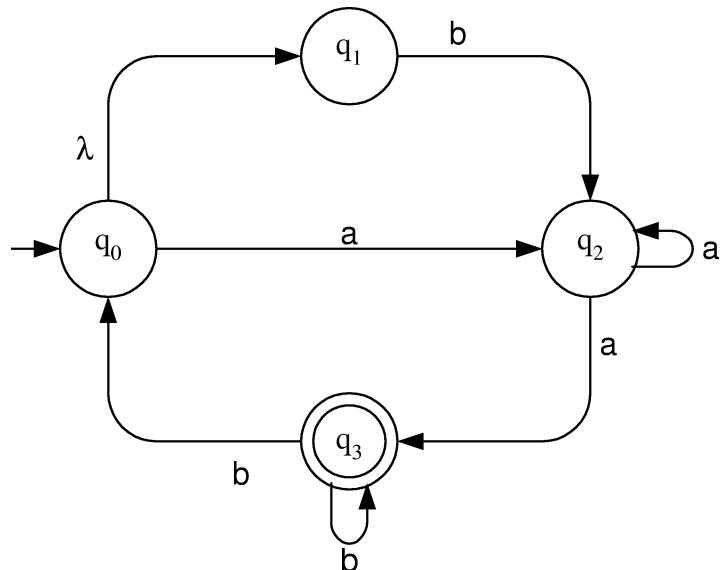


Figura 2.25: AFN do exercício 21

22. Achar um AFN, sem  $\lambda$ -transições, com um único estado final que aceita o conjunto  $\{a\} \cup \{b^n/n \geq 1\}$ .
23. É verdade que para qualquer AFN  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  o complemento de  $L(M)$  é igual ao conjunto  $\{w \in \Sigma^*/ \delta^*(q_0, w) \cap F = \emptyset\}$ ? Prove sua resposta.
24. Mostre que se  $\mathcal{L}$  é reconhecida por um autômato finito (determinístico ou não), então  $\mathcal{L}^R$  e  $\overline{\mathcal{L}}$  também são reconhecidos por algum autômato finito.
25. Seja  $\Sigma = \{a, b\}$ . Construa um AFN, que não seja AFD, e depois transforme ele num AFD usando o algoritmo do teorema 2.5.3, para as seguintes linguagens:

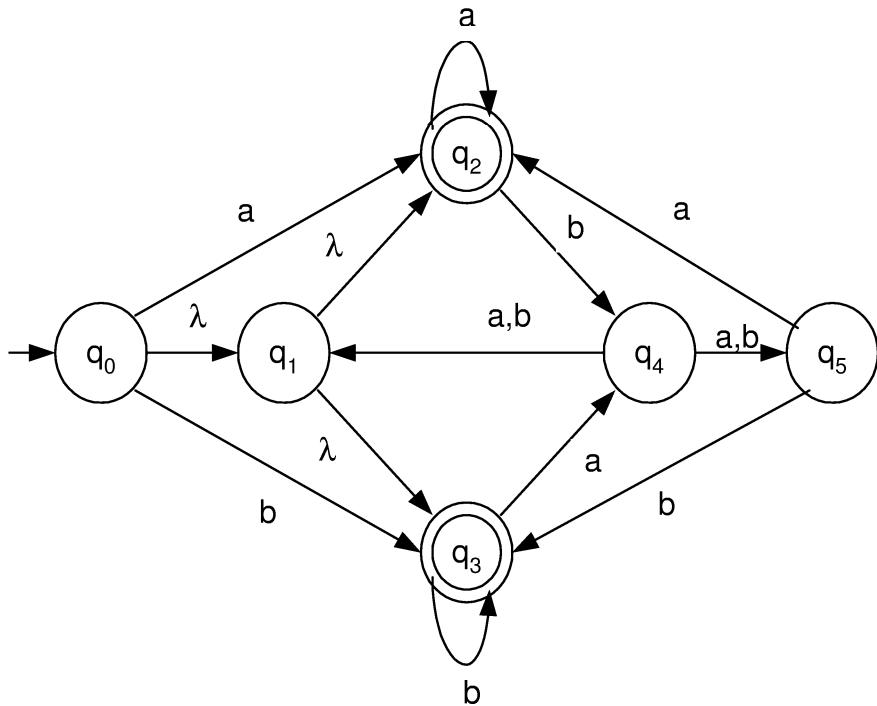


Figura 2.26: AFN do exercício 21

- (a)  $\mathcal{L} = \{w \in \Sigma^* / w = uaaav \text{ para alguma cadeia } u \text{ e } v \text{ tal que } N_b(v) \text{ é múltiplo de } 3\}$ ,
- (b)  $\mathcal{L} = \{w \in \{a,b\}^* / N_a(w) \text{ ou } N_b(w) \text{ é divisível por } 3\}$ ,
- (c)  $\mathcal{L} = \{w \in \Sigma^* / w = uaaav, \text{ para alguma cadeia } u, v \in \Sigma^*, \text{ e } N_a(u) \text{ é ímpar}\}$ ,
- (d)  $\mathcal{L} = \{w \in \Sigma^* / w = uaaav, \text{ para alguma cadeia } u, v \in \Sigma^*, \text{ e } N_b(v) \text{ é ímpar}\}$ ,
- (e)  $\mathcal{L} = \{uv \in \Sigma^* / N_a(u) = 3 \text{ e } N_b(v) \text{ é múltiplo de } 3\}$ ,
- (f)  $\mathcal{L} = \{uvw / u, w \in \Sigma^* \text{ e } v = a^3b^N a \text{ para algum número ímpar } n\}$ .
26. Renomeie os estados dos AFD resultantes dos exercícios 21 e 25 e aplique o algoritmo da minimização de estados.
27. Sejam os AFD das figuras A.31 e 2.30. Aplique o algoritmo da minimização de estados para achar os AFD mínimos equivalentes a eles.

## 2.7. Exercícios

---

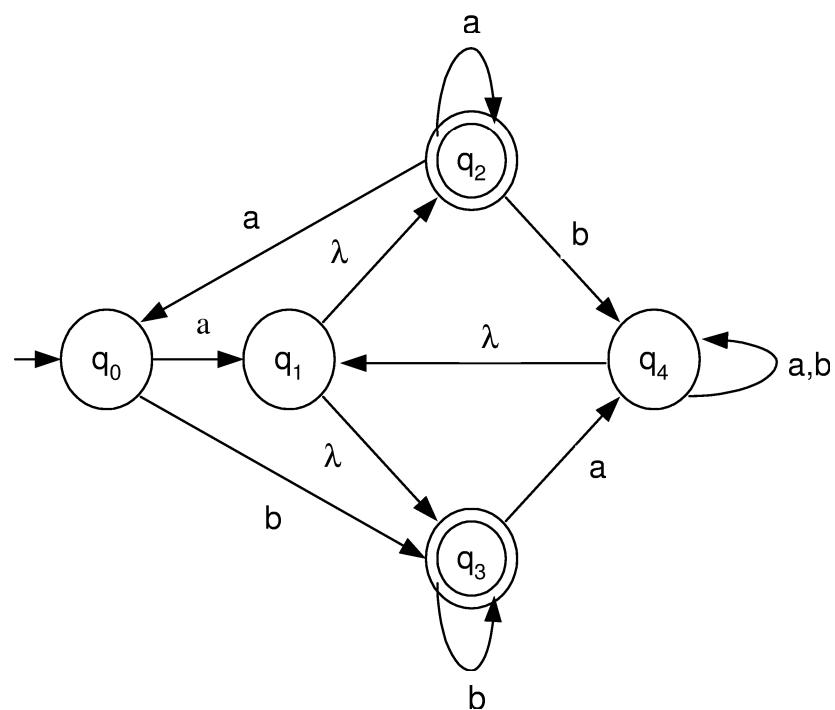


Figura 2.27: AFN do exercício 21

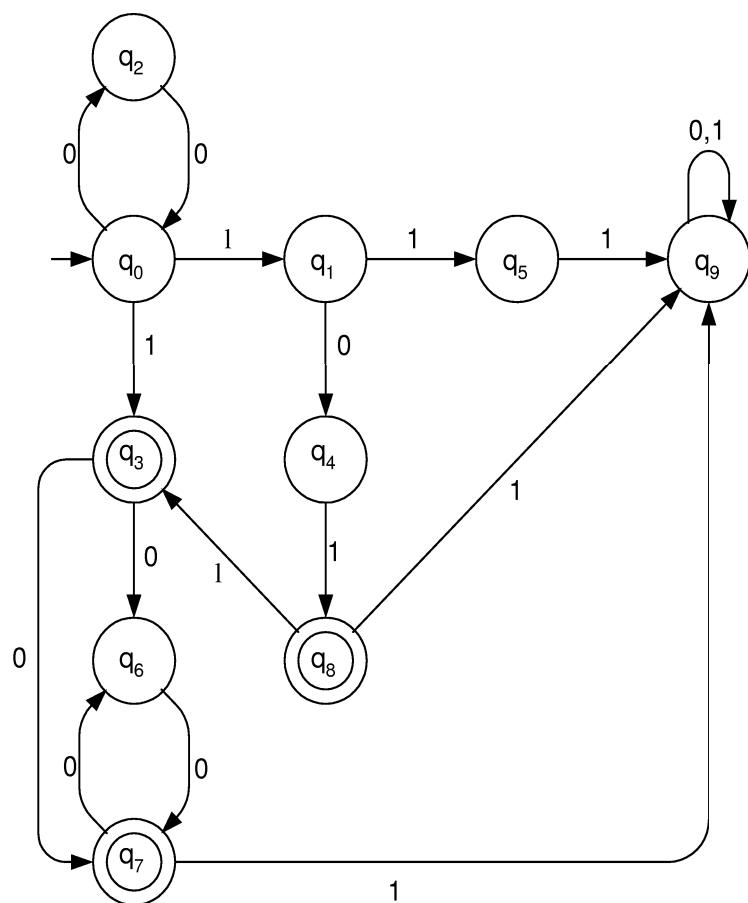


Figura 2.28: AFN do exercício 21

## 2.7. Exercícios

---

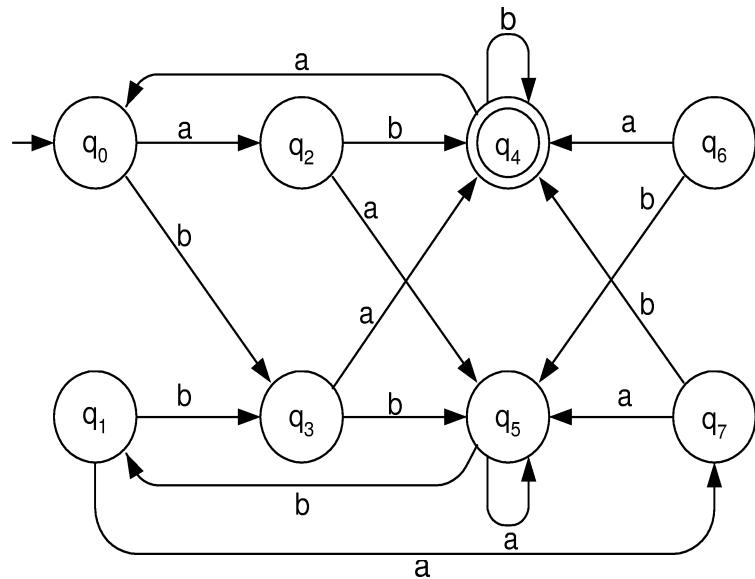


Figura 2.29: AFD não mínimo do exercício 27

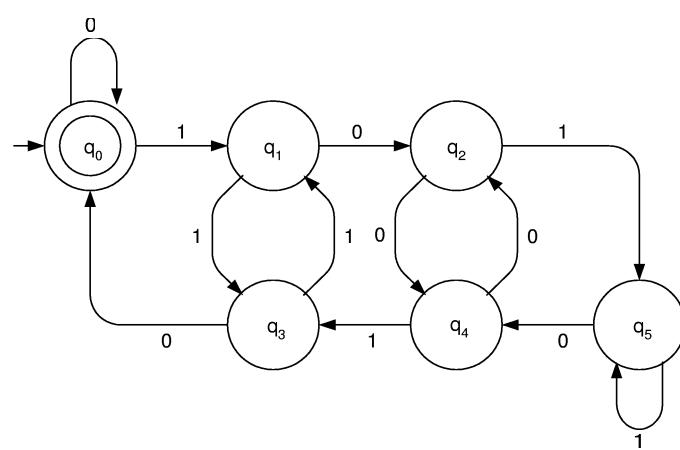


Figura 2.30: AFD não mínimo que reconhece os números em binário múltiplos de 6.

## Capítulo 3

# Expressões e Gramáticas Regulares

Segundo nossa definição, uma linguagem é regular se existe um autômato finito para ela. Assim, toda linguagem regular pode ser descrita por algum AFD ou algum AFN. Tal descrição pode ser muito útil, por exemplo, se quisermos mostrar a lógica por meio da qual decidimos se uma dada cadeia pertence a uma certa linguagem. Mas em muitos exemplos, precisamos de um modo mais simples de descrever linguagens regulares.

### 3.1 Expressões Regulares: Sintaxe

Um modo de descrever linguagens regulares é através da notação de *expressões regulares*. Essa notação envolve uma combinação de cadeias de símbolos do alfabeto da linguagem, parênteses e dos operadores  $+$ ,  $\cdot$  e  $*$ . O caso mais simples é a linguagem  $\{a\}$ , que será denotado pela expressão regular  $a$ .

Ligeiramente mais complicada é a linguagem  $\{a, b, c\}$ , na qual usaremos “ $+$ ” para denotar união, obtendo  $a + b + c$ , pois  $\{a, b, c\} = \{a\} \cup \{b\} \cup \{c\}$ . Usaremos “ $\cdot$ ” para concatenação e “ $*$ ” para o fecho-estrela. Assim, a expressão  $(a + b \cdot c)^*$  representa  $\{\lambda, a, bc, aa, abc, bcbc, aaa, \dots\}$ .

No que segue, daremos uma definição recursiva para *expressão regular*.

**Definição 3.1.1** Seja  $\Sigma$  um alfabeto. Então

1.  $\emptyset$ ,  $\lambda$  e  $a$ , para todo  $a \in \Sigma$ , são expressões regulares. Elas são chamadas **expressões regulares primitivas**.
2. Se  $r_1$  e  $r_2$  são expressões regulares, então  $(r_1 + r_2)$ ,  $r_1 \cdot r_2$  e  $(r_1)^*$  são expressões regulares.
3. Uma cadeia é uma **expressão regular** se, e somente se, ela pode ser derivada de expressões regulares primitivas, aplicando a regra 2., um número finito de vezes.

**Exemplo 3.1.2** Para  $\Sigma = \{a, b, c\}$ , a cadeia  $((a + b \cdot c))^* \cdot ((c + \emptyset) + aa)$  é uma expressão regular, pois ela é construída aplicando as regras acima. Por exemplo, se  $r_1 = c$  e  $r_2 = \emptyset$ , achamos  $(c + \emptyset)$ , que é uma expressão regular. Usando esse argumento várias vezes mostraremos que a expressão toda é regular. Por outro lado,  $(a + b^+)$  não é uma expressão regular.

### 3.2. Expressões Regulares: Semântica

---

Uma expressão regular  $r'$  é uma **sub-expressão regular** de uma expressão regular  $r$  se  $r = r'$  ou existe uma expressão regular  $r_1$  tal que  $r = (r_1)^*$  e  $r'$  é sub-expressão regular de  $r_1$  ou ainda existem expressões regulares  $r_1$  e  $r_2$  tal que  $r'$  é uma sub-expressão regular de  $r_1$  ou  $r_2$ , e  $r = (r_1 + r_2)$  ou  $r = r_1 \cdot r_2$ .

No exemplo 3.1.2, as expressões regulares  $a, b, c, (c + \emptyset)$  e  $(a + b \cdot c)$  são exemplos de sub-expressões regulares de  $((a + b \cdot c))^* \cdot ((c + \emptyset) + aa)$ .

## 3.2 Expressões Regulares: Semântica

Expressões regulares nos são úteis para descrevermos linguagens regulares. A conexão entre expressões regulares e linguagens é simples. Em geral, se  $r$  é uma expressão regular,  $L(r)$  representa a linguagem denotada por  $r$ . Ou seja, a semântica de uma expressão regular é uma linguagem.

**Definição 3.2.1** A linguagem  $L(r)$ , denotada por qualquer expressão regular  $r$ , é definida pelas seguintes regras:

1.  $L(\emptyset) = \emptyset$ ,
2.  $L(\lambda) = \{\lambda\}$ ,
3.  $L(a) = \{a\}$ , para todo  $a \in \Sigma$ ,
4. Se  $r_1$  e  $r_2$  são expressões regulares, então
  - (a)  $L((r_1 + r_2)) = L(r_1) \cup L(r_2)$ ,
  - (b)  $L(r_1 \cdot r_2) = L(r_1)L(r_2)$ ,
  - (c)  $L((r_1)^*) = (L(r_1))^*$ ,

As duas primeiras regras, mostram que os símbolos  $\emptyset$  e  $\lambda$  têm significados completamente diferentes, enquanto o primeiro denota a linguagem vazia, o segundo denota a linguagem contendo uma cadeia: a cadeia vazia.

As três últimas regras (4.(a) até 4.(c)), dessa definição, são usadas para obter  $L(r)$ , recursivamente, a partir das componentes mais simples. As três primeiras são as condições de término para essa recursão.

A linguagem  $L(r)$  assim obtida pode ser vista como a **semântica denotacional** da expressão regular  $r$ .

**Exemplo 3.2.2** Exibir a linguagem  $L(((a)^* \cdot (a + b) + \lambda))$ , em notação de conjunto.

$$\begin{aligned}
 L(((a)^* \cdot (a + b) + \lambda)) &= L((a)^* \cdot (a + b)) \cup L(\lambda) \\
 &= L((a)^*)L((a + b)) \cup \{\lambda\} \\
 &= (L(a))^*(L(a) \cup L(b)) \cup \{\lambda\} \\
 &= \{a\}^*(\{a\} \cup \{b\}) \cup \{\lambda\} \\
 &= \{\lambda, a, aa, aaa, \dots\} \{a, b\} \cup \{\lambda\} \\
 &= \{a, b, aa, ab, aaa, aab, aaaa, aaab, \dots\} \cup \{\lambda\} \\
 &= \{\lambda, a, b, aa, ab, aaa, aab, aaaa, aaab, \dots\}
 \end{aligned}$$

Lamentavelmente, expressões regulares podem ter mais parênteses que o necessário. Por exemplo, no caso da expressão regular do exemplo 3.1.2, os dois parênteses consecutivos poderiam ser reduzidos a um único sem levar a qualquer tipo de confusão. Assim, a seguir descreveremos algumas regras simples de simplificação de parênteses para as expressões regulares. Observe que isto não modifica a linguagem só diz que é uma abreviação, ou seja que embora ela não seja a rigor uma expressão regular (veja o caso de  $(a + b \cdot c)^*$ ) ela representa de forma unívoca (a menos de equivalência semântica) uma expressão regular válida.

1. Quando  $r_1$  for um expressão regular primitiva em  $(r_1)^*$  vamos simplificar a notação e escrever  $r_1^*$ .
2. Quando  $r_1$  em  $(r_1)^*$  for da forma  $(r + s)$ , escreveremos  $(r + s)^*$  em vez de  $((r + s))^*$ .
3. Quando a expressão regular for  $(r_1 + r_2)$  descartaremos os parênteses mais externos.
4. Quando tivermos uma sub-expressão regular do tipo  $(r_1 + (r_2 + r_3))$  ou do tipo  $((r_1 + r_2) + r_3)$  abreviaremos para  $(r_1 + r_2 + r_3)$

Assim, por exemplo, a expressão regular dos exemplos 3.1.2 e 3.2.2 seriam abreviadas para  $(a + b \cdot c)^* \cdot (c + \emptyset + aa)$  e  $a^* \cdot (a + b) + \lambda$ , respectivamente.

Para simplificar ainda mais a notação, daqui em diante, o “.” da concatenação também será omitido.

A seguir alguns exemplos de expressões regulares e as linguagens que elas denotam.

**Exemplo 3.2.3** Para  $\Sigma = \{a, b\}$ , a expressão  $r = (a + b)^*(a + bb)$  é regular. Ela denota a linguagem

$$L(r) = \{a, bb, aa, abb, ba, bbb, aaa, aabb, aba, abbb, bba, bbbb, \dots\}$$

Podemos ver isso considerando as várias partes de  $r$ . A primeira parte,  $(a + b)^*$ , representa qualquer cadeia de  $a$ 's e  $b$ 's. A segunda parte,  $(a + bb)$ , representa ou um  $a$  ou duas cópias de  $b$ . Consequentemente,  $L(r)$  é o conjunto de todas as cadeias sobre  $\{a, b\}$ , que terminam em  $a$  ou em  $bb$ . Podemos, também, obter este resultado usando as regras da definição 3.2.1, junto com as regras de precedência descritas anteriormente, como segue

### 3.2. Expressões Regulares: Semântica

---

$$\begin{aligned}
L(r) &= L((a+b)^*)L(a+bb) \\
&= (L((a+b)))^*(L(a) \cup L(bb)) \\
&= (L(a+b))^*(\{a\} \cup (L(b)L(b))) \\
&= (L(a) \cup L(b))^*(\{a\} \cup (\{b\}\{b\})) \\
&= (\{a\} \cup \{b\})^*\{a, bb\} \\
&= \{a, b\}^*\{a, bb\} \\
&= \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\}\{a, bb\} \\
&= \{a, bb, aa, abb, ba, bbb, aaa, aabb, aba, abbb, bba, bbbb, \dots\}
\end{aligned}$$

**Exemplo 3.2.4** A expressão

$$r = (aa)^*(bb)^*b$$

denota o conjunto de todas as cadeias com um número par de a's (0 é considerado par), seguido por um número ímpar de b's, isto é,

$$L(r) = \{a^{2n}b^{2m+1} / n \geq 0, m \geq 0\}$$

**Exemplo 3.2.5** Para  $\Sigma = \{0, 1\}$ , escrever uma expressão regular r tal que

$$L(r) = \{w \in \Sigma^* / w \text{ tem pelo menos um par de zeros consecutivos}\}.$$

Usaremos o seguinte raciocínio: toda cadeia em  $L(r)$  deve conter 00 em algum lugar, mas o que vem antes e o que vem depois é completamente arbitrário. Uma cadeia arbitrária, sobre  $\{0, 1\}$  pode ser denotada por  $(0+1)^*$ . Colocando juntas essas observações, chegamos à solução

$$r = (0+1)^*00(0+1)^*.$$

**Exemplo 3.2.6** Achar uma expressão regular para a linguagem

$$\mathcal{L} = \{w \in \{0, 1\}^* / w \text{ não tem qualquer par de zeros consecutivos}\}.$$

Embora esse exemplo pareça com o anterior, ele é muito mais difícil de se construir. Uma observação útil é que se um 0 ocorre ele deve ser imediatamente seguido de um 1 (a menos que seja o último da cadeia). Tal subcadeia pode ser seguida ou precedida por um número arbitrário de 1's. Isto sugere que a resposta envolve a repetição de cadeias da forma  $1^n01^m$ , com  $n \geq 0$  e  $m \geq 1$ , ou seja, a linguagem denotada pela expressão regular  $1^*(011^*)^*$ . Entretanto, a resposta ainda está incompleta, pois as cadeias terminando em 0 não foram consideradas. Após levar em conta isto chegamos à seguinte resposta

$$r = 1^*(011^*)^*(0 + \lambda).$$

Se raciocinarmos de modo ligeiramente diferente, podemos chegar a uma outra resposta. Se olharmos para  $\mathcal{L}$ , como a repetição das cadeias 1 e 01, a expressão mais curta

$$r = (1 + 01)^*(0 + \lambda),$$

poderá ser obtida. A primeira parte,  $(1 + 01)^*$ , garante que só obteremos cadeias sem dois 0's consecutivos. A segunda parte,  $(0 + \lambda)$ , garante que estas cadeias sem 0's consecutivos podem ou não terminar em 0. Uma resposta análoga, usando um raciocínio inverso seria

$$r = (0 + \lambda)(1 + 10)^*.$$

Embora a primeira expressão regular pareça diferente das duas últimas, ambas as respostas estão corretas, pois elas denotam a mesma linguagem. Em geral, existe um número ilimitado de expressões regulares para uma dada linguagem.

### 3.3 Equivalência entre Expressões Regulares

O exemplo anterior, introduz a noção de equivalência de expressões regulares. Dizemos que duas expressões regulares são **equivalentes** se elas denotam a mesma linguagem, isto é, as expressões regulares  $r_1$  e  $r_2$  são equivalentes, denotado por  $r_1 \equiv r_2$ , se  $L(r_1) = L(r_2)$ . Claramente esta noção de equivalência é uma relação de equivalência, isto é, é reflexiva, simétrica e transitiva.

**Exemplo 3.3.1** Assim, para mostrar que as expressões regulares  $(a + b)^*$  e  $(a^*b^*)^*$  são equivalentes, isto é que  $(a + b)^* \equiv (a^*b^*)^*$ , devemos mostrar que  $L((a + b)^*) = L((a^*b^*)^*)$ . Uma tal prova é dada a seguir.

$$\begin{aligned} L((a + b)^*) &= L((a + b))^* \\ &= (L(a) \cup L(b))^* \\ &= (\{a\} \cup \{b\})^* \\ &= \{a, b\}^* \\ &= \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\} \\ \\ L((a^*b^*)^*) &= L(a^*b^*)^* \\ &= (L(a^*)L(b^*))^* \\ &= (L(a)^*L(b)^*)^* \\ &= (\{a\}^*\{b\}^*)^* \\ &= (\{\lambda, a, aa, aaa, \dots\}\{\lambda, b, bb, bbb, \dots\})^* \\ &= \{\lambda, b, a, bb, ab, aa, bbb, abb, aab, aaa, \dots\}^* \\ &= \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\} \end{aligned}$$

Logo,  $L((a + b)^*) = L((a^*b^*)^*)$ , e, portanto,  $(a + b)^* \equiv (a^*b^*)^*$ .

**Proposição 3.3.2** Sejam  $r_1$ ,  $r_2$  e  $r_3$  três expressões regulares arbitrárias. Então

$$r_1(r_2 + r_3) \equiv r_1r_2 + r_1r_3.$$

### **3.4. Conexão entre Expressões Regulares e Linguagens Regulares**

---

DEMONSTRAÇÃO:

$$\begin{aligned} L(r_1(r_2 + r_3)) &= L(r_1)L(r_2 + r_3) \\ &= L(r_1)(L(r_2) \cup L(r_3)) \\ &= L(r_1)L(r_2) \cup L(r_1)L(r_3) \\ &= L(r_1r_2) \cup L(r_1r_3) \\ &= L(r_1r_2 + r_1r_3) \end{aligned}$$

Logo,  $r_1(r_2 + r_3) \equiv r_1r_2 + r_1r_3$ . ■

Poderíamos derivar várias regras para simplificar expressões regulares, mas certamente não teríamos um algoritmo (como temos para autômatos finitos determinísticos) que dada uma expressão regular arbitrária garanta obter sempre a menor expressão regular equivalente a ela. Para isso teríamos primeiro que estabelecer que significaria para nos que uma expressão regular é menor que uma outra expressão regular e depois descrever esse algoritmo. Parece razoável que um tal conceito de "menor" poderia ser obtido a partir da quantidade de símbolos usados na expressão, ou seja uma expressão regular  $r$  seria menor que uma expressão regular  $s$  se  $|r| \leq |s|$ , onde  $|r|$  é a quantidade de símbolos usados na expressão  $r$  (sem considerar parênteses). Por exemplo,  $(a+b)^*$  seria menor que  $(a^*b^*)^*$ , pois  $|(a+b)^*| = 4$  enquanto  $|(a^*b^*)^*| = 5$ . Mas, para definir um tal algoritmo deveríamos conseguir uma quantidade finita de reduções, isto é pares de expressões regulares genéricas equivalentes onde a segunda expressão seja menor que a primeira, capazes de que quando substituirmos exaustivamente as subexpressões de uma expressão regular arbitrária que sejam instanciadas no lado esquerdo de alguma das reduções pelo seu respectivo lado direito, obtenhamos sempre a menor expressão possível. Lamentavelmente isto não será possível.

## **3.4 Conexão entre Expressões Regulares e Linguagens Regulares**

Os conceitos de expressão regular e linguagem regular são, essencialmente, os mesmos, no sentido que para toda linguagem regular existe uma expressão regular que a denota e vice-versa. Mostraremos esse resultado em duas partes.

### **3.4.1 As Expressões Regulares Denotam Linguagens Regulares**

Mostremos primeiro que se  $r$  é uma expressão regular, então  $L(r)$  é uma linguagem regular. Nossa definição diz que uma linguagem é regular se ela é reconhecida por algum AFD. Como os AFD's e os AFN's são equivalentes, uma linguagem também é regular se é reconhecida por algum AFN. Mostraremos, agora, que se tivermos qualquer expressão regular  $r$ , podemos construir um AFN que reconhece  $L(r)$ . A construção desse AFN se baseia na definição recursiva de  $L(r)$ . Antes construiremos autômatos simples para as partes de 1. a 3. da definição 3.2.1, e, então mostraremos como eles são combinados para implementar as partes mais complicadas de 4.a. a 4.d.

**Teorema 3.4.1** *Seja  $r$  uma expressão regular. Então, existe algum AFN que reconhece  $L(r)$ . Conseqüentemente,  $L(r)$  é uma linguagem regular e portanto,  $r$  especifica uma linguagem regular.*

**DEMONSTRAÇÃO:** Como a definição de expressão regular é induutiva, demonstraremos este teorema por indução na complexidade da expressão regular  $r$ . A cada passo da indução sempre obteremos um AFN com um único estado final.

**Base da indução:** Começaremos com autômatos que reconhecem as linguagens para as expressões regulares primitivas  $\emptyset$ ,  $\lambda$  e  $a$ , para cada  $a \in \Sigma$ . Esses autômatos são mostrados nas figuras 3.1 (a), (b) e (c), respectivamente.

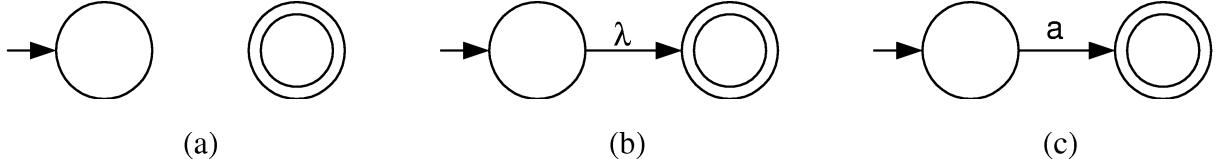


Figura 3.1: AFN's que aceitam as linguagens regulares  $\emptyset$ ,  $\{\lambda\}$  e  $\{a\}$ , respectivamente.

**Etapa indutiva:** Suponhamos que existem autômatos  $M(r_1)$  e  $M(r_2)$  que reconhecem as linguagens denotadas pelas expressões regulares  $r_1$  e  $r_2$ , respectivamente. Não precisamos construir esses autômatos explicitamente, mas podemos representá-los como na figura 3.2, isto é, como uma caixa preta contendo um estado de início e outro de saída. Observe que os autômatos das figuras 3.1.a, 3.1.b e 3.1.c, têm estas características (um único estado final) e as construções que realizaremos aqui (figuras 3.3, 3.4 e 3.5) preservam essas características.

Nesse esquema,  $M(r)$  é uma caixa preta da qual só sabemos que reconhece  $L(r)$ , cujo vértice à esquerda do grafo representa o estado inicial e o vértice à direita o estado final. Com  $M(r_1)$  e  $M(r_2)$  representados dessa maneira, podemos construir autômatos para as expressões regulares  $r_1 + r_2$ ,  $r_1 r_2$ ,  $r_1^*$  e  $(r_1)$ . As construções para as três primeiras expressões regulares são mostradas nas figuras 3.3 a 3.5, já a quarta expressão regular, isto é  $(r_1)$ , tem como autômato o próprio autômato de  $r_1$ .

Como está indicado no desenho, os estados inicial e final das máquinas constituintes perdem seus *status* e são trocados por novos estados inicial e final. Combinando várias etapas como essas, podemos construir autômatos para expressões regulares arbitrariamente complexas.

Observando os grafos, é possível concluir que esta construção funciona. ■

**Exemplo 3.4.2** Achar um AFN que reconhece  $L(r)$ , onde

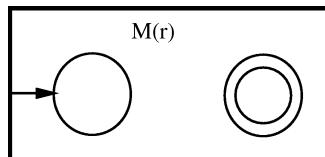


Figura 3.2: Representação esquemática de um AFN que reconhece  $L(r)$ .

### 3.4. Conexão entre Expressões Regulares e Linguagens Regulares

---

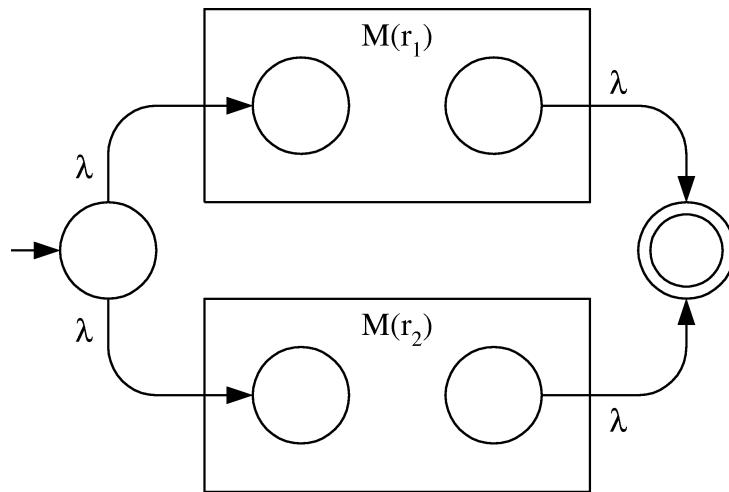


Figura 3.3: Autômato para  $L(r_1 + r_2)$ .

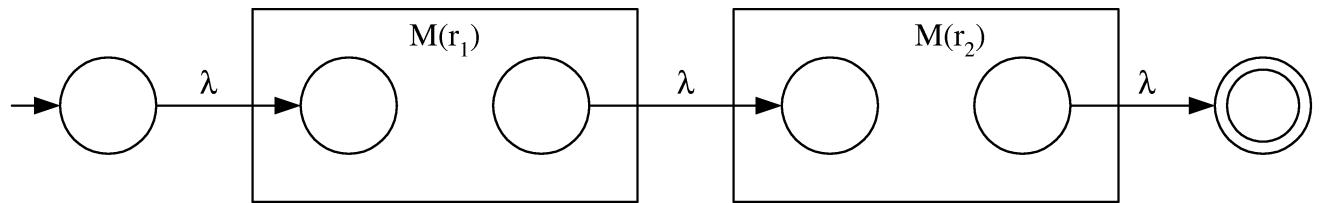


Figura 3.4: Autômato para  $L(r_1r_2)$ .

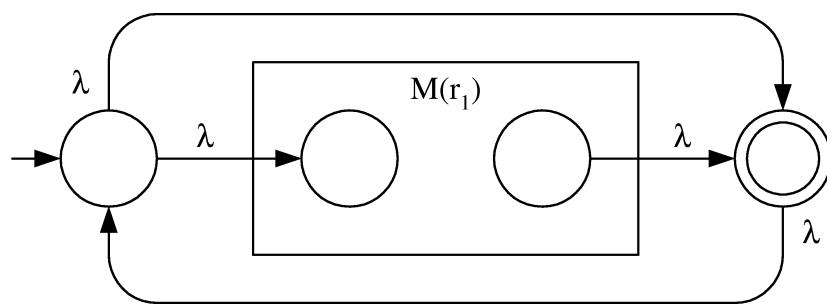


Figura 3.5: Autômato para  $L(r_1^*)$ .

$$r = (aa + bb)^*ba.$$

Para simplificar vamos fazer diretamente os autômatos para as expressões regulares mais simples ( $aa$ ,  $bb$  e  $(ba)$ ). As figuras 3.6 e 3.9 mostram esses casos simples. As figuras 3.7 a 3.8 e 3.10 ilustram o processo, descrito no teorema 3.4.1, para determinar o AFN desejado.

### 3.4.2 Linguagens Regulares para Expressões Regulares

É razoável esperar que a inversa do teorema 3.4.1 seja verdadeira e que, portanto, para toda linguagem regular exista uma expressão regular correspondente. Como qualquer linguagem regular tem associada um AFD e portanto um grafo de transição, tudo que precisamos fazer é achar uma expressão regular capaz de gerar os rótulos de todos os caminhos do estado inicial a qualquer estado final, isto não parece muito difícil, mas, certamente, é um pouco complicado. Para obter isso precisaremos de algum conhecimento intermediário, isto é, de quais são as expressões regulares que representam as cadeias que saem de um estado  $q$  (não necessariamente o inicial) e chegam a um estado  $q'$  (não necessariamente o final) sem passar por alguns estados.

**Teorema 3.4.3** Seja  $\mathcal{L}$  uma linguagem regular. Então existe uma expressão regular  $r$  tal que  $\mathcal{L} = L(r)$ .

**DEMONSTRAÇÃO:** Seja  $M$  um AFD que aceita  $\mathcal{L}$ . Facilmente podemos redefinir os estados de  $M$  de tal forma que  $Q = \{q_1, q_2, \dots, q_n\}$ , para algum natural  $n \geq 1$ , com  $q_1$  sendo o estado inicial.

Seja  $R_{i,j}^k$  a expressão regular denotando o conjunto de todas as cadeias em  $w \in \Sigma^*$  tal que  $\delta^*(q_i, w) = q_j$ , sem passar por um estado intermediário  $q_m$ , com  $m > k$ , onde  $q_m$  é um estado intermediário de  $\delta^*(q_i, w)$  se para algum prefixo  $u$  de  $w$ , tal que  $u \neq \lambda$  e  $u \neq w$ ,  $\delta^*(q_i, u) = q_m$ . Assim,  $i$  e  $j$  podem ser maiores que  $k$ .

Podemos definir  $R_{i,j}^k$ , recursivamente.

$$R_{i,j}^0 = \begin{cases} \sum_{\delta(q_i, a) = q_j} a & \text{se } i \neq j \\ \lambda + \sum_{\delta(q_i, a) = q_j} a & \text{se } i = j \end{cases}$$

onde o somatório é com respeito ao operador  $+$ , sobre as expressões regulares. Assim, por exemplo,  $\sum_{i=1}^3 a_i$  é a expressão regular  $a_1 + a_2 + a_3$ .

$$R_{i,j}^k = (R_{i,k}^{k-1}(R_{k,k}^{k-1})^*R_{k,j}^{k-1}) + R_{i,j}^{k-1}.$$

Da definição acima,  $R_{i,j}^k$  é uma expressão regular.

Para mostrar que a definição acima de  $R_{i,j}^k$  satisfaz a intuição de serem as cadeias que fazem o autômato  $M$  ir de  $q_i$  a  $q_j$ , sem passar por estados intermediários com índices superiores a  $k$ , devemos realizar uma análise minuciosa dessa definição. Objetivando dar uma evidência disso, note que  $R_{i,j}^0$  representa todos os símbolos em  $\Sigma$  para os quais há uma transição saindo do estado  $q_i$  para o estado  $q_j$  e, portanto, nunca passará por qualquer estado intermediário. Por outro lado, a

### 3.4. Conexão entre Expressões Regulares e Linguagens Regulares

---

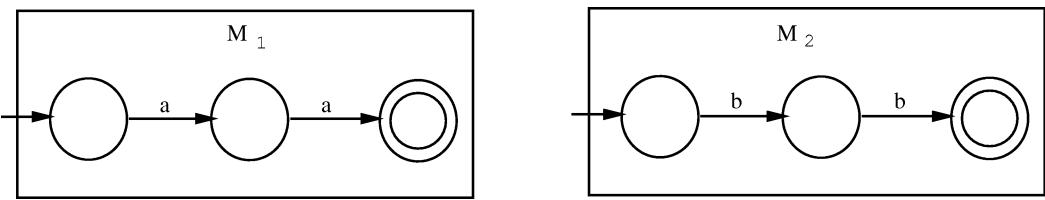


Figura 3.6:  $M_1$  reconhece  $L(aa)$  e  $M_2$  reconhece  $L(bb)$ .

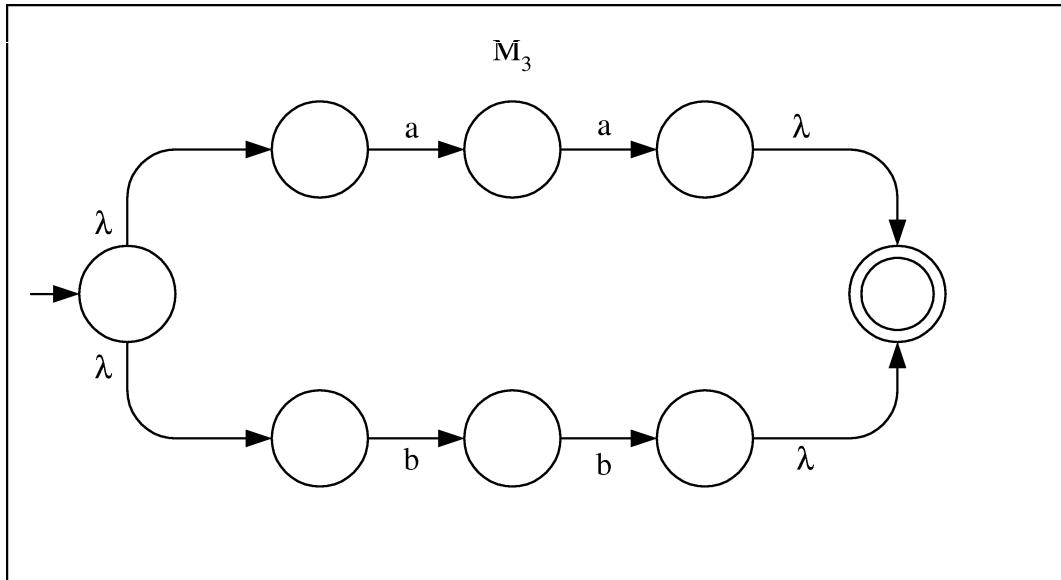


Figura 3.7:  $M_3$  reconhece  $L(aa + bb)$ .

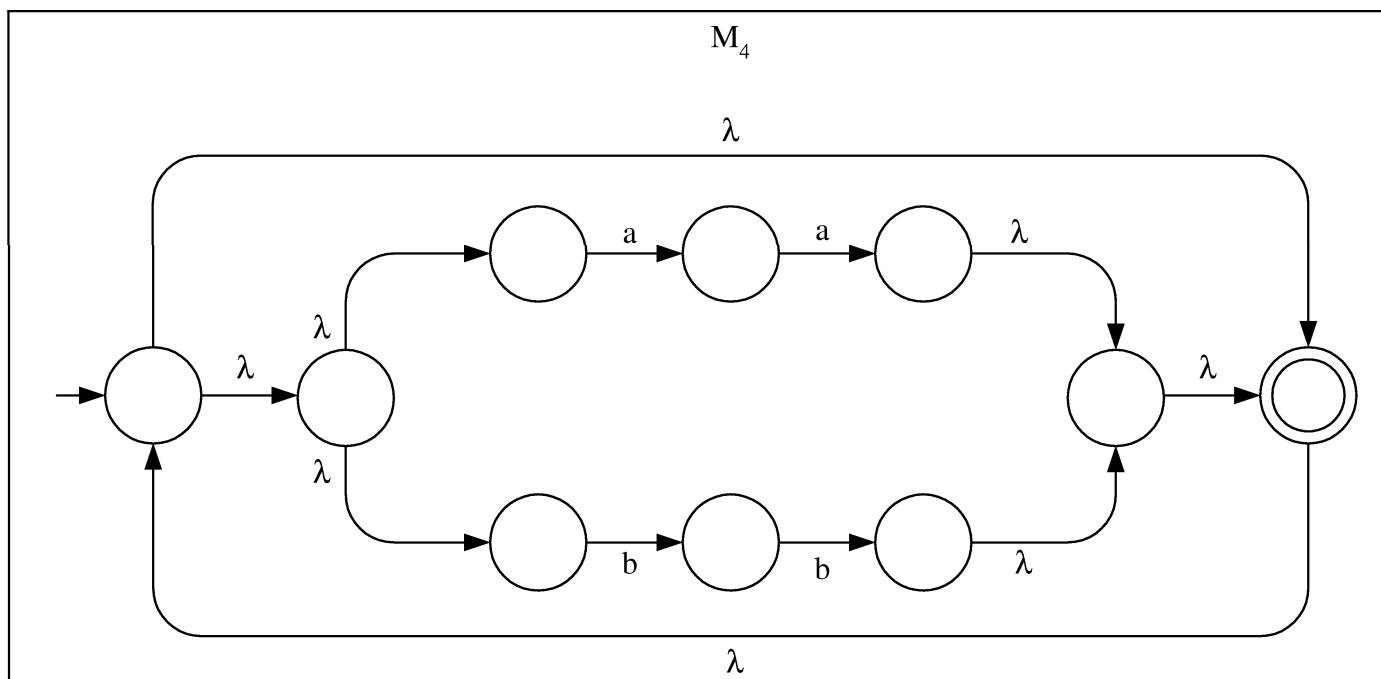
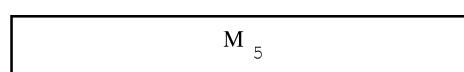


Figura 3.8:  $M_4$  reconhece  $L((aa + bb)^*)$ .



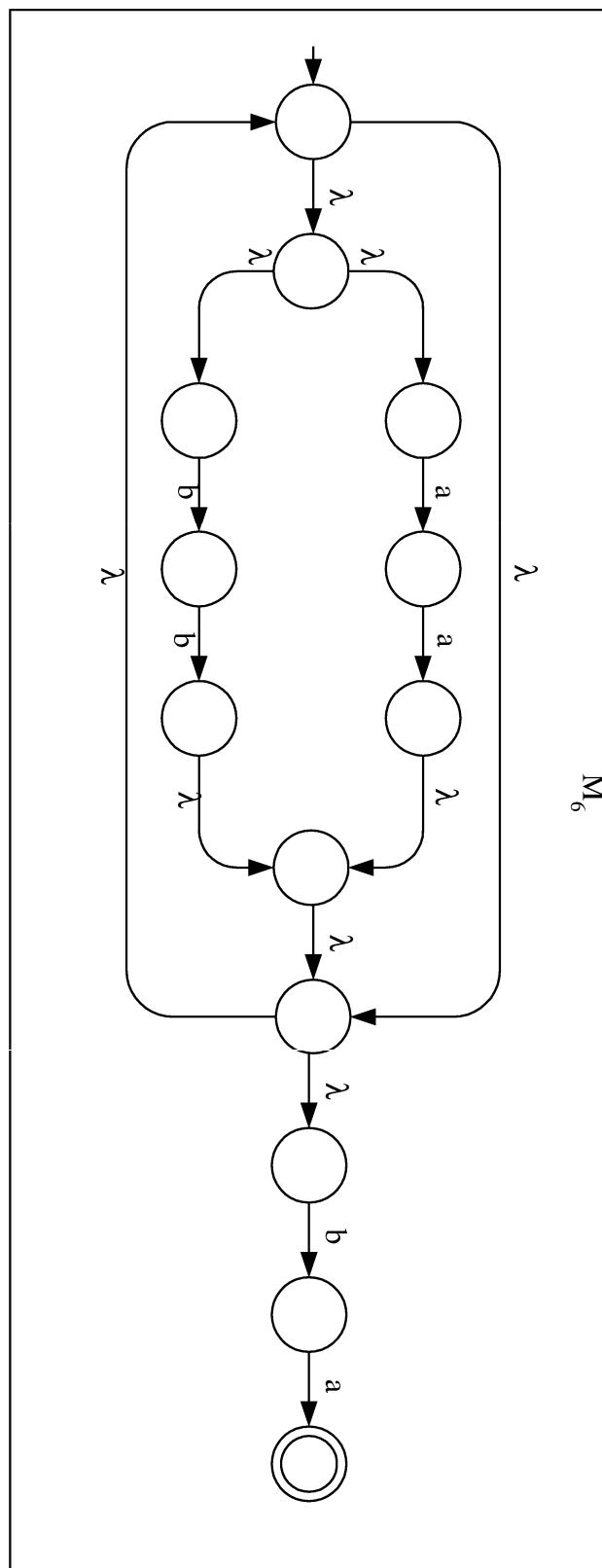


Figura 3.10:  $M_6$  reconhece  $L((aa + bb)^*ba)$ .

### 3.4. Conexão entre Expressões Regulares e Linguagens Regulares

---

definição de  $R_{i,j}^k$  esta composta só por  $R_{m,n}^{k-1}$ , com  $m$  e  $n$  sendo  $i, j$  ou  $k$ , portanto nunca passará por um estado com índice superior a  $k$ .

Logo,

$$r = \sum_{q_f \in F} R_{1,f}^n,$$

para  $n = |Q|$  (o número de estados do autômato) é a expressão regular que denota as cadeias em  $\Sigma^*$  que são aceitas por  $M$ . ■

Assim, como consequência destes dois últimos teoremas temos que a classe de linguagens descrita pelas expressões regulares é a classe das linguagens regulares. Portanto, as expressões regulares, AFD's e AFN's têm a mesma potencialidade, porém divergem na sua natureza. Enquanto as expressões regulares descrevem ou denotam linguagens regulares os AFD's e AFN's reconhecem tais linguagens.

**Exemplo 3.4.4** Seja  $M$  o autômato da figura 3.11. A seguir usaremos o “algoritmo” da prova do teorema anterior para determinar a expressão regular que descreve  $L(M)$ .

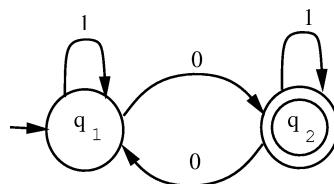


Figura 3.11: AFD para o exemplo 3.4.4.

Segundo esse teorema devemos achar  $R_{1,2}^2$ , mas, da definição de  $R_{i,j}^k$ , devemos antes achar  $R_{1,2}^1$  e  $R_{2,2}^1$ . Para achar estas expressões regulares devemos achar  $R_{1,1}^0$ ,  $R_{1,2}^0$ ,  $R_{2,1}^0$  e  $R_{2,2}^0$ . Assim sendo,

$$R_{1,1}^0 = \lambda + 1$$

$$R_{1,2}^0 = 0$$

$$R_{2,1}^0 = 0$$

$$R_{2,2}^0 = \lambda + 1$$

$$\begin{aligned} R_{1,2}^1 &= (R_{1,1}^0(R_{1,1}^0)^*R_{1,2}^0) + R_{1,2}^0 \\ &= ((\lambda + 1)(\lambda + 1)^*0) + 0 \\ &= 1^*0 \end{aligned}$$

$$\begin{aligned} R_{2,2}^1 &= (R_{2,1}^0(R_{1,1}^0)^*R_{1,2}^0) + R_{2,2}^0 \\ &= (0(\lambda + 1)^*0) + (\lambda + 1) \\ &= 01^*0 + \lambda + 1 \end{aligned}$$

$$\begin{aligned} R_{1,2}^2 &= (R_{1,2}^1(R_{2,2}^1)^*R_{2,2}^1) + R_{1,2}^1 \\ &= (1^*0(01^*0 + \lambda + 1)^*(01^*0 + \lambda + 1)) + (1^*0) \\ &= 1^*0(01^*0 + 1)^*(01^*0 + \lambda) + 1^*0 \\ &= 1^*0(01^*0 + 1)^* + 1^*0 \\ &= 1^*0(01^*0 + 1)^* \end{aligned}$$

Logo, a expressão regular que representa a linguagem regular reconhecida pelo autômato da figura 3.11 é

$$1^*0(01^*0 + 1)^*,$$

que representa a linguagem de todas as cadeias no alfabeto  $\{0, 1\}$ , com uma quantidade ímpar de 0's.

### 3.5 Gramáticas Regulares

Uma terceira maneira de descrever uma linguagem regular é através de gramáticas. Gramáticas são modos alternativos de especificar linguagens. Sempre que descrevemos uma família de linguagens através de um autômato ou de qualquer outra maneira, estaremos interessados em saber que espécie de gramática podemos associar a essa família. Pois, ela descreve como podem ser gerados os elementos da linguagem. Assim, a seguir veremos um tipo de gramáticas as quais, posteriormente, provaremos que geram linguagens regulares e vice-versa (isto é, que toda linguagem regular pode ser gerada por uma tal gramática).

**Definição 3.5.1** Uma gramática  $G = \langle V, T, S, P \rangle$  diz-se **linear** se o lado direito de cada produção em  $P$  tem no máximo uma variável.

### 3.5. Gramáticas Regulares

---

**Exemplo 3.5.2** A gramática  $G = \langle \{S\}, \{a, b\}, S, P \rangle$ , com  $P$  sendo

$$S \longrightarrow aSb \mid aSbb \mid aSbbb \mid \lambda$$

é uma gramática linear.

A classe de linguagens geradas pelas gramáticas lineares é chamada de **Linguagens Lineares**. Existem diversos modelos de autômatos para esta classe de linguagens, entre eles destacamos: um tipo especial de autômato finito de duas fitas [Ros67, Har78], tradutores finitos [Ros67, Mak03], autômatos com pilha considerando no máximo uma volta [Har78, HU79, ABB97, HE04] e autômatos lineares não-determinísticos [Bed08].

Toda gramática linear pode ser transformada numa outra gramática linear equivalente, isto é que gere a mesma linguagem, e onde todas as produções tenham uma das seguintes formas:

$$\begin{aligned} A &\longrightarrow wB \\ A &\longrightarrow Bw \\ A &\longrightarrow w, \end{aligned}$$

para algum  $A, B \in V$  e  $w \in T^*$ .

Quando uma gramática tem esta forma, dizemos que está na **forma normal linear**. Por exemplo, a forma normal linear da gramática no exemplo 3.5.2, é a gramática  $G_L = \langle \{S, B\}, \{a, b\}, S, P_L \rangle$ , onde  $P_L$  tem as seguintes produções:

$$\begin{aligned} S &\longrightarrow aB \mid \lambda \\ B &\longrightarrow Sb \mid Sbb \mid Sbbb. \end{aligned}$$

É claro que para toda gramática linear há uma gramática na forma normal linear equivalente a ela.

**Definição 3.5.3** Uma gramática linear é **linear à esquerda** se todas as produções têm a forma  $A \longrightarrow Bw$  ou  $A \longrightarrow w$ . Analogamente, uma gramática linear é **linear à direita** se todas as produções têm a forma  $A \longrightarrow wB$  ou  $A \longrightarrow w$ .

Uma **gramática regular** é uma gramática linear à direita ou linear à esquerda.

Assim, nas gramáticas lineares, lineares à direita, lineares à esquerda e regulares todas as produções têm uma variável no lado esquerdo (lembre que na definição geral de gramática o lado esquerdo de uma produção pode ser qualquer cadeia em  $(V \cup T)^+$ ) e no máximo uma no lado direito, cuja localização é no final ou no inicio da cadeia. O nome “mais à direita” se deve justamente a esse fato: a variável no lado direito da produção, caso tenha uma, sempre ocorre na posição mais à direita (no final) da cadeia. Analogamente com as gramáticas lineares à esquerda, o nome se deve a que a variável no lado direito da produção, caso tenha uma, sempre ocorre na posição mais à esquerda (no inicio) da cadeia.

Observe que nem toda gramática linear é regular. Por exemplo, a gramática

$$S \longrightarrow aS \mid Sb \mid ab$$

é linear mas não é regular, pois nem é regular à direita nem regular à esquerda.

**Exemplo 3.5.4** A gramática  $G_1 = \langle \{S\}, \{a, b\}, S, P_1 \rangle$ , com  $P_1$  sendo

$$S \longrightarrow abS \mid a$$

é linear à direita. A gramática  $G_2 = \langle \{S\}, \{a, b\}, S, P_2 \rangle$ , com  $P_2$  sendo

$$S \longrightarrow Sba \mid a$$

é linear à esquerda. Ambas geram a mesma linguagem, isto é,  $L(G_1) = L(G_2)$ . Em particular a seqüência

$$S \implies abS \implies ababS \implies ababa$$

é uma derivação de  $ababa$ , com  $G_1$  e a seqüência

$$S \implies Sba \implies Sbaba \implies ababa$$

é uma derivação da mesma cadeia, mas através de  $G_2$ . A partir desse exemplo simples podemos conjecturar que  $L(G_1)$ , e portanto  $L(G_2)$  também, é a linguagem regular  $(ab)^*a$ .

**Exemplo 3.5.5** Descrever uma gramática regular que gere a linguagem regular de todas as cadeias no alfabeto  $\{0, 1\}$  que não contenham dois 0's seguidos.

Uma tal gramática deve ter as produções de modo que sempre que possa gerar quantos 1's quiser mas ao gerar um 0 deva, em seguida, gerar um 1 ou terminar de produzir. Assim, a gramática é  $G = \langle \{S, A\}, \{0, 1\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$\begin{aligned} S &\longrightarrow 0A \mid A \\ A &\longrightarrow 1A \mid 10A \mid \lambda \end{aligned}$$

**Exemplo 3.5.6** Descrever uma gramática regular que gere a linguagem, sobre o alfabeto  $\{a, b, c\}$ , de todas as cadeias da forma  $a^n b^m c^p$ , com  $n \geq 0$ ,  $m \geq 0$  e  $p \geq 0$ .

Uma tal gramática deve ter uma produção que permita a geração arbitrária de  $a$ 's, e que permita a passagem para uma outra produção que gere uma quantidade arbitrária de  $b$ 's e esta possibilite a ida a uma outra produção que gere um quantidade arbitrária de  $c$ 's. Assim, a gramática deve ter três variáveis, uma ( $A$ ) para gerar  $a$ 's, outra ( $B$ ) para gerar os  $b$ 's e a última ( $C$ ) para gerar os  $c$ 's. Uma tal gramática é definida por  $G = \langle \{A, B, C\}, \{a, b, c\}, A, P \rangle$ , onde  $P$  é composto das seguintes produções

$$\begin{aligned} A &\longrightarrow aA \mid B \\ B &\longrightarrow bB \mid C \\ C &\longrightarrow cC \mid \lambda \end{aligned}$$

### 3.6. Conexão entre Gramáticas Regulares e Linguagens Regulares

---

**Exemplo 3.5.7** Seja a linguagem  $\mathcal{L}_{Pal}$  de todas as cadeias sobre o alfabeto  $\{a, b\}$  que quando lidas da esquerda para a direita seja o mesmo que quando lida da direita para a esquerda. Cadeias deste tipo são conhecidas como palíndromos. Ou seja,

$$\mathcal{L}_{Pal} = \{w \in \{a, b\}^* \mid w = w^R\}.$$

A gramática  $G = \langle \{S, A, B\}, \{a, b\}, S, P \rangle$  com produções

$$\begin{array}{lcl} S & \longrightarrow & aA \mid bB \mid \lambda \\ A & \longrightarrow & Sa \mid \lambda \\ B & \longrightarrow & Sb \mid \lambda \end{array}$$

gera esta linguagem, isto é,  $L(G) = \mathcal{L}_{Pal}$ . Observe que a gramática  $G$  é linear mas não é linear à direita, nem linear à esquerda, portanto não é regular. No próximo capítulo mostraremos que esta linguagem não é regular e portanto, como veremos nas próximas seções, não existe qualquer gramática regular que a gere.

## 3.6 Conexão entre Gramáticas Regulares e Linguagens Regulares

Analogamente ao caso de expressões regulares, para toda linguagem regular existe uma gramática regular que a denota e vice-versa. Mostraremos esse resultado em duas partes.

### 3.6.1 Gramáticas Lineares à Direita Geram Linguagens Regulares

Primeiramente, mostraremos que uma linguagem gerada por uma gramática linear à direita é sempre regular. Para isso construiremos um AFN que simula as derivações de uma gramática linear à direita. Observe que as formas sentenciais de uma gramática linear à direita tem uma forma especial na qual existe exatamente uma variável que ocorre como símbolo mais à direita. Suponha que temos uma etapa

$$a_1a_2 \dots a_n A \implies a_1a_2 \dots a_n bB$$

numa derivação, obtida usando-se uma produção  $A \longrightarrow bB$ . O AFN correspondente pode imitar esta etapa indo do estado  $A$  para o estado  $B$ , quando o símbolo  $b$  for encontrado. Nesse esquema, o estado do autômato corresponde à variável na forma sentencial, enquanto a parte da cadeia já processada é idêntica ao prefixo terminal da forma sentencial. Esta idéia simples é a base da prova do seguinte teorema.

**Teorema 3.6.1** Seja  $G = \langle V, T, S, P \rangle$  uma gramática linear à direita. Então  $L(G)$  é uma linguagem regular.

**DEMONSTRAÇÃO:** Vamos assumir que  $V = \{V_0, V_1, \dots, V_n\}$ , com  $S = V_0$  e que para cada  $i = 0, \dots, n$ , temos uma ou mais produções da forma  $V_i \rightarrow w_i V_j$  ou  $V_i \rightarrow w_i$ . Se  $w$  é uma cadeia em  $L(G)$ , então por causa da forma das produções em  $G$ , a derivação deve ter a forma

$$\begin{aligned} V_0 &\implies w_1 V_{i_1} \\ &\implies w_1 w_2 V_{i_2} \\ &\vdots \\ &\stackrel{*}{\implies} w_1 w_2 \dots w_k V_{i_k} \\ &\implies w_1 w_2 \dots w_k w_{k+1} = w \end{aligned} \tag{3.1}$$

onde  $w_i \in T^*$  para cada  $i = 1, \dots, k + 1$ .

O autômato  $M$  a ser construído reproduzirá a derivação consumindo cada um desses  $w'_i$ s. O estado inicial do autômato será rotulado  $V_0$  e existirá um único estado final  $V_f$ , com  $f > n$ . Para cada variável  $V_i$ , com  $i = 0, \dots, n$ , existirá um estado não final rotulado  $V_i$ . Para cada produção

$$V_i \rightarrow a_1 a_2 \dots a_m V_j,$$

o autômato terá transições para conectar  $V_i$  e  $V_j$ , isto é,  $\delta$  será definido tal que

$$\delta^*(V_i, a_1 a_2 \dots a_m) = V_j.$$

Para cada produção

$$V_i \rightarrow a_1 a_2 \dots a_m,$$

a transição do autômato correspondente será

$$\delta^*(V_i, a_1 a_2 \dots a_m) = V_f.$$

Os estados intermediários necessários para isso não são de interesse e podem ser dados rótulos arbitrários. O esquema geral é mostrado na figura 3.12. O autômato completo é montado destas partes individuais.

Suponha, agora, que  $w \in L(G)$  tal que (3.1) é satisfeita. No AFN existe, por construção, um caminho de  $V_0$  a  $V_i$  rotulado  $w_1$ , um caminho de  $V_i$  a  $V_j$  rotulado  $w_2$ , e assim por diante, tal que, claramente

$$V_f \in \delta^*(V_0, w),$$

e portanto  $w$  é aceito por  $M$ .

Inversamente, suponha que  $w$  é aceito por  $M$ . Pelo modo como  $M$  foi construído, para aceitar  $w$  o autômato tem de passar pela seqüência de estados  $V_0, V_i, \dots$  para  $V_f$ , usando caminhos rotulados  $w_1, w_2, \dots$ . Portanto,  $w$  deve ter a forma  $w = w_1 w_2 \dots w_{k+1}$  e a derivação

### 3.6. Conexão entre Gramáticas Regulares e Linguagens Regulares

---

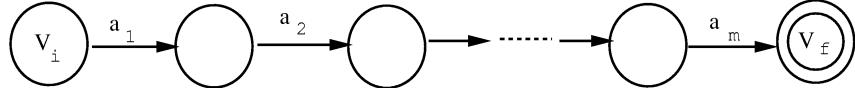
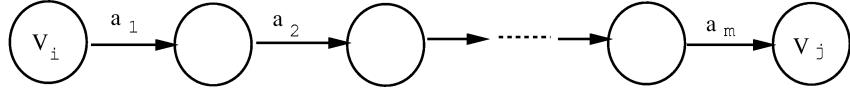


Figura 3.12: Representam  $V_i \rightarrow a_1a_2 \dots a_m V_j$  e  $V_i \rightarrow a_1a_2 \dots a_m$ , respectivamente.

$$V_0 \Rightarrow w_1 V_i \Rightarrow w_1 w_2 V_j \xrightarrow{*} w_1 w_2 \dots w_k V_f \Rightarrow w_1 w_2 \dots w_k w_{k+1}$$

para algum  $k \geq 0$ , é possível. Logo  $w$  está em  $L(G)$ , e assim o teorema está provado. ■

**Exemplo 3.6.2** Construir um autômato que aceite a linguagem gerada pela gramática

$$\begin{array}{l} V_0 \xrightarrow{} aV_1 \\ V_1 \xrightarrow{} abV_0 \mid b \end{array}$$

Começamos do grafo de transição com vértices  $V_0$ ,  $V_1$  e  $V_f$ .  $V_0$  é o estado inicial e  $V_f$  o estado final. A primeira regra de produção cria uma aresta rotulada  $a$  entre  $V_0$  e  $V_1$ . Para a segunda regra, precisamos introduzir um vértice adicional tal que exista um caminho rotulado  $ab$  entre  $V_1$  e  $V_0$ . Finalmente, precisamos adicionar uma aresta rotulada  $b$  entre  $V_1$  e  $V_f$ , dando o autômato mostrado na figura 3.13. A linguagem gerada pela gramática e reconhecida pelo autômato é a linguagem regular  $L((aab)^*ab)$ .

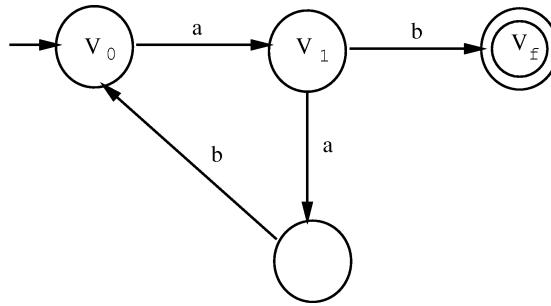


Figura 3.13: AFN resultante de uma gramática linear à direita.

#### 3.6.2 Linguagens Regulares são Geradas por Gramáticas Lineares à Direita

Para mostrar que toda linguagem regular pode ser gerada por alguma gramática linear à direita, começamos com o AFD que reconhece a linguagem e invertemos a construção mostrada no

teorema 3.6.1. Os estados do AFD tornam-se agora as variáveis da gramática, e os símbolos que causam as transições tornam-se os terminais nas produções.

**Teorema 3.6.3** *Se  $\mathcal{L}$  é uma linguagem regular, sobre o alfabeto  $\Sigma$ , então existe uma gramática linear à direita,  $G = \langle V, T, S, P \rangle$ , tal que  $L(G) = \mathcal{L}$ .*

**DEMONSTRAÇÃO:** Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD tal que  $L(M) = \mathcal{L}$ . Assumiremos que  $Q = \{q_0, q_1, \dots, q_n\}$  e  $\Sigma = \{a_1, a_2, \dots, a_m\}$ . Vamos construir uma gramática linear à direita  $G = \langle V, T, S, P \rangle$ , com  $V = \{q_0, q_1, \dots, q_n\}$ ,  $T = \Sigma$  e  $S = q_0$  tal que  $L(G) = L(M)$ . Para cada transição  $\delta(q_i, a_j) = q_k$  de  $M$ , colocamos em  $P$  a produção  $q_i \rightarrow a_j q_k$ . Além disso, para cada  $q_f$  em  $F$ , acrescentamos a  $P$  a produção  $q_f \rightarrow \lambda$ .

Primeiro, mostraremos que  $G$  definida dessa maneira pode gerar toda cadeia em  $\mathcal{L}$ . Considere  $w \in \mathcal{L}$  da forma  $w = a_i a_j \dots a_k a_l$ .

Para  $M$  aceitar essa cadeia ele deve se movimentar via

$$\delta(q_0, a_i) = q_p,$$

$$\delta(q_p, a_j) = q_r,$$

⋮

$$\delta(q_s, a_k) = q_t$$

$$\delta(q_t, a_l) = q_f \in F.$$

Por construção, a gramática terá uma produção para um desses  $\delta$ 's. Assim, podemos fazer a derivação

$$q_0 \implies a_i q_p \implies a_i a_j q_r \stackrel{*}{\implies} a_i a_j \dots a_k q_t \implies a_i a_j \dots a_k a_l q_f \implies a_i a_j \dots a_k a_l, \quad (3.2)$$

com a gramática  $G$ , e portanto  $w \in L(G)$ .

Inversamente se  $w \in L(G)$ , então sua derivação deve ter a forma (3.2). Mas isto implica que

$$\delta^*(q_0, a_i a_j \dots a_k a_l) = q_f,$$

completando a prova. ■

**Exemplo 3.6.4** *Seja o AFD do exemplo 2.1.2. Do grafo de transição e do teorema anterior, podemos observar que a gramática linear à direita que gera a linguagem reconhecida por este autômato é  $G = \langle \{q_0, q_1, q_2\}, \{0, 1\}, q_0, P \rangle$ , onde  $P$  é dado pelo seguinte conjunto de produções*

$$\begin{aligned} q_0 &\longrightarrow 0q_0 \mid 1q_1 \\ q_1 &\longrightarrow 0q_0 \mid 1q_2 \mid \lambda \\ q_2 &\longrightarrow 0q_2 \mid 1q_1 \end{aligned}$$

*A cadeia 0011001 pode ser derivada da gramática construída como segue*

$$q_0 \implies 0q_0 \implies 00q_0 \implies 001q_1 \implies 0011q_2 \implies 00110q_2 \implies 001100q_2 \implies 0011001q_1 \implies 0011001\lambda = 001100.$$

### 3.6. Conexão entre Gramáticas Regulares e Linguagens Regulares

---

#### 3.6.3 Equivalência entre Linguagens Regulares e Gramáticas Regulares

Os resultados anteriores estabelecem a conexão entre linguagens regulares e gramáticas lineares à direita. Podemos fazer uma conexão análoga entre linguagens regulares e gramáticas lineares à esquerda, mostrando, assim, a completa equivalência de gramáticas regulares e linguagens regulares.

**Teorema 3.6.5** *Uma linguagem  $\mathcal{L}$  é regular se e somente se existe uma gramática linear à esquerda,  $G$ , tal que  $L(G) = \mathcal{L}$ .*

DEMONSTRAÇÃO: Seja  $G$  uma gramática linear à esquerda. Para cada produção de  $G$  da forma

$$A \longrightarrow Bw,$$

ou

$$A \longrightarrow w,$$

troque ela por uma produção da forma

$$A \longrightarrow w^R B,$$

ou

$$A \longrightarrow w^R,$$

respectivamente. Claramente, a gramática  $G'$  resultante deste processo é linear à direita e, mais ainda,  $L(G')^R = L(G)$ .

Do exercício 24 do capítulo anterior, temos que a reversa de qualquer linguagem regular é regular. Como, pelo teorema 3.6.1,  $L(G')$  é regular, então  $L(G)$  também é uma linguagem regular.

Por outro lado, seja  $\mathcal{L}$  uma linguagem regular, então pelo exercício 24 do capítulo anterior, temos que a reversa de  $\mathcal{L}$ , isto é  $\mathcal{L}^R$ , é uma linguagem regular. Logo, pelo teorema 3.6.3, existe uma gramática linear à direita  $G$ , tal que  $L(G) = \mathcal{L}^R$ . Assim, usando um processo análogo ao anterior, podemos construir uma gramática linear à esquerda a partir de  $G$ , simplesmente, trocando as produções da forma

$$A \longrightarrow wB,$$

ou

$$A \longrightarrow w,$$

pelas produções da forma

$$A \longrightarrow Bw^R,$$

ou

$$A \longrightarrow w^R,$$

respectivamente. Claramente, a gramática linear à esquerda  $G'$ , resultante deste processo, é tal que  $L(G') = L(G)^R$ . Logo,  $L(G') = L(G)^R = (\mathcal{L}^R)^R = \mathcal{L}$ . Portanto, para cada linguagem regular,  $\mathcal{L}$ , existe uma gramática linear à esquerda que gera ele. ■

Colocando os teoremas 3.6.1, 3.6.3 3.6.5 juntos temos a equivalência entre linguagens regulares e gramáticas regulares.

**Teorema 3.6.6** *Uma linguagem,  $\mathcal{L}$ , é regular se e somente se existe uma gramática regular  $G$  tal que  $L(G) = \mathcal{L}$ .*

DEMONSTRAÇÃO: Direto dos teoremas 3.6.1, 3.6.3 e 3.6.5. ■

### 3.7. Exercícios

---

## 3.7 Exercícios

1. Construa um autômato para reconhecer as linguagens representadas pelas expressões regulares
  - (a)  $1^*0^*$
  - (b)  $(00 + 10)^*$
  - (c)  $(01 + 10)^*$
  - (d)  $(0 + 10 + 110)^*(\lambda + 1 + 11)$
2. Escrever uma expressão regular para as seguintes linguagens sobre o alfabeto  $\Sigma = \{a, b\}$ .
  - (a)  $\mathcal{L} = \{a^n b^m / n \geq 1, m \geq 1, n \cdot m \geq 3\}$ .
  - (b)  $\mathcal{L} = \{w \in \Sigma^* / N_a(w) \leq 3\}$ .
  - (c)  $\mathcal{L} = \{w \in \Sigma^* / \text{existem cadeias } u \text{ e } v \text{ tais que } w = u111v \text{ e } N_a(v) = 1 \text{ ou } N_a(v) = 3\}$
  - (d)  $\mathcal{L} = \{w \in \Sigma^* / \text{existem cadeias } u \text{ e } v \text{ tais que } w = u111v \text{ e } N_a(u) = 3 \text{ ou } N_b(v) = 3\}$
  - (e)  $\mathcal{L} = \{w \in \Sigma^* / N_1(w) \text{ é ímpar e múltipla de } 3\}$
3. Escrever expressões regulares para as seguintes linguagens, sobre  $\{0, 1\}$ 
  - (a) Todas as cadeias terminando em 01.
  - (b) Todas as cadeias não terminando em 01.
  - (c) Todas as cadeias contendo um número par de zeros.
  - (d) Todas as cadeias com no máximo duas ocorrências da subcadeia 00.
  - (e) Todas as cadeias que contenham as subcadeias 000 e 111.
  - (f) Todas as cadeias que não contenham as subcadeias 00 e 11.
  - (g) Todas as cadeias que para alguma ocorrência de dois zeros eles estejam separados por uma subcadeia de tamanho  $3i$ , para algum  $i \geq 0$ .
  - (h) Todas as cadeias que toda ocorrência de dois zeros seguidos eles estejam separados por uma quantidade ímpar de uns.
  - (i) Todas as cadeias com uma quantidade par de 1's.
  - (j) Todas as cadeias com uma quantidade ímpar de 0's.
  - (k) Todas as cadeias com uma quantidade par de 1's e ímpar de 0's.
  - (l) Todas as cadeias que não contenham qualquer ocorrência da subcadeia 000.
  - (m) Todas as cadeias que contém ao menos uma ocorrência da subcadeia 111 mas nenhuma ocorrência da subcadeia 000.
  - (n) Todas as cadeias com exatamente uma ocorrência da subcadeia 111.
  - (o) Todas as cadeias com no máximo uma ocorrência da subcadeia 100.
  - (p) Todas as cadeias com exatamente duas ocorrências da subcadeia 000 e as quais estejam separadas por uma quantidade ímpar de uns.

- (q) Todas as cadeias que não sejam da forma  $1^k$  onde  $k$  é múltiplo de 3.
4. Escrever expressões regulares para as seguintes linguagens sobre  $\Sigma = \{a, b, c\}$
- Todas as cadeias contendo exatamente um  $a$ .
  - Todas as cadeias contendo não mais do que três  $a$ 's.
  - Todas as cadeias que contém no mínimo uma ocorrência de cada símbolo em  $\Sigma$ .
  - Todas as cadeias que contém três  $a$ 's, três  $b$ 's e três  $c$ 's consecutivos, nessa ordem. Por exemplo as cadeias  $abaaacbabbbaccccba$  e  $ccccaaababbabcccabccc$  fazem parte dessa linguagem, já a cadeia  $aaabbcccbba$  não.
  - Todas as cadeias que não contém dois  $a$ 's consecutivos.
  - Todas as cadeias com uma quantidade par de  $a$ 's (0 é considerado como par).
5. Demonstre que
- $(a + b)^* \equiv (a^*b^*)^*$
  - $a^* + b^* \not\equiv (a + b)^*$
  - $a^*b^* \not\equiv (ab)^*$
  - $(b + ab)^*(a + \lambda) \equiv (a + \lambda)(ba + b)^*$
  - $aa^* \equiv a^*a$
- onde  $r_1 \not\equiv r_2$  se, e somente se,  $L(r_1) \neq L(r_2)$ .
6. Prove que para toda expressão regular  $r_1$ ,  $r_2$  e  $r_3$
- $(r_1^*)^* \equiv r_1^*$
  - $(r_1 + r_2) \equiv r_2 + r_1$
  - $(r_1 + r_2) + r_3 \equiv r_1 + (r_2 + r_3)$
  - $r_1(r_2 + r_3) \equiv r_1r_2 + r_1r_3$
  - $r_1 + r_1 \equiv r_1$
  - $(r_1 + r_2)^* \equiv (r_1^*r_2^*)^*$
  - $r_1^*r_1^* \equiv r_1^*$
  - $(r_1r_2)^*r_1 \equiv r_1(r_2r_1)^*$
7. Achar autômatos finitos que aceitem as seguintes linguagens
- $L(aa^*(a + b))$
  - $L((ab + abb)^*(bb + aa + \lambda))$
  - $L((ab + b)^*(a + \lambda))^*$
  - $L(aa^* + aba^*b^*)$
  - $L(aa^*bb^*aa^*)$
  - $L(ab(a + ab)(a + aa))$

### 3.7. Exercícios

---

- (g)  $L(a^*(b(bb)^*aa)^*)$   
 (h)  $L((a+b)^*a(bbb+bab)^*a(a+b)^*)$
8. Construir seguindo o algoritmo da seção 3.4.2 a expressão regular que denota a linguagem reconhecida pelo AFD da figura 3.14.

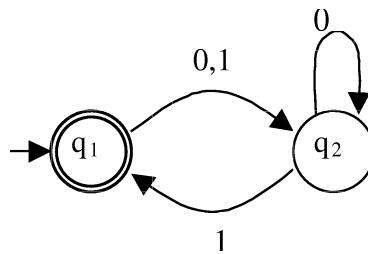


Figura 3.14: Autômato finito determinístico do exercício 8.

9. Construir uma gramática linear á direita e uma linear à esquerda para as linguagens
- $L((aab^*abab)^*)$
  - $L((a+b)^*aaa(a+b)^*bbb(a+b)^*)$
  - $L((a+b)^*aaa)$
  - $L((ab+ba)^*(a+b+\lambda))$
  - $L((a(aa)^*(bb)^*)^*aa^*)$
  - $L((ab^*a+b)^*ab^* + (ba^*ba^*b+a)^*)$
10. Construir uma gramática regular que gere cada uma das linguagens do exercício ??.
11. Construir uma gramática regular que gere cada uma das seguintes linguagens sobre o alfabeto  $\{a, b\}$ .
- Todas as cadeias que terminem com três  $a'$ s.
  - Todas as cadeias que contenha uma subcadeia do tipo  $ab^n a$ , para algum  $n \geq 0$ .
  - Todas as cadeias diferentes de  $(aaa)^k$ , para qualquer  $k \geq 0$ .
  - Todas as cadeias diferentes de  $a^k$ , para qualquer  $k \geq 3$ .
  - Todas as cadeias da forma  $aawaa$ , onde  $|w|$  é múltiplo de três.
  - Todas as cadeias que contém exatamente uma única ocorrência da subcadeia  $aa$ .
  - Todas as cadeias com uma quantidade par de  $a$ 's mas sem qualquer ocorrência da subcadeia  $aaa$ .
  - Todas as cadeias que contenham exatamente uma única ocorrência de três símbolos iguais.

12. Achar as gramáticas regulares para as linguagens reconhecidas pelos autômatos da figura 3.15 seguindo o algoritmo da subseção 3.6.2.
13. Construir um AFN, seguindo o algoritmo da subseção 3.6.1, para cada uma das seguintes gramáticas
  - (a)  $S \longrightarrow abA$   
 $A \longrightarrow baB$   
 $B \longrightarrow aA \mid bb$
  - (b)  $S \longrightarrow aaB \mid b;$   
 $B \longrightarrow bbS;$
  - (c)  $S \longrightarrow aA \mid bS \mid \lambda;$   
 $A \longrightarrow aB \mid bS \mid \lambda;$   
 $B \longrightarrow aaS \mid bS \mid \lambda$

### 3.7. Exercícios

---

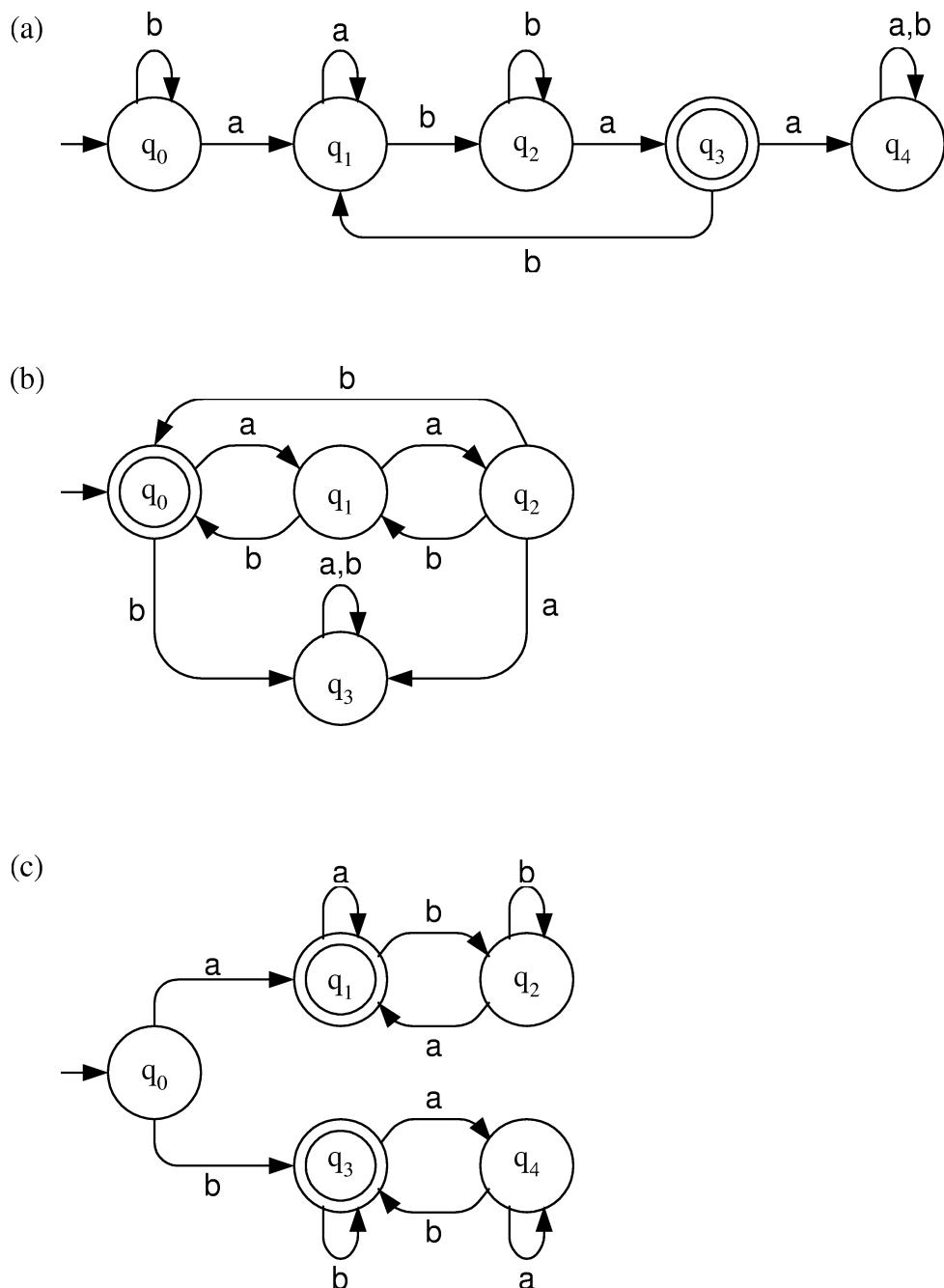


Figura 3.15: AFD's do exercício 12

## Capítulo 4

# Propriedades das Linguagens Regulares

Estamos no momento de colocar a seguinte questão: quão geral são as linguagens regulares? Seria toda linguagem formal regular? Talvez qualquer conjunto que possamos especificar seja aceito por algum autômato finito, embora complexo. Como veremos essa conjectura é falsa. Para entender essa resposta, devemos nos aprofundar na natureza das linguagens regulares e ver que propriedades a família como um todo tem.

A primeira questão que levantaremos é o que acontece quando operamos com linguagens regulares. As operações que consideraremos são operações tais como concatenação, união, etc. É a linguagem resultante ainda regular? Nos referiremos a isso como uma questão de *fecho*. Propriedades de fecho, embora, principalmente, de interesse teórico, permiti-nos-á discriminar muitas linguagens regulares que encontraremos.

A segunda questão que levantaremos trata de nossa habilidade para decidir sobre certas propriedades. Por exemplo, podemos fazer um algoritmo que decida se uma linguagem regular arbitrária é finita ou não? Como veremos, tais questões são facilmente respondidas para a classe das linguagens regulares, mas não para outras classes de linguagens.

Finalmente, consideremos a importante questão: como podemos dizer se uma dada linguagem é regular ou não? Se a linguagem é de fato regular, podemos sempre mostrar isso exibindo um AFD que a reconhece, ou uma expressão regular que a denota ou uma gramática regular que a gera. Mas se não for o caso, precisaríamos de outra abordagem, pois o fato de não termos conseguido encontrar um AFD que reconheça a linguagem não significa necessariamente que tal autômato não exista. Uma maneira de mostrar que uma linguagem não é regular é estudar as propriedades gerais das linguagens regulares.

### 4.1 Propriedades de Fecho de Linguagens Regulares

Considere a seguinte questão: dadas duas linguagens regulares arbitrárias  $\mathcal{L}_1$  e  $\mathcal{L}_2$ , é sua intersecção também regular? Em exemplos específicos a resposta pode ser óbvia, mas aqui pretendemos atacar o problema em geral. Formularemos questões análogas para as demais operações.

#### 4.1. Propriedades de Fecho de Linguagens Regulares

---

**Teorema 4.1.1** Se  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são linguagens regulares, então  $\mathcal{L}_1 \cup \mathcal{L}_2$ ,  $\mathcal{L}_1\mathcal{L}_2$ ,  $\mathcal{L}_1^*$ ,  $\overline{\mathcal{L}_1}$ ,  $\mathcal{L}_1 \cap \mathcal{L}_2$  e  $\mathcal{L}_1 - \mathcal{L}_2$  também são. Dizemos com isso que a família das linguagens regulares é fechada sob união, concatenação, fecho estrela, complemento e intersecção.

**DEMONSTRAÇÃO:** Se  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são linguagens regulares, então existem expressões regulares  $r_1$  e  $r_2$  tais que  $L(r_1) = \mathcal{L}_1$  e  $L(r_2) = \mathcal{L}_2$ .

**União e fecho estrela:** Da definição 3.2.1, temos

$$\mathcal{L}_1 \cup \mathcal{L}_2 = L(r_1) \cup L(r_2) = L(r_1 + r_2)$$

$$\mathcal{L}_1\mathcal{L}_2 = L(r_1)L(r_2) = L(r_1r_2)$$

$$\mathcal{L}_1^* = (L(r_1))^* = L(r_1^*)$$

Portanto, o fecho sob união, concatenação e fecho-estrela é imediato.

**Complemento:** Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD que aceita  $\mathcal{L}_1$ . Então, trivialmente, o AFD

$$\widehat{M} = \langle Q, \Sigma, \delta, q_0, Q - F \rangle,$$

aceita  $\overline{\mathcal{L}_1}$ . Observe que na definição de um AFD, assumimos que  $\delta^*$  era uma função total, ou seja  $\delta^*(q_0, w)$  está definida para todo  $w \in \Sigma^*$ . Conseqüentemente, ou  $\delta^*(q_0, w)$  é um estado final, e nesse caso  $w \in \mathcal{L}_1$ , ou não ( $\delta^*(q_0, w) \in Q - F$ ) e nesse caso  $w \in \overline{\mathcal{L}_1}$ .

**Intersecção:** Sejam  $M_1 = \langle Q, \Sigma_1, \delta_1, q_0, F_1 \rangle$  e  $M_2 = \langle P, \Sigma_2, \delta_2, p_0, F_2 \rangle$  AFD's que reconhecem  $\mathcal{L}_1$  e  $\mathcal{L}_2$ , ou seja  $\mathcal{L}_1 = L(M_1)$  e  $\mathcal{L}_2 = L(M_2)$ . Vamos construir a partir de  $M_1$  e  $M_2$  o autômato

$$\widehat{M} = \langle \widehat{Q}, \Sigma_1 \cap \Sigma_2, \widehat{\delta}, (q_0, p_0), \widehat{F} \rangle,$$

cujo conjunto de estados,  $\widehat{Q} = Q \times P$ , consiste de pares  $(q_i, p_j)$ , e cuja função de transição  $\widehat{\delta}$  é tal que  $\widehat{M}$  está no estado  $(q_i, p_j)$ , se  $M_1$  está no estado  $q_i$  e  $M_2$  está no estado  $p_j$ . Isto é conseguido tomando

$$\widehat{\delta}((q_i, p_j), a) = (\delta_1(q_i, a), \delta_2(p_j, a)).$$

$\widehat{F}$  é definido como o conjunto de todos os  $(q_i, p_j)$  tal que  $q_i \in F_1$  e  $p_j \in F_2$ , isto é,  $\widehat{F} = F_1 \times F_2$ .

É fácil ver que  $w \in \mathcal{L}_1 \cap \mathcal{L}_2$  se, e somente se,  $w \in L(\widehat{M})$ . Assim,  $L(\widehat{M}) = L(M_1) \cap L(M_2)$ . Conseqüentemente,  $\mathcal{L}_1 \cap \mathcal{L}_2$  é regular.

**Diferença:** A família das linguagens regulares é fechada com respeito à diferença, se quando  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são linguagens regulares, então  $\mathcal{L}_1 - \mathcal{L}_2$  também é regular. Mas

$$\mathcal{L}_1 - \mathcal{L}_2 = \mathcal{L}_1 \cap \overline{\mathcal{L}_2}.$$

Como já mostramos aqui que as linguagens regulares são fechadas sobre a intersecção e complemento, podemos concluir que  $\mathcal{L}_1 - \mathcal{L}_2$  é uma linguagem regular. ■

**Exemplo 4.1.2** Sejam  $M_1$  e  $M_2$  os AFD's descritos nas figuras 4.1 e 4.2, respectivamente. A intersecção das linguagens  $L(M_1)$  com  $L(M_2)$ , é reconhecida pelo autômato do exercício 2.2.6, o qual, se substituirmos  $PP$  por  $(q_0, q_0)$ ,  $PI$  por  $(q_0, q_1)$ ,  $IP$  por  $(q_1, q_0)$  e  $II$  por  $(q_1, q_1)$ , é exatamente o mesmo autômato que o construído usando o mecanismo descrito no teorema anterior, para a intersecção.

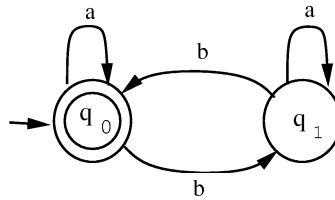


Figura 4.1: AFD  $M_1$

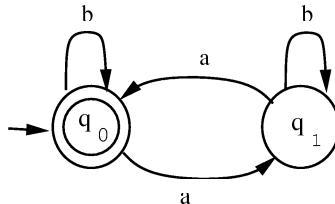


Figura 4.2: AFD  $M_2$

**Exemplo 4.1.3** Sejam as linguagens  $\mathcal{L}_1 = \{waa / w, u \in \{a, b\}^*\}$  e  $\mathcal{L}_2 = \{wbbbu / w, u \in \{a, b\}^*\}$ . Os autômatos ilustrados na figura 4.3.A e 4.3.B, reconhecem as linguagens  $\mathcal{L}_1$  e  $\mathcal{L}_2$ , respectivamente. Note que  $\mathcal{L}_1 = L((a+b)^*aa(a+b)^*)$  e  $\mathcal{L}_2 = L((a+b)^*bbb(a+b)^*)$ . A intersecção destas linguagens, isto é  $\mathcal{L}_1 \cap \mathcal{L}_2$ , é reconhecida pelo autômato ilustrado na figura 4.4.

**Definição 4.1.4** Sejam  $\Sigma_1$  e  $\Sigma_2$  alfabetos. Uma função  $h : \Sigma_1 \longrightarrow \Sigma_2^*$  é chamada um **homomorfismo**.

#### 4.1. Propriedades de Fecho de Linguagens Regulares

---

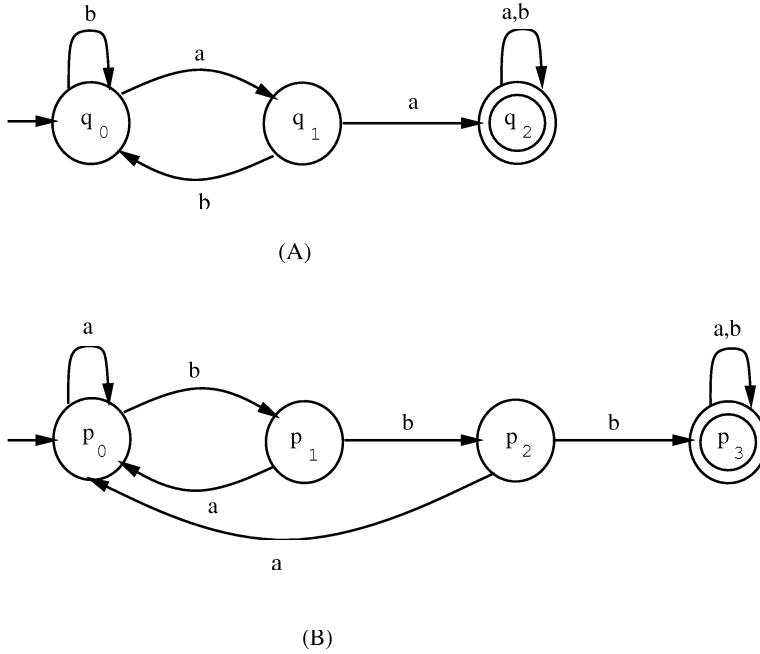


Figura 4.3: AFD's que reconhecem as linguagens  $\mathcal{L}_1 = L((a + b)^*aa(a + b)^*)$  e  $\mathcal{L}_2 = L((a + b)^*bbb(a + b)^*)$ .

Em outras palavras, um homomorfismo é uma substituição no qual um simples símbolo é trocado por uma cadeia. É possível estender a função  $h$  para uma função  $\hat{h}$  cujo domínio sejam cadeias em  $\Sigma_1$  em vez de símbolos de modo óbvio.

**Definição 4.1.5** Sejam  $\Sigma_1$  e  $\Sigma_2$  alfabetos e  $h : \Sigma_1 \longrightarrow \Sigma_2^*$  um homomorfismo. Um **homomorfismo estendido** de  $h$ , denotado por  $\hat{h}$  é a função  $\hat{h} : \Sigma_1^* \longrightarrow \Sigma_2^*$  definida por:

1.  $\hat{h}(\lambda) = \lambda$
2.  $\hat{h}(a) = h(a)$  para cada  $a \in \Sigma_1$
3.  $\hat{h}(wa) = \hat{h}(w)h(a)$  para cada  $w \in \Sigma_1^*$  e  $a \in \Sigma_1$ .

Assim, se  $w = a_1a_2 \dots a_n$ , então  $\hat{h}(w) = h(a_1)h(a_2) \dots h(a_n)$ . Note que trivialmente  $\hat{h}(wv) = \hat{w}\hat{v}$ .

Se  $\mathcal{L}$  é uma linguagem, sobre  $\Sigma_1$ , sua **imagem homomorfa** é definida como

$$h(\mathcal{L}) = \{\hat{h}(w) / w \in \mathcal{L}\}.$$

Note que  $\hat{h}(wv) = \hat{w}\hat{v}$ . Portanto, trivialmente, temos que  $h(\mathcal{L}_1\mathcal{L}_2) = h(\mathcal{L}_1)h(\mathcal{L}_2)$ .

Por simplicidade notacional e em vista de que  $\hat{h}$  é uma extensão óbvia de  $h$ , de aqui em diante usaremos o mesmo nome de função ( $h$ ) para ambos o homomorfismo e sua extensão.

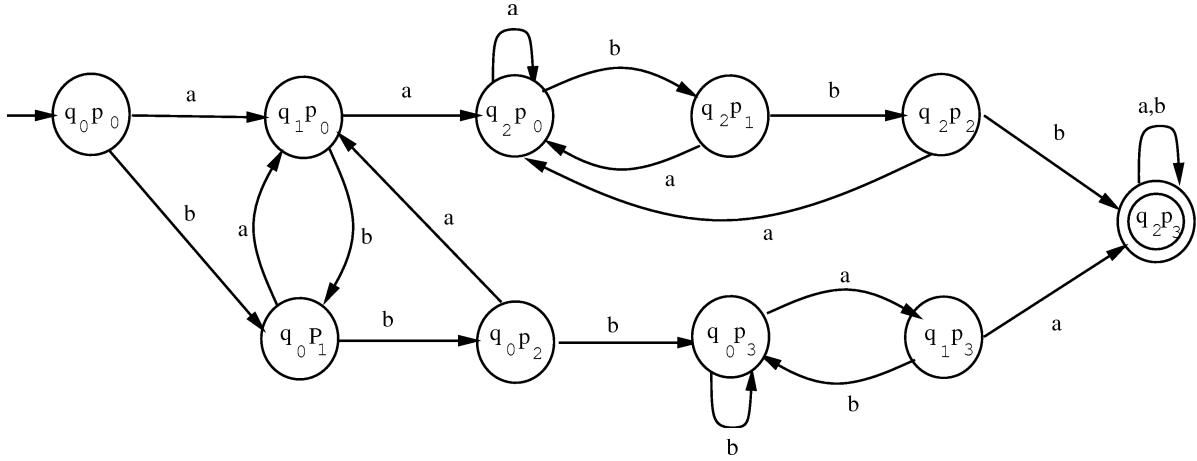


Figura 4.4: AFD que reconhece  $L((a+b)^*aa(a+b)^*) \cap L((a+b)^*bbb(a+b)^*)$ .

**Exemplo 4.1.6** Sejam  $\Sigma_1 = \{a, b\}$  e  $\Sigma_2 = \{a, b, c\}$ . Defina  $h$  por  $h(a) = ab$  e  $h(b) = bbc$ . Então  $h(aba) = h(a)h(b)h(a) = abbbcab$ . A imagem homomorfa de  $\mathcal{L} = \{aa, aba\}$  é a linguagem

$$h(\mathcal{L}) = \{abab, abbbcab\}.$$

**Exemplo 4.1.7** Tome  $\Sigma_1 = \{a, b\}$  e  $\Sigma_2 = \{b, c, d\}$ . Defina  $h : \Sigma_1 \longrightarrow \Sigma_2^*$  por

$$\begin{aligned} h(a) &= dbcc \\ h(b) &= bdc \end{aligned}$$

Se  $\mathcal{L}$  é a linguagem regular denotada por  $r = (ab + b^*)(aa)^*$ , então

$$r_1 = (h(a)h(b) + h(b)^*)(h(a)h(a))^* = (dbccbdc + (bdc)^*)(dbccdbcc)^*,$$

denota a linguagem regular  $h(\mathcal{L})$ .

**Teorema 4.1.8** Seja  $h : \Sigma_1 \longrightarrow \Sigma_2^*$  um homomorfismo. Se  $\mathcal{L}$  é uma linguagem regular, sobre o alfabeto  $\Sigma_1$ , então sua imagem homomorfa é uma linguagem regular sobre o alfabeto  $\Sigma_2$ . A família das linguagens regulares é, portanto, fechada sob homomorfismos arbitrários.

**DEMONSTRAÇÃO:** Seja  $\mathcal{L}$  uma linguagem regular, sobre  $\Sigma_1$ , denotada por alguma expressão regular  $r$ , isto é  $L(r) = \mathcal{L}$ . Seja  $h(r)$  a expressão obtida ao substituir em  $r$ , cada símbolo  $a \in \Sigma_1$  por  $h(a) \in \Sigma_2^*$ , de modo similar ao exemplo 4.1.7. Note que no caso particular das expressões regulares primitivas  $\lambda$  e  $\emptyset$ ,  $h(\lambda) = \lambda$  e  $h(\emptyset) = \emptyset$ . Pode ser mostrado diretamente apelando à definição de uma expressão regular que o resultado  $(h(r))$  é uma expressão regular. É fácil ver que a expressão regular  $h(r)$  denota  $h(L(r))$ . Tudo que precisamos fazer é mostrar que para todo  $w \in L(r)$ , o correspondente  $h(w)$  está em  $L(h(r))$  e, inversamente, que para todo  $v \in L(h(r))$

## 4.2. Questões Elementares sobre Linguagens Regulares

---

existe  $w \in \mathcal{L}$  tal que  $v = h(w)$ . Noutras palavras, devemos mostrar que  $L(h(r)) = h(L(r))$ . Deixamos os detalhes como exercício. ■

A importância do homomorfismo, é que pode servir para simplificar demonstrações. Por exemplo, se sabemos que a linguagem  $\mathcal{L} = \{10^n 1^k 0 / n, k \geq 1\}$ , então via o homomorfismo  $h(0) = aa$  e  $h(1) = bb$ , saberemos que a linguagem  $\mathcal{L}' = \{bb(aa)^n(bb)^k aa / n, m \text{ são pares maiores ou iguais a } 2\}$  também é uma linguagem regular, não precisamos construir um autômato finito que reconheça ela, ou uma gramática regular que a gere ou uma expressão regular que denote essa linguagem.

As linguagens regulares também são fechadas sobre reversões, prefixos e sufixos, como mostra o seguinte teorema.

**Teorema 4.1.9** *Se  $\mathcal{L}$  é uma linguagem regular, então  $\mathcal{L}^R$ ,  $\mathcal{L}^P$  e  $\mathcal{L}^S$  também são regulares.*

**DEMONSTRAÇÃO:** Se  $\mathcal{L}$  é regular, então existe uma gramática linear à direita que gera  $\mathcal{L}$ . Logo, o procedimento descrito no teorema 3.6.5 teríamos uma gramática linear à esquerda que gera  $\mathcal{L}^R$  e portanto  $\mathcal{L}^R$  é uma linguagem regular.

Se  $\mathcal{L}$  é regular, então existe um AFD,  $M$ , que reconhece  $\mathcal{L}$ . Construa um novo AFD a partir de  $M$  dando o status de estado final a cada estado que esteja em algum caminho do estado inicial a algum estado final de  $M$ . Claramente,  $L(M) = \mathcal{L}^P$ .

Por outro lado, como trivialmente  $\mathcal{L}^S = ((\mathcal{L}^R)^P)^R$ , e linguagens regulares são fechadas sobre prefixos e reversões, então  $\mathcal{L}^S$  é também uma linguagem regular. ■

## 4.2 Questões Elementares sobre Linguagens Regulares

Podemos discutir agora o seguinte problema fundamental: dada uma linguagem arbitrária  $\mathcal{L}$  e uma cadeia  $w$ , podemos determinar algorítmicamente se  $w$  pertence a  $\mathcal{L}$  ou não? Esta questão é conhecida como **questão de pertinência** e o método de respondê-la é o algoritmo de pertinência. A questão da existência e natureza desse algoritmo será de muito interesse no que seguirá. É um problema em geral difícil. Para as linguagens regulares, no entanto, ele é fácil.

O que queremos dizer por “dado uma linguagem ...”? Em muitos argumentos é importante que essa frase seja clara. Usamos várias maneiras de descrever as linguagens regulares: descrição verbal informal, notação de conjunto, autômatos finitos, expressões regulares e gramáticas regulares. Somente as três últimas são suficientemente bem definidas para usar em provas de teoremas. Portanto, dizemos que uma linguagem regular é dada numa **representação padrão** se, e somente se, ela é descrita por um autômato finito, uma expressão regular ou uma gramática regular.

**Teorema 4.2.1** *Dado uma representação padrão de uma linguagem regular arbitrária  $\mathcal{L}$ , sobre  $\Sigma$ , e uma cadeia  $w \in \Sigma^*$ , existe um algoritmo para determinar se  $w \in L$  ou não.*

**DEMONSTRAÇÃO:** Primeiro representamos a linguagem por algum autômato finito (se a representação padrão da linguagem é outra, então podemos, usando os algoritmo deste capítulo, sempre transformar este num autômato finito equivalente). Depois simulamos o comportamento

do autômato para a entrada  $w$ . Se ao finalizar a simulação o autômato pára num estado final, então o algoritmo aceita a cadeia senão o algoritmo rejeita a cadeia. ■

Existem diversos simuladores de autômatos finitos disponíveis na internet, por exemplo o SAGEMoLic desenvolvido pelo grupo de Teoria da Computação da Universidade de Brasília e disponível em: <http://www.mat.unb.br/ayala/TCgroup/SAGEMoLiC/>.

Outras questões importantes são:

1.  $\mathcal{L}$  é uma linguagem vazia ou não vazia?
2.  $\mathcal{L}$  é uma linguagem finita ou infinita?
3. São  $\mathcal{L}_1$  e  $\mathcal{L}_2$  a mesma linguagem?
4.  $\mathcal{L}_1$  é um subconjunto de  $\mathcal{L}_2$ ?

**Teorema 4.2.2** *Existe um algoritmo para determinar se uma dada linguagem regular, na representação padrão, é vazia, finita ou infinita.*

**DEMONSTRAÇÃO:** A resposta é imediata a partir da representação da linguagem como grafo de transição de um AFD. Se existe um caminho do vértice inicial a qualquer vértice final, então a linguagem é não-vazia.

Para determinar se a linguagem é infinita ou não, achamos todos os vértices que são a base de algum ciclo. Se alguns desses estão sobre um caminho de um vértice inicial a um final, a linguagem é infinita. Caso contrário, ela é finita. ■

Algoritmos para encontrar caminhos e ciclos são bem conhecidos em teoria dos grafos.

Outra forma de verificar se uma linguagem é infinita, é verificar se na representação padrão de expressões regulares, ela contém o fecho estrela  $*$  aplicado a uma expressão contendo pelo menos um símbolo de  $\Sigma$ .

A questão da igualdade de duas linguagens é também um problema prático importante. Freqüentemente, existem várias definições de uma linguagem de programação e, precisamos saber, apesar de suas diferenças aparente, se elas especificam a mesma linguagem. Esse, em geral, é um problema difícil. Mesmo para linguagens regulares o argumento não é óbvio. Não é possível argumentar na comparação palavra a palavra, pois ele só funciona para linguagens finitas. Não é também fácil apelar para expressões regulares, gramáticas regulares ou AFD's, pois existem, por exemplo, infinitas expressões regulares para denotar a mesma linguagem. Uma solução elegante usa a propriedade de fecho.

**Teorema 4.2.3** *Dadas as linguagens regulares  $\mathcal{L}_1$  e  $\mathcal{L}_2$ , na representação padrão, existe um algoritmo para determinar se  $\mathcal{L}_1 = \mathcal{L}_2$  ou não.*

### **4.3. Identificando Linguagens Não Regulares**

---

DEMONSTRAÇÃO: Usando  $\mathcal{L}_1$  e  $\mathcal{L}_2$  podemos construir a linguagem

$$\mathcal{L}_3 = (\mathcal{L}_1 \cup \mathcal{L}_2) - (\mathcal{L}_1 \cap \mathcal{L}_2).$$

$\mathcal{L}_3$  é regular, pois  $\overline{\mathcal{L}_1}$  e  $\overline{\mathcal{L}_2}$  são e pelo teorema 4.1.1, a união, intersecção e diferença de duas linguagens regulares são regulares. Portanto, podemos achar um AFD,  $M$ , que reconheça  $\mathcal{L}_3$ . Observe que se  $\mathcal{L}_1 = \mathcal{L}_2$ , então

$$\begin{aligned}\mathcal{L}_3 &= (\mathcal{L}_1 \cup \mathcal{L}_2) - (\mathcal{L}_1 \cap \mathcal{L}_2) \\ &= (\mathcal{L}_1 \cup \mathcal{L}_1) - (\mathcal{L}_1 \cap \mathcal{L}_1) \\ &= \mathcal{L}_1 - \mathcal{L}_1 \\ &= \emptyset\end{aligned}$$

Se  $\mathcal{L}_1 \neq \mathcal{L}_2$ , então ou existe um elemento  $w \in \mathcal{L}_1$  tal que  $w \notin \mathcal{L}_2$  ou existe um elemento  $w \in \mathcal{L}_2$  tal que  $w \notin \mathcal{L}_1$ . Em ambos os casos  $w \in \mathcal{L}_1 \cup \mathcal{L}_2$  e  $w \notin \mathcal{L}_1 \cap \mathcal{L}_2$ . Logo,  $\mathcal{L}_3 = (\mathcal{L}_1 \cup \mathcal{L}_2) - (\mathcal{L}_1 \cap \mathcal{L}_2) \neq \emptyset$ .

Assim,  $\mathcal{L}_1 = \mathcal{L}_2$  se, e somente se,  $\mathcal{L}_3 = \emptyset$ .

Logo, podemos usar o algoritmo do teorema 4.2.2 para determinar se  $\mathcal{L}_3$  é vazia ou não e assim concluir se  $\mathcal{L}_1 = \mathcal{L}_2$  ou  $\mathcal{L}_1 \neq \mathcal{L}_2$ . ■

## **4.3 Identificando Linguagens Não Regulares**

As linguagens regulares podem ser infinitas. No entanto, o fato delas poderem ser associados com autômatos que tem memória finita, impõe alguns limites na estrutura das linguagens regulares. Nossa intuição diz que uma linguagem é regular somente se, em processando qualquer cadeia, a informação a ser armazenada em qualquer estágio é estritamente limitada. Isso é verdade, mas teremos de mostrar precisamente.

### **4.3.1 Usando o Princípio da Casa de Pombos**

O termo “princípio da casa de pombos” é usado pelos matemáticos para se referir à seguinte simples observação: Se colocamos  $n$  objetos (pombos) em  $m$  caixas (casa de pombos), e se  $n > m$ , então no mínimo uma caixa deve conter mais de um ítem. Nesta metáfora, a casa dos pombos correspondem aos estados de um AFD e os  $n$  pombos a uma cadeia de tamanho  $n$ .

**Exemplo 4.3.1** A linguagem  $\mathcal{L} = \{a^n b^n / n \geq 0\}$  é regular? A resposta é não. Mostraremos usando uma prova por contradição. Suponha que  $\mathcal{L}$  é regular. Então, existe um afd  $M = \langle Q, \{a, b\}, \delta, q_0, F \rangle$  que a reconhece. Agora, olhemos para  $\delta^*(q_0, a^i)$ , com  $i = 1, 2, 3, \dots$ . Como existe um número ilimitado de  $i$ 's, mas somente um número limitado de estados em  $M$ , o princípio da casa de pombos nos diz que para  $n > |Q|$  deve existir algum estado, digamos  $q$ , tal que

$$\delta^*(q_0, a^n) = q \text{ e } \delta^*(q_0, a^m) = q, \text{ para algum } m \neq n.$$

Mas, como  $M$  aceita  $a^n b^n$ , devemos ter

$$\begin{aligned}\delta^*(q_0, a^n b^n) &= q_f \in F \\ \delta^*(\delta^*(q_0, a^n), b^n) &= q_f \in F \\ \delta^*(q, b^n) &= q_f \in F.\end{aligned}$$

Disso podemos concluir que

$$\delta^*(q_0, a^m b^n) = \delta^*(\delta^*(q_0, a^m), b^n) = \delta^*(q, b^n) = q_f \in F.$$

Isto contradiz a suposição original de que  $M$  aceita  $a^m b^n$  somente se  $m = n$  (e neste caso  $m \neq n$ ). Logo,  $\mathcal{L}$  não pode ser regular.

Neste argumento, o princípio da casa de pombos é uma maneira de estabelecer precisamente o que queremos dizer quando dizemos que um autômato finito tem memória limitada. Para reconhecer todos  $a^n b^n$ , um autômato teria de distinguir todos os prefixos de  $a^n$ . Mas como existe somente um número finito de estados internos para fazer isso, quando  $n$  for maior que a quantidade de estados existiriam alguns prefixos de  $a^n$  para os quais essa distinção não poderia ser feita.

Para usar esse tipo de argumento numa variedade de situações é conveniente codificá-lo como um teorema geral. Existem várias maneiras de fazer isso, a que faremos é talvez a mais famosa.

### 4.3.2 Lema do Bombeamento para Linguagens Regulares

O resultado que segue, conhecido como *lema do bombeamento* para linguagens regulares, usa o princípio da casa de pombos numa outra forma. A prova é baseada na observação de que num grafo de transição com  $n$  vértices, qualquer caminho de comprimento  $n$  ou mais longo deve repetir algum vértice, isto é, contém um ciclo. O lema do bombeamento só precisará analisar linguagens infinitas, pois trivialmente toda linguagem finita necessariamente é regular.

**Teorema 4.3.2 (Lema do bombeamento)** Seja  $\mathcal{L}$  uma linguagem infinita. Se  $\mathcal{L}$  é regular então, existe um inteiro positivo  $m$  tal que todo  $w \in \mathcal{L}$ , com  $|w| \geq m$ , pode ser decomposto como  $w = xyz$ , com  $|xy| \leq m$  e  $|y| \geq 1$  tal que

$$w_i = xy^i z, \quad (4.1)$$

está também em  $\mathcal{L}$ , para todo  $i = 0, 1, 2, \dots$

**DEMONSTRAÇÃO:** Se  $\mathcal{L}$  é regular, existe um AFD que a reconhece. Seja  $M$  um desses AFD's, com estados rotulados por  $q_0, q_1, \dots, q_n$ . Agora tome uma cadeia  $w \in \mathcal{L}$  tal que  $|w| = k \geq m = n + 1$ . Como  $\mathcal{L}$  é infinita isso pode sempre ser considerado. Seja  $q_0, q_i, q_j, \dots, q_f$  o conjunto de estados do autômato quando ele reconhece  $w$ .

Como essa cadeia tem no mínimo  $n + 1$  entradas, então pelo menos um estado deve ser repetido, e tal repetição deve começar não após o  $m$ -ésimo movimento. Portanto, a seqüência deve ter a seguinte forma

### 4.3. Identificando Linguagens Não Regulares

---

$$q_0, q_i, q_j, \dots, q_r, \dots, q_r, \dots, q_f,$$

indicando que devem existir subcadeias  $x$ ,  $y$ , e  $z$  de  $w$  tal que

$$\begin{aligned}\delta^*(q_0, x) &= q_r, \\ \delta^*(q_r, y) &= q_r, \\ \delta^*(q_r, z) &= q_f,\end{aligned}$$

com  $|xy| \leq n + 1 = m$  e  $|y| \geq 1$ . Donde segue imediatamente que

$$\delta^*(q_0, xz) = q_f, \text{ assim como } \delta^*(q_0, xy^2z) = q_f, \quad \delta^*(q_0, xy^3z) = q_f,$$

e assim por diante, completando a prova do teorema. ■

Ou seja, se  $\mathcal{L}$  for regular e infinita, então toda palavra suficientemente longa em  $\mathcal{L}$  pode ser quebrada em três partes, de tal modo que um número arbitrário de repetições da parte do meio gera necessariamente uma outra palavra em  $\mathcal{L}$ . Dizemos que a cadeia do meio é “bombeada”. mas isto não significa que em qualquer decomposição  $xyz$  da cadeia  $w$ ,  $y$  possa ser bombeada arbitrariamente, ou seja perfeitamente podem existir decomposições de  $w$  tais que o bombeamento da parte do meio gere uma cadeia não pertencente à linguagem.

Por exemplo a linguagem  $\mathcal{L} = \{a^n / n \text{ é par}\}$  é infinita e regular e portanto satisfaz o lema do bombeamento. Para ver isto basta escolher  $m = 2$ , pois qualquer cadeia  $w = a^{2k}$  para algum  $k \geq 1$  pode ser decomposta em  $x = \lambda$ ,  $y = aa$  e  $z = a^{2(k-1)}$ , ao bombear  $y$   $i$ -vezes, teremos a cadeia

$$xy^i z = (aa)^i a^{2(k-1)} = a^{2i} a^{2(k-1)} = a^{2(i+k-1)}$$

que é da forma  $a^{2k'}$  e portanto faz parte da linguagem  $\mathcal{L}$ . Observe que se tivéssemos decomposto  $w$  em  $x = \lambda$ ,  $y = a$  e  $z = a^{2k-1}$ , ao bombear  $y$  duas vezes ( $i = 2$ ), teríamos a cadeia  $xy^2z = a^2 a^{2k-1} = a^{2+2k-1} = a^{2k+1}$  que não é de comprimento par.

Enunciamos o lema do bombeamento somente para linguagens infinitas. Linguagens finitas, embora sempre regulares, não podem ser bombeadas pois o bombeamento automaticamente cria um conjunto infinito de cadeias. Assim, o teorema não vale para linguagens finitas, pois ele se tornaria vazio (poderia ser escolhido um  $m$  maior do que o maior comprimento das cadeias na linguagem, de modo que nenhuma cadeia pode ser bombeada).

**Exemplo 4.3.3** Seja a linguagem

$$\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \text{ e } \mathcal{N}_b(w) \text{ são pares}\}.$$

Esta linguagem é regular (veja exemplo 2.5). Logo, pelo lema do bombeamento deve existir um inteiro positivo  $m$  tal que todo  $w \in \mathcal{L}$ , com  $|w| \geq m$ , pode ser decomposto como  $w = xyz$ , com  $|xy| \leq m$  e  $|y| \geq 1$  satisfazendo a equação 4.1.

Seja  $m = 3$  e uma cadeia arbitrária  $w = a_1a_2\dots a_n \in \mathcal{L}$  (Portanto  $\mathcal{N}_a(w) = 2p$  e  $\mathcal{N}_b(w) = 2q$  para algum  $p, q \in \mathbb{N}$ ) tal que  $|w| = n \geq m = 3$ . Claramente, como só temos dois símbolos no alfabeto ou  $a_2 = a_1$  ou  $a_2 = a_3$ . Supondo que  $a_2 = a_1 = a$  então podemos decompor  $w$  em  $x = \lambda$ ,  $y = a_1a_2$  e  $z = a_3\dots a_n$ , bombeando  $y$   $k$  vezes, nos teremos que

$$\begin{aligned}\mathcal{N}_a(xy^kz) &= \mathcal{N}_a(x) + \mathcal{N}_a(y^k) + \mathcal{N}_a(z) \\ &= 0 + \mathcal{N}_a((aa)^k) + (\mathcal{N}_a(w) - 2) \\ &= 2k + 2p - 2 \\ &= 2(k + p - 1)\end{aligned}$$

Portanto,  $\mathcal{N}_a(xy^kz)$  é par. por outro lado, como não modificamos a quantidade de  $b$ 's, temos que  $\mathcal{N}_b(xy^kz) = \mathcal{N}_b(w) = 2q$ . Assim,  $xy^kz \in \mathcal{L}$ . Se  $a_2 = a_3 = b$  procedemos de forma similar.

Já se  $a_2 = a_3 = a$  ou se  $a_2 = a_3 = b$  então podemos decompor  $w$  em  $x = a_1$ ,  $y = a_2a_3$  e  $z = a_4\dots a_n$  e proceder de forma análoga ao caso anterior.

Portanto  $\mathcal{L}$  é uma linguagem regular.

Porém o lema do bombeamento, assim como o argumento do princípio da casa de pombos do exemplo 4.3.1, é mais usado para se mostrar que certas linguagens não são regulares. A demonstração é sempre por contradição.

**Exemplo 4.3.4** Usaremos o lema do bombeamento para mostrar que  $\mathcal{L} = \{a^n b^n / n \geq 0\}$  não é uma linguagem regular. Suponhamos que  $\mathcal{L}$  é regular, portanto o lema do bombeamento deveria valer. Não sabemos o valor de  $m$ , mas qualquer que seja ele podemos escolher  $n > m$ . Logo, a subcadeia  $y$  deve consistir inteiramente de  $a$ 's. Suponha que  $|y| = k$  com  $k \geq 1$ . Então, a cadeia obtida usando-se  $i = 0$ , na equação 4.1, é

$$a^{n-k}b^n.$$

Como  $n - k \neq n$ , então, claramente,  $a^{n-k}b^n$  não está em  $\mathcal{L}$ . Isso contradiz o lema do bombeamento e, portanto, a hipótese é falsa, isto é,  $\mathcal{L}$  não é regular.

**Exemplo 4.3.5** Seja a linguagem  $\mathcal{L}_{Pal}$  do exemplo 3.5.7. Vamos demonstrar usando o lema do bombeamento, que esta linguagem não é regular. Suponhamos que é regular, portanto o lema do bombeamento deveria valer. Como não conhecemos o valor de  $m$ , usemos um  $m$  genérico. Agora escolha a cadeia

$$a^m b a^m.$$

Claramente,  $a^m b a^m$  é um palíndromo, independente do valor de  $m$ . Pelo lema do bombeamento, podemos decompor essa cadeia em três partes ( $x$ ,  $y$  e  $z$ ) satisfazendo as condições do lema do bombeamento e bombear a subcadeia  $y$  tantas vezes quanto desejarmos, que ainda obtemos uma cadeia em  $\mathcal{L}_{Pal}$ . Para refutar este resultado devemos mostrar que qualquer que seja esta decomposição de  $a^m b a^m$  (satisfazendo as condições do lema) podemos gerar, ao bombear a subcadeia  $y$ , uma cadeia que não pertence à linguagem. Faremos estas decomposições genericamente. Como  $|xy| \leq m$  e  $|y| \geq 1$  então  $x = a^i$  com  $i \leq m$ ,  $y = a^j$  com  $1 \leq j \leq m$ ,  $i + j \leq m$

### 4.3. Identificando Linguagens Não Regulares

---

e  $z = a^{m-(i+j)}ba^m$ . Pelo lema do bombeamento, todas as cadeias da forma  $xy^kz$ , com  $k \in \mathbb{N}$  também pertencem a  $\mathcal{L}_{Pal}$ . Porém para  $k = 2$ , temos que

$$\begin{aligned} xy^2z &= a^i a^j a^j a^{m-(i+j)}ba^m \\ &= a^i a^j a^{m-(i+j)}a^j ba^m \\ &= a^m a^j ba^m \\ &= a^{m+j}ba^m \end{aligned}$$

e portanto  $xy^2z \neq a^n b^{2n} a^{n+j}$  para todo  $n \in \mathbb{N}$ . Ou seja  $xy^2z$  não é um palíndromo o que contradiz o lema do bombeamento. Logo a hipótese que  $\mathcal{L}_{Pal}$  é uma linguagem regular é falsa.

Estes exemplos sugerem que é possível reescrever o lema do bombeamento na sua forma contra-positiva para demonstrar diretamente quando uma linguagem não é regular.

**Corolário 4.3.6** Seja  $\mathcal{L}$  uma linguagem infinita. Se para qualquer inteiro positivo  $m$  existe uma cadeia  $w \in \mathcal{L}$  tal que  $|w| \geq m$  e para toda possível decomposição de  $w$  em três cadeias  $x, y$  e  $z$  ( $w = xyz$ ) com  $|xy| \leq m$  e  $|y| \geq 1$ , temos que a cadeia  $xy^i z \notin \mathcal{L}$  para algum inteiro não negativo  $i$ , então  $\mathcal{L}$  não é regular.

**Exemplo 4.3.7** Demonstremos usando o corolário acima que a linguagem

$$\mathcal{L} = \{uu \mid u \in \{a, b\}^*\}$$

não é regular.

Seja  $m$  um inteiro positivo qualquer e  $w = a^m b a^m b$ . Então,  $w = uu$  para  $u = a^m b$  e portanto  $w \in \mathcal{L}$  e  $|w| = 2m + 2 > m$ . Suponha que  $w = xyz$  com  $|xy| \leq m$  e  $|y| \geq 1$ . Claramente,  $y$  está na primeira metade e está so composto de  $a$ 's. No menor dos casos  $y = a$ . Logo se bombearmos  $y$ , digamos em três vezes, a quantidade de  $a$ 's da primeira metade aumentará pelo menos em dois, e portanto o único  $b$  que fazia parte da primeira metade passará para a segunda metade, resultando assim numa cadeia cuja metade esquerda é necessariamente diferente da metade direita e portanto não pertencerá à linguagem. Formalmente, se  $|x| = p$  e  $|y| = q$  (com  $q \geq 1$  e  $p + q \leq m$ ) então  $w = a^p a^q a^{m-(p+q)} b a^m b$ . Bombeando duas vezes  $y$  teremos a cadeia

$$\begin{aligned} xy^3z &= a^p a^{2q} a^{m-(p+q)} b a^m b \\ &= a^{m-(p+q)+p+q+q} b a^m b \\ &= a^{m+q} b a^m b \end{aligned}$$

Como  $q \geq 1$  então  $m + q > m$  e portanto  $xy^3z = a^{m+q} b a^m b \notin \mathcal{L}$ . Logo,  $\mathcal{L}$  não é uma linguagem regular.

Lamentavelmente, o lema do bombeamento estabelece apenas uma condição necessária para uma linguagem infinita ser regular. Mas isto não é suficiente para provarmos que uma linguagem qualquer seja regular ou não. Analogamente, o corolário 4.3.6, da uma condição suficiente, mas não necessária, para provarmos que uma linguagem não é regular, ou seja existem linguagens que satisfazem as condições desse corolário mas que não são regulares.

**Exemplo 4.3.8** Seja a linguagem

$$\mathcal{L} = \{uu^Rv \mid u, v \in \{a, b\}^+\}$$

Mostraremos que esta linguagem embora claramente não seja regular, não pode ser provada pelo lema do bombeamento.

Seja  $m = 4$  e  $w = a_1 \dots a_n \in \mathcal{L}$  tal que  $n \geq 4$ . Por definição da linguagem, existe um número  $1 \leq k \leq \lfloor \frac{n}{2} \rfloor - 1$  tal que para cada  $1 \leq j \leq k$ ,  $a_j = a_{2k-j+1}$ , ou seja fazendo  $u = a_1 \dots a_k$  e  $v = a_{2k+1} \dots n$  temos que  $|u| \geq 1$ ,  $|v| \geq 1$  e  $w = uu^Rv$ .

Se  $k = 1$  então é suficiente escolher  $x = a_1a_2$ ,  $y = a_3$  e  $z = a_4 \dots a_n$ , pois neste caso  $x = uu^R$  e portanto trivialmente  $xy^iz \in \mathcal{L}$ .

Se  $k \geq 2$  então considere  $x = \lambda$ ,  $y = a_1$  e  $z = a_2 \dots a_n$ . Note que se  $a_1 \dots a_{2k}$  é uma palíndromo de tamanho par, então  $a_2 \dots a_{2k-1}$  também é um palíndromo de tamanho par. Logo, para  $i = 0$ ,  $xy^iz = z = a_2 \dots a_n = u'u'^Rv$ , onde  $u' = a_2 \dots a_k$  e  $v = a_{2k+1} \dots a_n$  e por tanto está em  $\mathcal{L}$ . Se  $i = 1$  então trivialmente  $xy^iz = xyz = w \in \mathcal{L}$ . Se  $i \geq 2$  então  $xy^iz = a_1a_1a_1^{i-2}a_2 \dots a_n$ . Assim tomando  $u = a_1$  e  $v = a_1^{i-2}a_2 \dots a_n$  temos que  $|u| \geq 1$ ,  $|v| \geq 1$  e  $w = uu^Rv$ . Portanto  $xy^iz \in \mathcal{L}$ .

Desta forma seria impossível aplicar o corolário 4.3.6 para provarmos que essa linguagem não é regular.

Existem formas seguras de provarmos que linguagens não regulares são de fato não regulares. Por exemplo [LV95] apresenta uma técnica eficiente com provas simples, mas que estão baseadas na complexidade de Kolmogorov que não foi considerada neste texto.

#### 4.4. Exercícios

---

### 4.4 Exercícios

1. Prove que toda linguagem finita é regular.
2. Aplique a construção usada na prova do teorema 4.1.1 para obter o AFD que reconhece a intersecção das linguagens aceitas pelos AFD's da figura 4.5.

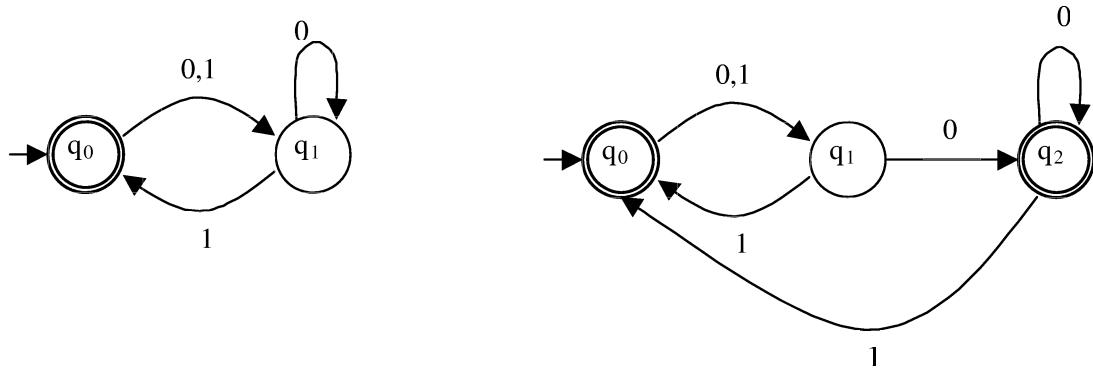


Figura 4.5: AFD do exercício 2.

3. Quais das seguintes igualdades são verdadeiras para todas as linguagens regulares e todos os homomorfismos? Justifique.
  - (a)  $h(\mathcal{L}_1 \cup \mathcal{L}_2) = h(\mathcal{L}_1) \cup h(\mathcal{L}_2)$
  - (b)  $h(\mathcal{L}_1 \cap \mathcal{L}_2) = h(\mathcal{L}_1) \cap h(\mathcal{L}_2)$
  - (c)  $h(\mathcal{L}^n) = h(\mathcal{L})^n$
  - (d)  $h(\mathcal{L}^*) = h(\mathcal{L})^*$
  - (e)  $h(\mathcal{L}^R) = h(\mathcal{L})^R$
  - (f)  $h(\overline{\mathcal{L}}) = \overline{h(\mathcal{L})}$
  - (g)  $h(\mathcal{L}_1 - \mathcal{L}_2) = h(\mathcal{L}_1) - h(\mathcal{L}_2)$
4. Na prova do teorema 4.1.8, mostre que  $h(r)$  é uma expressão regular. Então, mostre que  $h(r)$  denota  $h(\mathcal{L})$ .
5. Mostre que a família das linguagens regulares é fechada sobre união finita e intersecção finita, isto é, se  $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n$  são linguagens regulares então

$$\bigcup_{i=1}^n \mathcal{L}_i \quad \text{e} \quad \bigcap_{i=1}^n \mathcal{L}_i$$

também são.

## Capítulo 4. Propriedades das Linguagens Regulares

---

6. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  duas linguagens sobre os alfabetos  $\Sigma_1$  e  $\Sigma_2$ , respectivamente. O NEM de  $\mathcal{L}_1$  e  $\mathcal{L}_2$  é definido por

$$NEM(\mathcal{L}_1, \mathcal{L}_2) = \{w \in (\Sigma_1 \cup \Sigma_2)^* / w \notin \mathcal{L}_1 \text{ e } w \notin \mathcal{L}_2\}.$$

Mostre que a família das linguagens regulares é fechada sobre NEM.

7. Mostre que as linguagens regulares são fechadas sobre a união disjunta, onde a união disjunta de dois conjuntos (linguagens)  $\mathcal{L}_1$  e  $\mathcal{L}_2$  é definida por

$$\mathcal{L}_1 \uplus \mathcal{L}_2 = \{w0 / w \in \mathcal{L}_1\} \cup \{w1 / w \in \mathcal{L}_2\}$$

8. Seja  $\Sigma = \{a, b\}$  e  $\widehat{\cdot}: \Sigma^* \rightarrow \Sigma^*$  a operação definida recursivamente a seguir:

- $\widehat{\lambda} = a$
- $\widehat{w\widehat{a}} = b\widehat{w}$
- $\widehat{w\widehat{b}} = a\widehat{w}$

Seja  $\mathcal{L}$  uma linguagem regular sobre  $\Sigma$ . Mostre que  $\widehat{\mathcal{L}} = \{\widehat{w} / w \in \Sigma^*\}$  também é regular.

9. Seja  $\Sigma = \{a, b\}$  e  $\widehat{\cdot}: \Sigma^* \rightarrow \Sigma^*$  a operação definida a seguir:

$$\widehat{\mathcal{L}} = \{wa / wb \in \mathcal{L}\} \cup \{wb / wa \in \mathcal{L}\}$$

Mostre que a classe das linguagens regulares é fechada sobre esta operação.

10. Seja  $\mathcal{L}$  uma linguagem sobre um alfabeto  $\Sigma$ . Conforme visto no capítulo 1, a linguagem de **sufixos** de  $\mathcal{L}$ , denotada por  $\mathcal{L}^S$ , é definida como

$$\mathcal{L}^S = \{w \in \Sigma^* / vw \in \mathcal{L} \text{ para algum } v \in \Sigma^*\}.$$

Demonstre que a classe das linguagens regulares é fechada sobre sufixos.

11. Seja  $\mathcal{L}$  uma linguagem sobre um alfabeto  $\Sigma$ . Defina

- (a)  $L^\ominus = \{w / w \notin L \text{ e } |w| \neq 100\}$
- (b)  $\mathcal{L}^T = \{w \in \Sigma^* / aw \in \mathcal{L} \text{ para algum } v \in \Sigma^*\}$ . Observe que claramente,  $\mathcal{L}^T \subseteq \mathcal{L}^S$ .

Demonstre que a classe das linguagens regulares é fechada sobre estes operadores.

12. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  duas linguagens, defina o operador  $\oplus$  por

$$\mathcal{L}_1 \oplus \mathcal{L}_2 = \{w \in \mathcal{L}_1 / w \notin \mathcal{L}_2\} \cup \{w \in \mathcal{L}_2 / w \notin \mathcal{L}_1\}$$

Mostre que as linguagens regulares são fechadas sobre a operação  $\oplus$  entre linguagens.

#### 4.4. Exercícios

---

	<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i>	<i>a</i>	<i>a</i>	<i>c</i>
<i>b</i>	<i>c</i>	<i>a</i>	<i>b</i>
<i>c</i>	<i>b</i>	<i>c</i>	<i>a</i>

Tabela 4.1: Tabela multiplicativa não associativa nem comutativa.

13. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  linguagens sobre alfabetos  $\Sigma_1$  e  $\Sigma_2$ , respectivamente. Defina o operador  $\wedge_2$  por

$$\mathcal{L}_1 \wedge_2 \mathcal{L}_2 = \{uv / u \in \mathcal{L}_1 \cup \mathcal{L}_2 \text{ e } v \in (\Sigma_1 \cup \Sigma_2)^*\}$$

Mostre que as linguagens regulares são fechadas sobre a operação  $\wedge_2$  entre linguagens.

14. Seja  $\Sigma = \{a_0, \dots, a_n\}$ . Defina o complemento  $n$  de  $w \in \Sigma^*$ , denotado por  $w^c$ , como sendo  $\lambda^c = \lambda$  e  $(wa_i)^c = w^ca_{n-i}$

Por exemplo se  $\Sigma = \{a, b, c, d\}$  e considerando  $a_0 = a$ ,  $a_1 = b$ ,  $a_2 = c$  e  $a_3 = d$ , então  $aabcd^c = ddcba$ .

Se  $\mathcal{L}$  é uma linguagem regular sobre um alfabeto  $\Sigma = \{a_0, \dots, a_n\}$ , demonstre que a linguagem

$$\mathcal{L}^c = \{w^c / w \in \mathcal{L}\},$$

também é regular.

15. Mostre que existe um algoritmo para determinar se  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , para qualquer linguagem regular  $\mathcal{L}_1$  e  $\mathcal{L}_2$ .
16. Mostre que existe um algoritmo para determinar se a intersecção e união de duas linguagens regulares é finita, vazia ou infinita.
17. Seja a tabela 4.1. A primeira coluna representa os rótulos de cada fila na tabela enquanto a primeira fila representa os rótulos de cada coluna na tabela. Para cada  $x, y \in \Sigma = \{a, b, c\}$  denote por  $T(x, y)$  o conteúdo da tabela para a posição  $(x, y)$ . Seja a seguinte função  $A : \Sigma^+ \rightarrow \Sigma$  definida por

- $A(a) = a$ , para cada  $a \in \Sigma$  e
- $A(wa) = T(A(w), a)$ , para cada  $w \in \Sigma^+$  e  $a \in \Sigma$ .

Demonstre que a seguinte linguagem é regular:

$$\mathcal{L} = \{w \in \Sigma^+ / A(w) = A(w^R)\}$$

18. Seja  $\Sigma = \{0, 1\}$ . Mostre que é possível aplicar o lema do bombeamento para as seguintes linguagens regulares.

- (a)  $\mathcal{L} = \{w \in \Sigma^* / 0110 \text{ é um prefixo de } w\}$
- (b)  $\mathcal{L} = \{w \in \Sigma^* / 0110 \text{ é um sufixo de } w\}$
- (c)  $\mathcal{L} = \{w \in \Sigma^* / w = u111v \text{ para algum } u, v \in \Sigma^*\}$
- (d)  $\mathcal{L} = \{w \in \Sigma^* / \exists u, v \in \Sigma^* \text{ tal que } w = u111v\}$

19. Mostre que as linguagens sobre  $\Sigma = \{a, b\}$ , definidas a seguir, não são regulares.

- (a)  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}$
- (b)  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \neq \mathcal{N}_b(w)\}$
- (c)  $\mathcal{L} = \{w \in \Sigma^* / u = v \text{ para algum prefixo } u \neq \lambda \text{ e sufixo } v \text{ de } w\}.$
- (d)  $\mathcal{L} = \{a^m b^n / m > n\}$
- (e)  $\mathcal{L} = \{a^m b^n / m \neq n\}$
- (f)  $\mathcal{L} = \{a^n b a^{n+1} / n \geq 1\}$
- (g)  $\mathcal{L} = \{a^m b^n / 1 \leq m \leq n \leq 2m\}$
- (h)  $\mathcal{L} = \{a^{m+1} b^{n+1} / 2 \leq n \leq m \leq 3n\}$

onde  $\mathcal{N}_a(w)$  é o número de  $a$ 's que figuram em  $w$

20. Seja  $\Sigma = \{a, b\}$  e  $\widehat{\cdot} : \Sigma^* \rightarrow \Sigma^*$  a operação definida recursivamente a seguir:

- $\widehat{\lambda} = \lambda$
- $\widehat{wa} = \widehat{w}b$
- $\widehat{wb} = \widehat{w}a$

Mostre usando o lema do bombeamento que as linguagens

- (a)  $\mathcal{L} = \{ww / w \in \Sigma^*\}$
- (b)  $\mathcal{L} = \{w \in \Sigma^* / w = \widehat{w}^R\}$

não são linguagens regulares.

21. Seja  $\Sigma = \{a, b\}$  e  $\widehat{\cdot} : \Sigma^* \rightarrow \Sigma^*$  a operação definida recursivamente a seguir:

- $\widehat{\lambda} = \lambda$
- $\widehat{wa} = \widehat{w}aa$
- $\widehat{wb} = \widehat{w}bb$

Mostre usando o lema do bombeamento que a linguagem  $\mathcal{L} = \{ww / w \in \Sigma^*\}$  não é regular.

22. Seja  $\mathcal{L}$  uma linguagem sobre um alfabeto  $\Sigma$  qualquer. Defina  $P(\mathcal{L})$ , recursivamente por

- $P(\lambda) = \lambda$
- $P(wa) = aP(w)aa$  para cada  $a \in \Sigma$  e  $w \in \Sigma^*$ .

Mostre usando o lema do bombeamento que a linguagem  $\mathcal{L} = \{P(w) / w \in \Sigma^*\}$  não é regular.

#### **4.4. Exercícios**

---

## Capítulo 5

# Gramáticas Livres do Contexto

Vimos, anteriormente, que nem todas as linguagens são regulares. Algumas dessas linguagens não regulares também não são lineares. Observe que também é possível mostrar quando uma linguagem não é linear a través de lemas do bombeamento específicos para essa classe de linguagens (veja [Lin90]). A relevância dessa limitação para linguagens de programação fica evidente se re-interpretarmos alguns dos exemplos de linguagens não lineares, e portanto não regulares. Por exemplo, a linguagem  $\mathcal{L} = \{w \in \{(,)\}^* / N_{\mathcal{L}}(w) = N_{\mathcal{L}}(u) \text{ e para todo prefixo } u \text{ de } w, N_{\mathcal{L}}(u) \geq N_{\mathcal{L}}(u)\}$  não é uma linguagem linear. Exemplos de cadeias que pertencem a esta linguagem são: ((())) e ((())())() enquanto as cadeias () e ((())()) não pertencem a  $\mathcal{L}$ . Assim, esta linguagem descreve seqüências corretas de parênteses. Isto indica que algumas propriedades das linguagens de programação requerem algo além das linguagens lineares. Isto nos leva a considerar uma classe mais ampla de linguagens e gramáticas chamadas *livres do contexto*.

### 5.1 Gramáticas Livres do Contexto

As produções numa gramática regular são restritivas em dois pontos: o lado esquerdo da produção deve conter uma única variável, enquanto o lado direito tem uma forma especial. Para criar uma gramática “mais poderosa”, devemos relaxar algumas dessas condições. Manteremos a restrição sobre o lado esquerdo, mas deixaremos sem restrição o lado direito. Obteremos assim, uma gramática livre do contexto.

**Definição 5.1.1** Uma gramática  $G = \langle V, T, S, P \rangle$  diz-se **livre do contexto** se todas as produções em  $P$  tem a forma

$$A \longrightarrow x,$$

onde  $A \in V$  e  $x \in (V \cup T)^*$ .

Uma linguagem  $\mathcal{L}$  diz-se **livre do contexto** se existe uma gramática livre do contexto  $G$  tal que  $\mathcal{L} = L(G)$ .

## 5.1. Gramáticas Livres do Contexto

---

Toda linguagem regular é livre do contexto, pois toda gramática regular é também livre do contexto. Mas, a linguagem  $\{a^n b^n / n \geq 0\}$  não é regular, como foi visto no exemplo 4.3.1 e 4.3.4 usando o princípio da casa dos pombos e o lema do bombeamento, respectivamente. Acontece que, pelo exemplo 1.2.13, ela é livre do contexto. Logo o conjunto das linguagens regulares é um subconjunto próprio da família das linguagens livres do contexto.

As gramáticas livres do contexto têm esse nome devido de que o lado esquerdo de qualquer produção é composta necessariamente por uma única variável e que por tanto variáveis ocorrendo em formas sentenciais podem ser substituídas (pelo lado direito de uma produção) sem se preocupar do que acompanhar ela (seu contexto).

**Exemplo 5.1.2** A gramática  $G = \langle \{S\}, \{a, b\}, S, P \rangle$ , com produções

$$S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda,$$

é livre do contexto. Uma derivação típica nessa gramática é

$$S \implies aSa \implies aaSaa \implies aabSbaa \implies aabbbaa$$

Isto torna claro que

$$L(G) = \{w \in \{a, b\}^* / w = w^R\}.$$

Essa linguagem é livre do contexto, mas não é regular, conforme o exercício 4.3.5 do capítulo 3.

**Exemplo 5.1.3** A gramática  $G = \langle \{S, A, B\}, \{a, b\}, S, P \rangle$ , onde  $P$  é

$$\begin{array}{l} S \longrightarrow abB \\ A \longrightarrow aaBb \mid \lambda \\ B \longrightarrow bbAa \end{array}$$

é livre do contexto.  $L(G) = \{ab(bbba)^n bba(ba)^n / n \geq 0\}$ .

**Exemplo 5.1.4** A linguagem

$$\mathcal{L} = \{a^m b^n / m \neq n\}$$

é livre do contexto. Para mostrar isso precisamos produzir uma gramática livre do contexto que gere a linguagem  $\mathcal{L}$ . Observe que  $m \neq n$  se e somente se  $m < n$  ou  $n < m$ . Tomemos o caso  $n < m$ . Primeiramente, geramos uma cadeia com a mesma quantidade de  $a$ 's e de  $b$ 's, então adicionamos  $a$ 's extras à esquerda. Isso é feito com as seguintes produções

$$\begin{array}{l} S_1 \longrightarrow aS_1b \mid A \\ A \longrightarrow aA \mid a \end{array}$$

Usando um raciocínio análogo obtemos as seguintes produções para o caso  $m < n$

$$\begin{aligned} S_2 &\longrightarrow aS_2b \mid B \\ B &\longrightarrow Bb \mid b \end{aligned}$$

A junção desses dois conjuntos de produções é feita adicionando as produções

$$S \longrightarrow S_1 \mid S_2.$$

Assim, a gramática  $G = \langle \{S, S_1, S_2, A, B, C\}, \{a, b\}, S, P \rangle$ , onde  $P$  é o conjunto de produções acima gera a linguagem  $\mathcal{L}$ . Claro que também poderíamos realizar uma junção do raciocínio usado para os casos  $n > m$  e  $n < m$ , resultando numa gramática mais enxuta que  $G$ , cujas produções são:

$$\begin{aligned} S &\longrightarrow aSb \mid A \mid B \\ A &\longrightarrow aA \mid a \\ B &\longrightarrow Bb \mid b \end{aligned}$$

### 5.1.1 Derivações Mais à Esquerda e Mais à Direita

Em gramáticas livres do contexto que não são lineares, uma derivação pode envolver formas sentenciais com mais de uma variável. Nesse caso, temos uma escolha na ordem em que as variáveis são trocadas. Tome, por exemplo, a gramática  $G = \langle \{S, A, B\}, \{a, b\}, S, P \rangle$  com produções

1.  $S \longrightarrow AB$
2.  $A \longrightarrow aaA$
3.  $A \longrightarrow \lambda$
4.  $B \longrightarrow Bb$
5.  $B \longrightarrow \lambda$

É fácil ver que esta gramática gera a linguagem

$$\mathcal{L} = \{a^{2n}b^m / n \geq 0, m \geq 0\}.$$

Considere, agora, as duas derivações seguintes:

$$S \xrightarrow{1} AB \xrightarrow{2} aaAB \xrightarrow{3} aaB \xrightarrow{4} aaBb \xrightarrow{5} aab$$

$$S \xrightarrow{1} AB \xrightarrow{4} ABb \xrightarrow{2} aaABb \xrightarrow{5} aaAb \xrightarrow{3} aab.$$

Para mostrar que derivação foi usada enumeramos as produções e escrevemos o correspondente número em cima do símbolo  $\Rightarrow$ . Isso nos permite ver que as duas derivações não somente geram a mesma cadeia, mas que as mesmas produções são usadas. A diferença está somente na ordem em que as produções são aplicadas. Para remover fatores irrelevantes, freqüentemente pedimos que as variáveis sejam trocadas numa ordem específica.

## 5.1. Gramáticas Livres do Contexto

---

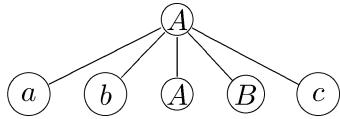


Figura 5.1: Parte da árvore de derivação para a produção  $A \rightarrow abABc$ .

**Definição 5.1.5** Uma derivação diz-se **mais à esquerda**, se em cada etapa a variável mais à esquerda é trocada na forma sentencial. Se em cada etapa a variável mais à direita é trocada na forma sentencial, dizemos que a derivação é **mais à direita**.

**Exemplo 5.1.6** Considere a gramática com produções

$$\begin{aligned} S &\longrightarrow aAB \\ A &\longrightarrow bBb \\ B &\longrightarrow A \mid \lambda \end{aligned}$$

Então

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbb \Rightarrow abbbb$$

é uma derivação mais à esquerda da cadeia abbbb. Uma derivação mais à direita para a mesma cadeia é

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb.$$

### 5.1.2 Árvores de Derivação

Uma segunda maneira de mostrar derivações, independente da ordem em que as produções são usadas, é através de **árvores de derivação**. Uma árvore de derivação é uma árvore ordenada na qual os nós são rotulados com os lados esquerdos das produções e no qual os filhos de cada nó representam seus correspondentes lados direitos.

Por exemplo, a figura 5.1 mostra parte de uma árvore de derivação representando a produção

$$A \longrightarrow abABc$$

Numa árvore de derivação um nó rotulado com uma variável ocorrendo no lado esquerdo da produção tem filhos consistindo dos símbolos no lado direito daquela produção e terminando em folhas que são terminais. Uma árvore de derivação mostra como cada variável é trocada na derivação. A definição que segue torna precisa essa noção.

**Definição 5.1.7** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Uma **árvore ordenada** é uma árvore de derivação para  $G$  se ela tem as seguintes propriedades

1. A raíz tem rótulo  $S$ .

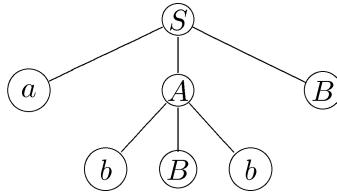


Figura 5.2: Uma árvore de derivação parcial.

2. Toda folha tem um rótulo de  $T \cup \{\lambda\}$ .
3. Todo vértice interior (um vértice que não é uma folha) tem um rótulo de  $V$ .
4. Se um vértice tem rótulo  $A \in V$ , e seus filhos são rotulados (da esquerda para direita)  $a_1, a_2, \dots, a_n$ , então  $P$  deve conter uma produção da forma

$$A \longrightarrow a_1 a_2 \dots a_n.$$

5. Toda folha rotulada  $\lambda$  é filho único.

Uma árvore ordenada satisfazendo as propriedades 3,4 e 5, na qual 1 não se verifica necessariamente e na qual a propriedade 2 é trocada por

2.a. Toda folha tem um rótulo de  $V \cup T \cup \{\lambda\}$ ,

diz-se **árvore de derivação parcial**.

A cadeia de símbolos obtida em se lendo, da esquerda para a direita, omitindo qualquer  $\lambda$  encontrado, diz-se **gerada** da árvore.

**Exemplo 5.1.8** Considere a gramática  $G$ , com produções

$$\begin{aligned} S &\longrightarrow aAB \\ A &\longrightarrow bBb \\ B &\longrightarrow A \mid \lambda \end{aligned}$$

A árvore na figura 5.2 é uma árvore de derivação parcial para  $G$ , enquanto a árvore na figura 5.3 é uma árvore de derivação. A cadeia  $abBbB$ , gerada pela primeira árvore, é uma forma sentencial de  $G$ . A cadeia gerada pela segunda árvore,  $abbb$ , é uma sentença de  $L(G)$ .

### 5.1.3 Relação entre Formas Sentenciais e Árvores de Derivação

As árvores de derivação dão uma descrição explícita e comprehensível das derivações. Assim como o grafo de transição para autômatos finitos, elas são úteis nos argumentos.

**Teorema 5.1.9** Seja  $G = \langle V, S, T, P \rangle$  uma gramática livre do contexto. Então para todo  $w \in L(G)$  existe uma árvore de derivação de  $G$  cuja cadeia gerada é  $w$ . Inversamente, a cadeia gerada por qualquer árvore de derivação de  $G$  está em  $L(G)$ . Além disso, se  $A_G$  é qualquer árvore de derivação parcial para  $G$  cuja raiz é rotulada  $S$ , então  $A_G$  gera uma forma sentencial de  $G$ .

## 5.1. Gramáticas Livres do Contexto

---

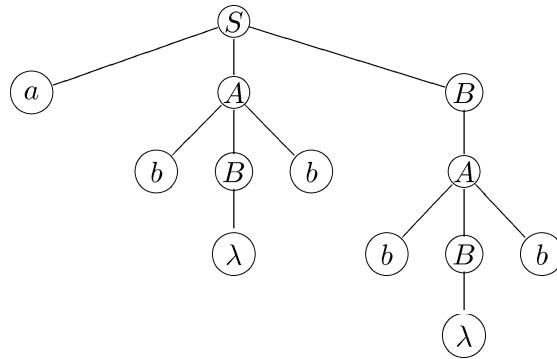


Figura 5.3: Uma árvore de derivação.

**DEMONSTRAÇÃO:** Primeiramente mostraremos que para toda forma sentencial de  $L(G)$  existe uma árvore de derivação parcial. Faremos isso por indução no número de etapas da derivação. Como base, observemos que a afirmação é verdadeira para toda forma sentencial derivada em uma etapa. Como  $S \Rightarrow u$ , existe uma produção  $S \rightarrow u$ . Isso segue imediatamente da definição 5.1.7.

Suponhamos que para toda forma sentencial pode ser derivada em  $n$  etapas, existe uma árvore de derivação parcial correspondente. Agora, qualquer  $w$  pode ser derivado em  $n+1$  etapas deve ser tal que

$$S \xrightarrow{*} xAy, \text{ com } x, y \in (V \cup T)^* \text{ e } A \in V, \text{ em } n \text{ etapas, e}$$

$$xAy \Rightarrow xa_1a_2 \cdots a_my = w \text{ com } a_i \in V \cup T.$$

Como, pela hipótese da indução, existe uma árvore de derivação parcial que gera  $xAy$ , e como a gramática deve ter a produção  $A \rightarrow a_1a_2 \cdots a_m$ , vemos que expandindo a folha rotulada  $A$ , obtemos uma árvore de derivação parcial que gera  $xa_1a_2 \cdots a_my = w$ . Por indução, o resultado é verdade para toda forma sentencial.

Usando um argumento análogo, podemos mostrar que toda árvore de derivação parcial representa alguma forma sentencial.

Como toda árvore de derivação é também uma árvore de derivação parcial cujas folhas são terminais, segue que toda sentença de  $L(G)$  é gerada por alguma árvore de derivação de  $G$  e a cadeia gerada por qualquer árvore de derivação está em  $L(G)$ . ■

As árvores de derivação mostram quais produções são usadas para se obter uma sentença, mas não fornecem a ordem de suas aplicações. As árvores de derivação são capazes de representar qualquer derivação, refletindo o fato de que esta ordem é irrelevante. Esta observação nos permite fechar o buraco da discussão anterior. Por definição, qualquer  $w \in L(G)$  tem uma derivação, mas não alegamos que ela também tem uma derivação mais à esquerda e uma mais à direita. Entretanto, uma vez que tenhamos uma árvore de derivação, podemos sempre obter uma derivação mais à esquerda, pensando na árvore como tendo sido construída de tal modo que a variável mais à esquerda da árvore foi sempre expandida primeiro. A menos de detalhes temos que todo  $w \in L(G)$  tem uma derivação mais à esquerda e uma mais à direita.

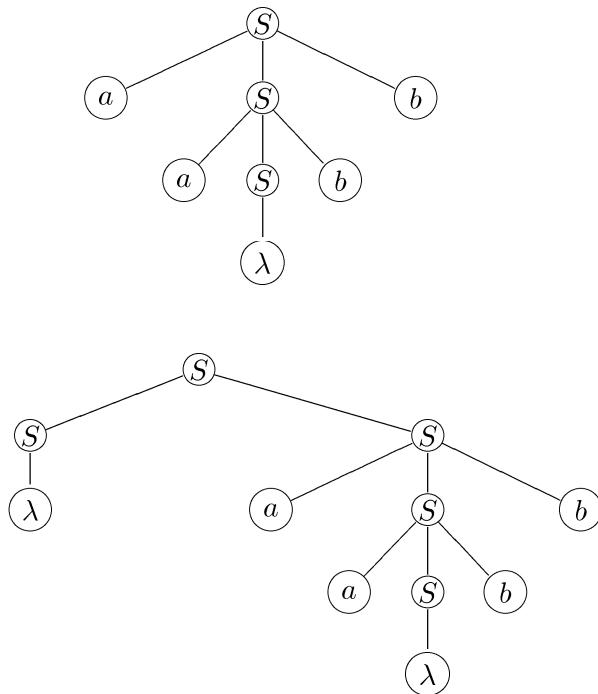


Figura 5.4: Exemplo de árvores de derivação diferentes para uma mesma cadeia.

## 5.2 Ambigüidade em Gramáticas e Linguagens

Assim, a menos de detalhes temos que todo  $w \in L(G)$  tem uma derivação mais à esquerda e uma mais à direita. Dizemos “uma” árvore de derivação em vez de “a” árvore de derivação devido ao fato de que existem muitas árvores de derivação, em princípio. Esta situação é designada por *ambigüidade*.

**Definição 5.2.1** Uma gramática livre do contexto,  $G = \langle V, T, S, P \rangle$  diz-se **ambígua** se existe algum  $w \in L(G)$  que tem no mínimo duas árvores de derivação distintas. Alternativamente, ambigüidade implica a existência de duas ou mais derivações à esquerda ou à direita.

**Exemplo 5.2.2** A gramática com produções

$$S \longrightarrow aSb \mid SS \mid \lambda,$$

é ambígua. A sentença  $aabb$  tem duas árvores de derivação como mostrada na figura 5.4.

A ambigüidade é uma característica comum em linguagens naturais, onde elas são toleradas e manipuladas de muitos modos. Em linguagens de programação, onde deve existir somente uma interpretação de cada instrução, a ambigüidade deve ser evitada quando possível. Em geral, se obtém isto re-escrevendo a gramática numa forma não-ambígua, equivalente.

## 5.2. Ambigüidade em Gramáticas e Linguagens

---

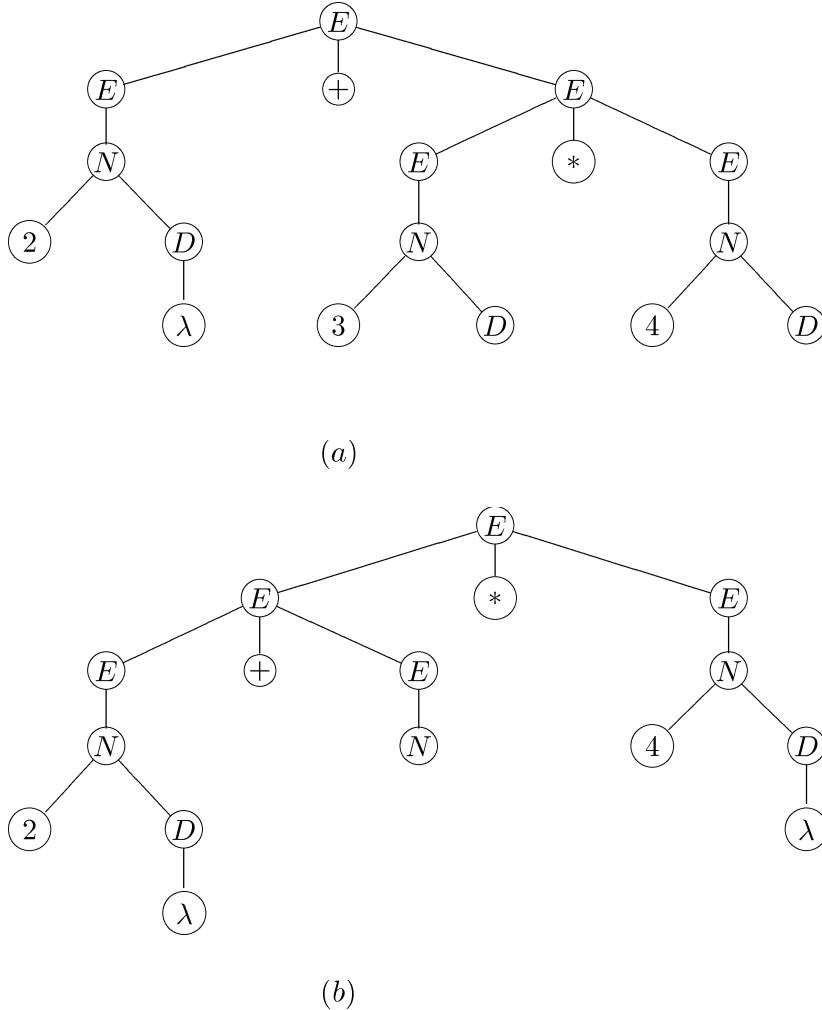


Figura 5.5: Exemplo de árvores de derivação diferentes para uma mesma cadeia.

**Exemplo 5.2.3** Considere a gramática livre do contexto  $G = \langle V, T, E, P \rangle$ , onde  $V = \{E, N, D\}$ ,  $T = \{0, 1, \dots, 9, +, *, (, )\}$ , e  $P$  é o conjunto de produções

$$\begin{aligned} E &\longrightarrow E+E \mid E*E \mid (E) \mid N \\ N &\longrightarrow 0 \mid 1D \mid 2D \mid \dots \mid 9D \\ D &\longrightarrow 0D \mid 1D \mid \dots \mid 9D \mid \lambda \end{aligned}$$

As cadeias  $(2+3)*4$  e  $2+3*4$  estão em  $L(G)$ . É fácil ver que esta gramática gera um subconjunto restrito de expressões aritméticas para linguagens de programação tipo Fortran ou Pascal. A gramática é ambígua. Por exemplo, a cadeia  $2+3*4$  tem duas árvores de derivação diferentes, como mostrada na figura 5.5.

Um modo de resolver a ambigüidade é, como está nos manuais de linguagens de programação, associar regras de precedência aos operadores  $+$  e  $*$ . Como  $*$ , normalmente, tem precedência mais alta que  $+$ , tomariamos a figura 5.3.(a) como a análise correta, pois ela indica que a

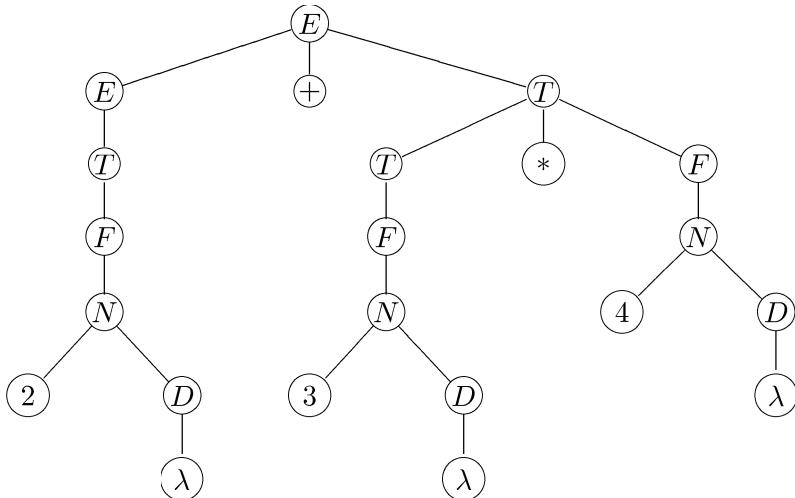


Figura 5.6: Exemplo de árvore de derivação numa gramática, não ambígua, equivalente à gramática ambígua do exemplo anterior.

subexpressão  $3 * 4$  é avaliada antes de tomar a adição. Entretanto, essa convenção está completamente fora da gramática. É melhor reescrever a gramática tal que só exista uma análise possível, aquela em que a multiplicação é feita antes de uma adição.

**Exemplo 5.2.4** Para reescrever a gramática do exemplo 5.2.3, anterior, introduzimos novas variáveis, tomando  $V$  como  $\{E, T, F, N, D\}$  e trocando as produções por

$$\begin{aligned} E &\longrightarrow T \mid E + T \\ T &\longrightarrow F \mid T * F \\ F &\longrightarrow N \mid (E) \\ N &\longrightarrow 0 \mid 1D \mid 2D \mid \dots \mid 9D \\ D &\longrightarrow 0D \mid 1D \mid \dots \mid 9D \mid \lambda \end{aligned}$$

Uma árvore de derivação da sentença  $2 + 3 * 4$  é mostrada na figura 5.6. Não existe outra árvore de derivação para essa cadeia: a gramática não é ambígua e é equivalente à gramática do exemplo 5.2.3. Não é muito difícil justificar esse argumento nesse exemplo específico, mas, em geral, as questões de se uma dada gramática livre do contexto é ambígua ou não e se duas gramáticas livres do contexto são equivalentes ou não, são muito difíceis de responder. De fato, veremos adiante que não existe nenhum algoritmo geral com o qual essas questões possam ser resolvidas.

No exemplo a seguir a ambigüidade veio da gramática no sentido de que ela poderia ser removida achando uma gramática não-ambígua equivalente. Em alguns casos, entretanto, isto não é possível porque a ambigüidade está na linguagem.

**Definição 5.2.5** Se  $\mathcal{L}$  é uma linguagem livre do contexto para o qual existe uma gramática não ambígua, então  $\mathcal{L}$  diz-se **não-ambígua**. Se toda gramática que gera  $\mathcal{L}$  é ambígua,  $\mathcal{L}$  diz-se **inherentemente ambígua**.

### 5.3. Algoritmos de Análise de Pertinência

---

**Exemplo 5.2.6** A linguagem

$$\mathcal{L} = \{a^m b^m c^n\} \cup \{a^m b^n c^n\},$$

com  $m$  e  $n$  não negativos, é uma linguagem livre do contexto inherentemente ambígua.

$\mathcal{L}$  é livre do contexto. De fato, observe que

$$\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2,$$

onde  $\mathcal{L}_1$  é gerada por

$$\begin{aligned} S_1 &\longrightarrow DC \\ D &\longrightarrow aDb \mid \lambda \\ C &\longrightarrow cC \mid \lambda \end{aligned}$$

e  $\mathcal{L}_2$  é dada por uma gramática análoga com símbolo de início  $S_2$  e produções

$$\begin{aligned} S_2 &\longrightarrow AE \\ A &\longrightarrow aA \mid \lambda \\ E &\longrightarrow bEc \mid \lambda. \end{aligned}$$

Então  $\mathcal{L}$  é gerada pela combinação dessas duas gramáticas com a produção adicional

$$S \longrightarrow S_1 \mid S_2.$$

A gramática é ambígua porque as cadeias  $a^n b^n c^n$  pertence a  $\mathcal{L}_1$  e a  $\mathcal{L}_2$ . Portanto, tem duas derivações distintas, uma começando com  $S \Rightarrow S_1$ , e outra com  $S \Rightarrow S_2$ . Disto não segue que  $\mathcal{L}$  é inherentemente ambígua pois pode existir outra gramática não-ambígua equivalente a essa. Mas de alguma maneira  $\mathcal{L}_1$  e  $\mathcal{L}_2$  tem exigências conflitantes, a primeira pondo uma restrição no número de  $a$ 's e  $b$ 's, enquanto a segunda faz o mesmo para  $b$ 's e  $c$ 's. Algumas poucas tentativas nos convencerá da impossibilidade de combinar essas exigências num conjunto único de produções que cubra unicamente o caso  $n = m$ . Um argumento rigoroso, entretanto, e muito técnico pode ser encontrado em [Har78].

## 5.3 Algoritmos de Análise de Pertinência

Até aqui temos nos concentrado no aspecto gerativo das gramáticas livres do contexto. Dada uma gramática  $G$ , estudamos o conjunto de cadeias que podem ser derivadas usando  $G$ . Porém, em caso de aplicações práticas estaremos também interessados com o lado analítico da gramática: dado uma cadeia de terminais  $w$ , queremos saber se  $w \in L(G)$  ou não. Se for o caso, poderemos estar interessados em achar uma derivação de  $w$ . Uma algoritmo que pode nos dizer se  $w \in L(G)$  ou não é chamado **algoritmo de pertinência**. O termo **análise** descreve o modo de achar uma seqüência de produções pela qual é derivada uma cadeia  $w \in L(G)$ .

### 5.3.1 Método de Análise de Pesquisa Exaustiva

Dada uma cadeia  $w$  em  $L(G)$  podemos analisá-la de maneira óbvia: sistematicamente construímos todas as possíveis derivações, por exemplo as mais à esquerda, e verificamos se alguma das formas sentenciais geradas casa com  $w$ . Especificamente começamos verificando todas as produções da forma

$$S \longrightarrow x$$

achando todos os  $x$ 's em  $(V \cup T)^*$  que podem ser derivados de  $S$  numa etapa. Se nenhum desses resultados é  $w$ , vamos para a segunda etapa, na qual aplicamos todas as produções possíveis à variável mais à esquerda de todo  $x$ . Isso nos dá um conjunto de formas sentenciais, algumas delas possivelmente levando para  $w$ . Em cada etapa subsequente tomamos novamente as variáveis mais à esquerda e aplicamos todas as produções possíveis. Pode ser que alguma dessas formas sentenciais possam ser excluídas porque  $w$  nunca seria derivada delas. Mas, em geral, temos em cada etapa o conjunto de todas as formas sentenciais possíveis. Após a primeira etapa, temos formas sentenciais que podem ser derivadas aplicando-se uma única produção, após a segunda etapa temos formas sentenciais que podem ser derivadas aplicando duas produções, e assim por diante. Se  $w \in L(G)$ , devemos ter uma derivação mais à esquerda de comprimento finito. Portanto, o método nos dará mais cedo ou mais tarde uma derivação mais à esquerda de  $w$ .

Como referência, chamaremos isto o método de **análise de pesquisa exaustiva**. Ele é uma espécie de análise *top-down* o qual podemos ver como construção de uma árvore de derivação da raiz para baixo.

**Exemplo 5.3.1** Considere a gramática

$$S \longrightarrow SS \mid aSb \mid bSa \mid \lambda$$

e a cadeia  $w = aabb$ . A etapa um nos dá:

1.  $S \implies SS$
2.  $S \implies aSb$
3.  $S \implies bSa$
4.  $S \implies \lambda$

As duas últimas dessas podem ser removidas das próximas considerações, por razões óbvias.

A etapa dois gera as seguintes formas sentenciais

1.  $S \implies SS \implies SSS$
2.  $S \implies SS \implies aSbS$

### 5.3. Algoritmos de Análise de Pertinência

---

3.  $S \Rightarrow SS \Rightarrow bSaS$
4.  $S \Rightarrow SS \Rightarrow S$
5.  $S \Rightarrow aSb \Rightarrow aSSb$
6.  $S \Rightarrow aSb \Rightarrow aaSbb$
7.  $S \Rightarrow aSb \Rightarrow abSab$
8.  $S \Rightarrow aSb \Rightarrow ab$

Novamente, várias dessas derivações (3, 7 e 8) podem ser desconsideradas. Na próxima etapa achamos, a partir da derivação 6, a cadeia procurada. Assim, a derivação mais à esquerda de  $w$  seria

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb.$$

A análise de pesquisa exaustiva tem sérios problemas. O mais óbvio deles é que ele é tedioso. Ele não pode ser usado quando se pede eficiência. Mas mesmo quando a eficiência for secundária, existe uma objeção fundamental: enquanto o método sempre analisa uma cadeia  $w \in L(G)$ , é possível que ele nunca termine para uma cadeia que não está em  $L(G)$ . Este é o caso se no exemplo anterior analisarmos a cadeia  $w = abb$ . O método irá produzindo formas sentenciais indefinidamente a menos que coloquemos nele uma condição de parada.

O problema da não terminação do método de pesquisa exaustiva é relativamente fácil de superar se restringirmos a forma da gramática. Se examinarmos o exemplo 5.3.1, veremos que a dificuldade vem da produção  $S \rightarrow \lambda$ . Esta produção pode ser usada para decrescer o tamanho das formas sentenciais sucessivas de modo que não possamos dizer facilmente quando parar. Se não tivéssemos tal produção, então teríamos muito menos dificuldade. De fato, existem dois tipos de produções que queremos deixar de fora. Aquelas da forma  $A \rightarrow B$  e  $A \rightarrow \lambda$ . Como veremos mais adiante, esta restrição não afeta, significativamente, o poder da gramática resultante.

#### Exemplo 5.3.2 A gramática

$$S \rightarrow SS \mid aSb \mid bSa \mid ab \mid ba$$

satisfaz as nossas exigências. Ela gera a linguagem do exemplo 5.3.1 sem a cadeia vazia.

Dado qualquer  $w \in \{a, b\}^+$ , o método da análise de pesquisa exaustiva sempre terminará em não mais do que  $|w|$  etapas. Isto é claro devido ao fato de que o comprimento da forma sentencial cresce em, no mínimo, um símbolo em cada etapa. Assim, após  $|w|$  etapas teremos produzido a cadeia  $w$  em análise ou saberemos que  $w \notin L(G)$ .

A idéia, nesse exemplo, pode ser generalizada e tornar-se um teorema para linguagens livres do contexto, em geral.

**Teorema 5.3.3** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto sem produções da forma

$$A \longrightarrow \lambda \text{ nem da forma } A \longrightarrow B$$

onde  $A, B \in V$ . Então o método de análise de pesquisa exaustiva pode se tornar um algoritmo tal que para cada  $w \in T^*$ , ou produz uma análise de  $w$  ou nos diz que nenhuma análise é possível.

**DEMONSTRAÇÃO:** Para cada forma sentencial, considere seu comprimento e o número de símbolos terminais. Cada etapa na derivação faz crescer a forma sentencial, no mínimo, em um símbolo (terminal ou variável). Mas, por não termos as produções do tipo  $A \longrightarrow \lambda$  ou  $A \longrightarrow B$ , sempre esse crescimento implicará no aumento de, pelo menos, um símbolo terminal. Como o comprimento não pode exceder  $|w|$ , uma derivação de  $w$  não pode envolver mais do que  $|w|$  etapas. Assim, se até a etapa  $|w|$  não tivermos derivado  $w$ , então  $w$  não será gerada pela gramática.

■

A objeção ao método de análise de pesquisa exaustiva com respeito à ineficiência é muito mais difícil de contornar. Numa análise de pesquisa exaustiva poderemos produzir algo como  $|P|^n$  formas sentenciais na primeira etapa, algo como  $|P|^2$  na segunda, e assim por diante, tal que a análise completa pode produzir  $|P|^{|w|}$  formas sentenciais. Certamente isso é muito e torna este método impraticável pelo que teria de ser encontrado um método mais eficiente.

### 5.3.2 Métodos de Análise mais Eficientes

Infelizmente, os métodos de análises eficientes não são óbvios e os algoritmos conhecidos são complexos. Este é um tópico que pertence, propriamente, ao curso de compiladores. Enunciaremos alguns resultados isolados para dar ao leitor uma idéia da extensão do assunto.

**Teorema 5.3.4** Para toda gramática livre do contexto existe um algoritmo que analisa qualquer  $w \in L(G)$  num número de operações proporcional a  $|w|^3$ .

■

Existem vários métodos conhecidos com uma eficiência da mesma ordem, entre eles destacamos

1. O algoritmo CYK, desenvolvido de forma independente por Tadao Kasami [Kas65], Daniel H. Younger [You67] e John Cocke (em conjunto com Jacob T. Schwartz) [?] na década do 60.
2. O algoritmo de Early desenvolvido por Jay Early em [Ear70]

Uma das razões para não perseguir isso com mais detalhe é que mesmo esses algoritmos são insatisfatórios. Um método no qual a sua complexidade vem na terceira potência do comprimento da cadeia é muito ineficiente, e um compilador baseado nele necessitaria de uma quantidade de tempo excessiva para analisá-la mesmo para programas não muito grandes. O que gostaríamos

### 5.3. Algoritmos de Análise de Pertinência

---

de ter é um método de análise que tomasse um tempo proporcional ao comprimento da cadeia. Tais métodos são referidos como algoritmos de análise em **tempo linear**. Não conhecemos nenhum método de análise em tempo linear para linguagens livres do contexto, em geral. Tais algoritmos podem ser encontrados, para casos especiais, porém importantes.

**Exemplo 5.3.5** Considere um tipo de gramática livre do contexto na qual todas as produções são da forma

$$A \longrightarrow ax$$

onde  $A \in V$ ,  $a \in T$ ,  $x \in V^*$  e para as quais existe uma restrição de que qualquer par  $(A, a)$  pode ocorrer em no máximo uma produção com lado esquerdo  $A$ , isto é, para um dado  $A$  no lado esquerdo não pode existir mais do que uma produção tendo um símbolo terminal  $a$  no inicio da cadeia, na direita da produção. Então qualquer cadeia na linguagem gerada por tal gramática pode ser analisada em no máximo  $|w|$  operações.

Para ver isto, use o método de pesquisa exaustiva com a cadeia  $w = a_1a_2 \cdots a_m$ . Como só existe, no máximo, uma produção com  $S$  na esquerda e iniciando com  $a_1$  na direita, a derivação deve começar com

$$S \implies a_1A_1A_2 \cdots A_m.$$

Em seguida, substituiremos a variável  $A_1$ , mas, novamente, existe no máximo uma escolha, e portanto devemos ter

$$S \implies a_1A_1A_2 \cdots A_m \implies a_1a_2B_1B_2 \cdots B_kA_2 \cdots A_m.$$

Vemos que cada etapa produz um símbolo terminal e portanto o processo todo deve ser completado em não mais do que  $|w|$  etapas.

Este exemplo não é artificial. Este tipo de gramática é chamada **gramática simples** ou **S-gramática**. Assim, em S-gramáticas todas as produções têm a forma  $A \longrightarrow ax$  com  $a \in T$  e  $x \in V^*$ . Se  $A \longrightarrow ax$  e  $A \longrightarrow ay$  são produções em  $P$ , então necessariamente  $x = y$ . Chamaremos a esta condição de **S-condição**.

Muitas características de uma linguagem, semelhante a Pascal, podem ser expressas com S-gramáticas.

**Exemplo 5.3.6** Seja  $\mathcal{L}$  a linguagem sobre o alfabeto  $\{(, )\}$ , consistindo de todas as cadeias de parênteses “aninhadas corretamente”, isto é, cadeias com a mesma quantidade de ambos os tipos de parênteses, e tais que qualquer prefixo contenha uma quantidade de “(” maior ou igual que “)”. Por exemplo, algumas cadeias válidas são  $((())()$  e  $((((())())()$ , já as cadeias  $()))()$ ,  $()()$  e  $\lambda$  não são consideradas como “aninhadas corretamente”. A seguinte S-gramática gera  $\mathcal{L}$ .

$$\begin{aligned} S &\longrightarrow (A \\ A &\longrightarrow (AA \mid ) \end{aligned}$$

Porém S-gramáticas são muito restritivas. Uma sub-classe de linguagens livres do contexto menos restritiva, as linguagens livres do contexto determinísticas, e que contempla a maioria das linguagens de programação, usualmente tem sido usada para realizar a análises de uma sub-classe de gramáticas livres do contexto em tempo linear (um tal algoritmo pode ser encontrado em [?]). Essa sub-classe de gramáticas são as gramáticas esquerda-direita, direita recursivas, ou abreviando  $LR(k)$ , e as gramáticas esquerda-direita, esquerda recursivas, ou abreviando  $LL(k)$ . As gramáticas  $LL(k)$  e  $LR(k)$  geram cadeias que podem ser analisadas lendo-as da esquerda para direita, considerando  $k$  símbolos em cada passo.

## 5.4 Gramáticas Livres do Contexto e Linguagens de Programação

Onde mais se utiliza a teoria das linguagens formais é na definição de linguagens de programação e na construção de interpretadores e compiladores. O problema básico aqui é definir precisamente uma linguagem de programação e usar esta definição como o ponto de início para escrever traduções eficientes e confiáveis de programas. Tanto as linguagens regulares como as livre do contexto são importantes na consecução desses objetivos. Linguagens regulares são usadas no reconhecimento de certos padrões simples que ocorrem em linguagens de programação, mas precisamos de linguagens livres do contexto para modelar aspectos mais complicados.

Podemos definir uma linguagem de programação por uma gramática. É costume quando se descreve a sintaxe de linguagens de programação usar uma convenção para especificar a gramática, chamada a “Forma de Backus-Naur” ou BNF . Esta forma, embora aparentemente diferente, é em essência a mesma que viemos usando. Em BNF, as variáveis são colocadas em colchetes triangulares “⟨ ⟩” e os símbolos terminais são escritos sem qualquer marca especial. A BNF também usa símbolos auxiliares como |, do mesmo modo como fizemos. Assim, a gramática no exemplo 5.2.4 deve aparecer, em BNF, como

```

⟨expressão⟩ ::= ⟨termo⟩ | ⟨expressão⟩ + ⟨termo⟩,
⟨termo⟩ ::= ⟨fator⟩ | ⟨termo⟩ * ⟨fator⟩,
⟨fator⟩ ::= ⟨identificador⟩ | ⟨(expressão)⟩
⟨identificador⟩ ::= a | b | c
    
```

Os símbolos +, \*, ) e ( são terminais; o símbolo | representa uma alternativa como em nossa notação e o símbolo ::= é usado em lugar de →. As descrições BNF das linguagens de programação tendem a usar identificadores de variáveis de forma mais explícita de modo a descrever melhor o que faz a produção. Mas além dessas questões, não existem diferenças significativas entre essas duas notações.

Muitas partes de uma linguagem de programação semelhante a Pascal são factíveis de serem definidas através de uma gramática livre do contexto. Por exemplo, um comando “if-then-else” pode ser definido por

```

⟨if-comando⟩ ::= if ⟨condição⟩⟨then-cláusula⟩⟨else-cláusula⟩
    
```

Aqui a palavra chave “if” é um símbolo terminal. Todos os outros termos são variáveis as quais ainda deverão ser definidas. Se compararmos isso com o exemplo 5.3.5 vemos que isto parece com uma S-produção. A variável ⟨if-comando⟩ à esquerda é sempre associada com o terminal **if** na direita. Por isso os comandos como esses são facilmente e eficientemente analisados. Vemos aqui uma razão porque usamos nas linguagens de programação algumas palavras-chaves:

## 5.5. Simplificações de Gramáticas Livres do Contexto

---

elas não somente provem alguma estrutura visual que nos pode guiar na leitura do programa, como também torna o trabalho do compilador muito mais fácil.

Infelizmente nem todas as características de uma linguagem de programação típica podem ser expressas por uma S-gramática. Por exemplo, as regras para  $\langle$ expressão $\rangle$  não são desse tipo, de modo que sua análise torna-se menos óbvia. Então colocamos a seguinte questão: que regras gramaticais podemos permitir e ainda assim analisá-las eficientemente? Em compiladores tem-se usado bastante o que se chama gramáticas LL( $k$ ) e LR( $k$ ). Estas gramáticas, mesmo tendo um poder de expressão menor que as gramáticas livres do contexto, têm a habilidade de expressar características menos óbvias de linguagens de programação permitindo-nos ainda analisá-las em tempo linear.

## 5.5 Simplificações de Gramáticas Livres do Contexto

A definição de uma gramática livre do contexto não impõe qualquer restrição no lado direito de uma produção. Entretanto não é necessário liberdade completa. No teorema 5.3.3, vimos a conveniência de certas restrições sobre as formas gramaticais, eliminando produções da forma  $A \rightarrow \lambda$  e  $A \rightarrow B$ , tornando os argumentos mais fáceis. Em muitas situações é desejável mesmo colocar mais restrições sobre a gramática. Por isso precisamos estudar métodos de transformar uma gramática livre do contexto arbitrária numa equivalente, que satisfaz certas restrições sobre sua forma.

A cadeia vazia desempenha um papel singular em muitos teoremas e provas e é necessário dar a ela uma atenção especial. Preferimos removê-la estudando somente linguagens que não a contém. Em se fazendo isso não perdemos generalidade, pelo seguinte motivo. Seja  $\mathcal{L}$  qualquer linguagem livre do contexto contendo a cadeia vazia, e  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto para  $L - \{\lambda\}$ . Então a gramática obtida acrescentando-se a nova variável  $S_0$ , tornando  $S_0$  a variável de início e acrescentando a produção

$$S_0 \rightarrow S \mid \lambda,$$

gera  $\mathcal{L}$ , ou seja existe um método para obter uma gramática livre do contexto  $\hat{G}$  tal que  $L(\hat{G}) = L(G) - \{\lambda\}$ . Portanto, qualquer conclusão não trivial que podemos tirar para  $L - \{\lambda\}$  vale também para  $\mathcal{L}$ . Conseqüentemente, para todos os propósitos práticos, não existe nenhuma diferença entre gramáticas livres do contexto que incluem  $\lambda$  e aquelas que não o incluem. No que segue nos restringiremos a gramáticas livre do contexto que não geram  $\lambda$ .

### 5.5.1 Regra Geral de Substituição

**Teorema 5.5.1** *Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Suponha que  $P$  contém produções da forma*

$$A \rightarrow x_1 B x_2. \tag{5.1}$$

*Suponha que  $A$  e  $B$  são variáveis diferentes e que*

$$B \longrightarrow y_1 \mid y_2 \mid \cdots \mid y_n$$

é o conjunto de todas as produções em  $P$  que tem  $B$  como o lado esquerdo. Seja  $\widehat{G} = \langle V, T, S, \widehat{P} \rangle$  a gramática na qual  $\widehat{P}$  é construída ao se retirar a produção (5.1) de  $P$  e adicionando a ele as produções

$$A \longrightarrow x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid \cdots \mid x_1 y_n x_2.$$

Então  $L(\widehat{G}) = L(G)$ .

**DEMONSTRAÇÃO:** Suponha que  $w \in L(G)$  é tal que  $S \xrightarrow{*G} w$  (derivação em  $G$ ). Se esta derivação não envolve a produção (5.1), então  $S \xrightarrow{*{\widehat{G}}} w$ .

Se for o caso de envolver, então observe na derivação a primeira vez que a produção (5.1) é usada. O  $B$ , assim introduzido, tem que ser trocado. Não perdemos nada em saber que isso é feito imediatamente. Portanto,

$$S \xrightarrow{*G} u_1 A u_2 \xrightarrow{G} u_1 x_1 B x_2 u_2 \xrightarrow{G} u_1 x_1 y_j x_2 u_2.$$

Mas, com a gramática  $\widehat{G}$ , podemos obter

$$S \xrightarrow{*{\widehat{G}}} u_1 A u_2 \xrightarrow{{\widehat{G}}} u_1 x_1 y_j x_2 u_2.$$

Portanto, podemos obter a mesma forma sentencial com  $G$  e  $\widehat{G}$ . Se (5.1) é usado novamente mais adiante, na derivação de  $w$ , o argumento se repete. Segue então, por indução no número de vezes que a produção é aplicada, que

$$S \xrightarrow{*{\widehat{G}}} w.$$

Portanto, se  $w \in L(G)$ , então  $w \in L(\widehat{G})$ . Analogamente se mostra que se  $w \in L(\widehat{G})$ , então  $w \in L(G)$ . ■

Observe que neste algoritmo para transformar uma gramática em outra equivalente, a gramática resultante ( $\widehat{G}$ ) é maior que  $G$ , no sentido que tem mais produções, porém ela gera as cadeias da linguagem em uma quantidade menor (ou igual) de passos.

**Exemplo 5.5.2** Considere  $G = \langle \{A, B\}, \{a, b\}, A, P \rangle$ , com produções

$$\begin{aligned} A &\longrightarrow a \mid aaA \mid abBc, \\ B &\longrightarrow abbA \mid b. \end{aligned}$$

Usando a substituição sugerida para a variável  $B$ , obtemos a gramática  $\widehat{G}$  com produções

$$\begin{aligned} A &\longrightarrow a \mid aaA \mid ababbAc \mid abbc, \\ B &\longrightarrow abbA \mid b. \end{aligned}$$

## 5.5. Simplificações de Gramáticas Livres do Contexto

---

A nova gramática,  $\widehat{G}$ , é equivalente a  $G$ . A cadeia  $aaabbc$  tem derivação

$$A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabbc$$

em  $G$ , e a correspondente derivação em  $\widehat{G}$  é

$$A \Rightarrow aaA \Rightarrow aaabbc.$$

Note que, neste caso, a variável  $B$  e as produções associadas estão, ainda, na gramática, embora elas não possam desempenhar seus papéis nas derivações.

### 5.5.2 Remoção de Produções Esquerda-Recursivas

O teorema 5.5.1 é uma regra de substituição simples e intuitiva: uma produção  $A \rightarrow x_1Bx_2$  pode ser eliminada da gramática se colocarmos em seu lugar o conjunto de produções nas quais  $B$  é trocado por todas as cadeias que ele deriva em uma etapa. Neste resultado é preciso que  $A \neq B$ . O caso quando  $A = B$  será parcialmente respondido no próximo teorema.

**Teorema 5.5.3** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Divida o conjunto de produções cujo lado esquerdo é alguma variável (digamos  $A$ ), em dois subconjuntos disjuntos

$$A \rightarrow Ax_1 \mid Ax_2 \mid \cdots \mid Ax_n, \quad (5.2)$$

e

$$A \rightarrow y_1 \mid y_2 \mid \cdots \mid y_m, \quad (5.3)$$

onde  $x_i, y_i$  estão em  $(V \cup T)^*$ , mas  $A$  não é prefixo de qualquer  $y_i$ . Considere a gramática  $\widehat{G} = \langle V \cup \{Z\}, T, S, \widehat{P} \rangle$ , onde  $Z \notin V$  e  $\widehat{P}$  é obtido trocando todas as produções de  $P$  da forma (5.2) e (5.3) por

$$A \rightarrow y_i \mid y_iZ, \quad i = 1, \dots, m,$$

$$Z \rightarrow x_i \mid x_iZ, \quad i = 1, \dots, n.$$

Então  $L(G) = L(\widehat{G})$ .

**DEMONSTRAÇÃO:** Considere uma derivação em  $G$ , envolvendo a variável  $A$ , numa seqüência de etapas usando produções da forma (5.2), digamos

$$A \Rightarrow Ax_i \Rightarrow Ax_jx_i \xrightarrow{*} Ax_k \dots x_jx_i$$

Mais cedo ou mais tarde, chegaremos numa sentença na qual uma produção da forma (5.3) deve ser usada, tal que

$$A \xrightarrow{*} y_h x_k \dots x_j x_i.$$

Mas essa seqüência também pode ser gerada por  $\widehat{G}$  por

$$A \Rightarrow y_h Z \Rightarrow y_h x_k Z \xrightarrow{*} y_h x_k \dots x_j Z \Rightarrow y_h x_k \dots x_j x_i.$$

Portanto, para qualquer  $w$  tal que  $S \xrightarrow{*}_G w$ , também temos  $S \xrightarrow{*}_{\widehat{G}} w$ . O argumento pode ser revertido facilmente. ■

As produções da forma (5.2) são chamadas **esquerda-recursivas**. Elas são um caso especial de recursão à esquerda em gramáticas. Em geral uma gramática diz-se **esquerda-recursiva** se existe alguma variável  $A$  para a qual

$$A \xrightarrow{*} Ax$$

seja possível.

**Exemplo 5.5.4** Use o teorema 5.5.3 para remover produções esquerda-recursivas da gramática

$$\begin{aligned} A &\longrightarrow Aa \mid aBc \mid \lambda, \\ B &\longrightarrow Bb \mid ba. \end{aligned}$$

Aplicando o teorema para a variável  $A$ , fazendo as identificações  $x_1 = a$ ,  $y_1 = aBc$  e  $y_2 = \lambda$ , obtemos o novo conjunto de produções

$$\begin{aligned} A &\longrightarrow aBc \mid aBcZ \mid \lambda \mid Z, \\ Z &\longrightarrow a \mid aZ, \\ B &\longrightarrow Bb \mid ba. \end{aligned}$$

Aplicando o teorema novamente, desta vez à variável  $B$ , obtemos o resultado final

$$\begin{aligned} A &\longrightarrow aBc \mid aBcZ \mid \lambda \mid Z, \\ Z &\longrightarrow a \mid aZ, \\ B &\longrightarrow ba \mid baY, \\ Y &\longrightarrow b \mid bY, \end{aligned}$$

Para remover produções esquerda-recursivas deve-se introduzir novas variáveis. A gramática resultante pode ser considerada mais simples do que a original, porque ela não tem qualquer produção esquerda-recursiva. Por outro lado, ela, geralmente, tem mais variáveis e produções.

## 5.5. Simplificações de Gramáticas Livres do Contexto

---

### 5.5.3 Remoção de Produções Inúteis

Queremos sempre remover produções de gramáticas que nunca tomam parte de derivações. Por exemplo, na gramática cujo conjunto de produções é

$$\begin{aligned} S &\longrightarrow B \mid aSa \mid aa \\ A &\longrightarrow a \mid aA, \\ B &\longrightarrow Bb \end{aligned}$$

a produção  $S \longrightarrow B$  não desempenha qualquer papel, pois  $B$  não pode ser transformado numa cadeia terminal. Por outro lado, as produções  $A \longrightarrow a \mid aA$  também não desempenham qualquer papel, pois embora de  $A$  possamos derivar uma cadeia terminal, nunca poderemos atingi-las a partir de  $S$ . Remover estas produções leva a uma gramática equivalente e mais simplificada.

**Definição 5.5.5** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Uma variável  $A$  diz-se **útil** se existe  $w \in L(G)$  tal que

$$S \xrightarrow{*} xAy \xrightarrow{*} w,$$

$x, y \in (V \cup T)^*$ . Em palavras, uma variável é útil se ocorre em, no mínimo, uma derivação. Caso contrário ela diz-se **inútil**. Uma produção é inútil se ela envolve qualquer variável inútil.

**Exemplo 5.5.6** Elimine símbolos e produções inúteis de  $G = \langle V, T, S, P \rangle$ , onde  $V = \{S, A, B, C\}$  e  $T = \{a, b\}$ , com  $P$  consistindo de

$$\begin{aligned} S &\longrightarrow aS \mid aA \mid C \\ A &\longrightarrow a \\ B &\longrightarrow bb \\ C &\longrightarrow aCb \end{aligned}$$

Primeiro identificamos o conjunto de variáveis que leva a uma cadeia terminal. Como  $A \longrightarrow a$  e  $B \longrightarrow aa$ , as variáveis  $A$  e  $B$  pertencem a esse conjunto. Do mesmo modo pertence  $S$  pois  $S \xrightarrow{*} aA \xrightarrow{*} aa$ . Entretanto não podemos usar esse argumento para  $C$ . Logo,  $C$  é inútil. Removendo  $C$  e as produções envolvendo  $C$  teremos a gramática  $G_1 = \langle V_1, T, S, P_1 \rangle$ , com  $V_1 = \{S, A, B\}$  e as produções

$$\begin{aligned} S &\longrightarrow aS \mid aA \\ A &\longrightarrow a \\ B &\longrightarrow bb \end{aligned}$$

Agora, devemos eliminar aquelas variáveis que não podem ser atingidas a partir do  $S$ . Para isso, podemos desenhar um **grafo de dependência** para as variáveis. Os grafos de dependência são maneiras de visualizar relacionamentos complexos e são encontrados em muitas aplicações. Para as gramáticas livres do contexto, os grafos de dependência tem seus vértices rotulados com

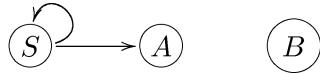


Figura 5.7: Exemplo de grafo de dependência

variáveis. Existirá uma aresta desde um vértice  $C$  em  $D$  se e somente se existe uma produção da forma

$$C \longrightarrow xDy.$$

Um grafo de dependência para  $G_1$  é mostrado na figura 5.7.

Se não existir um caminho rotulado  $S$  ao vértice rotulado com uma variável, então essa variável é inútil. No nosso caso, vendo a figura 5.7, a variável  $B$  é inútil. Removendo  $B$  junto com as correspondentes produções chegamos à resposta final  $\widehat{G} = \langle \widehat{V}, \widehat{T}, S, \widehat{P} \rangle$  com  $\widehat{V} = \{S, A\}$ ,  $\widehat{T} = \{a\}$  e as produções

$$\begin{aligned} S &\longrightarrow aS \mid aA, \\ A &\longrightarrow a \end{aligned}$$

A formalização desse processo leva à construção geral e ao teorema correspondente.

**Teorema 5.5.7** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Então existe uma gramática livre do contexto,  $\widehat{G} = \langle \widehat{V}, \widehat{T}, S, \widehat{P} \rangle$ , tal que  $L(G) = L(\widehat{G})$  e  $\widehat{G}$  não contém qualquer variável inútil.

**DEMONSTRAÇÃO:** A gramática  $\widehat{G}$  pode ser gerada de  $G$ , usando um algoritmo constituído de duas partes. Na primeira parte, construímos uma gramática intermediária,  $G_1 = \langle V_1, T, S, P_1 \rangle$ , tal que  $V_1$  contém somente variáveis das quais podemos derivar uma cadeia de símbolos terminais, isto é, variáveis  $A$  para as quais

$$A \xrightarrow{*} w \in T^*.$$

As etapas do algoritmo são:

1. Torne  $V_1$  igual a  $\emptyset$ .
2. Repita o seguinte passo até não conseguir mais acrescentar nenhuma variável a  $V_1$ .
  - (a) Para  $A \in V$ , onde  $P$  tem uma produção da forma

$$A \longrightarrow w, \text{ onde } w \in (T \cup V_1)^*$$

adicone  $A$  a  $V_1$ .

3. Tome  $P_1$  como todas as produções em  $P$  com símbolos em  $(V_1 \cup T)^*$ .

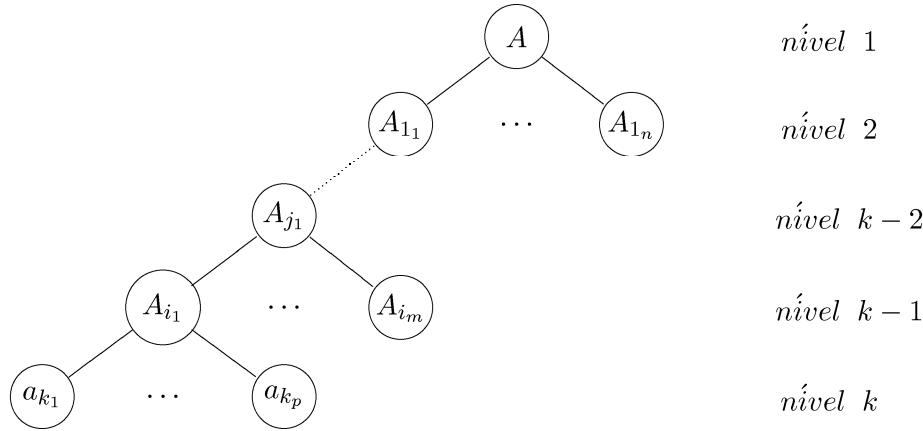


Figura 5.8: Árvore de derivação parcial

É claro que este processo terminará. É igualmente claro que se  $S \notin V_1$ , então  $L(G) = L(G_1) = \emptyset$ . Por outro lado, se  $A \in V_1$ , então  $A \xrightarrow{*} w$ , para algum  $w \in T^*$ , pode ser derivada em  $G_1 = \langle V_1, T, S, P_1 \rangle$ . O problema restante é se todo  $A$  para o qual  $A \xrightarrow{*} w = a_1a_2 \cdots a_m$  é adicionado a  $V_1$  antes do procedimento terminar. Considere qualquer um desses  $A$ 's e olhe a árvore de derivação parcial para aquela derivação (figura 5.8). No nível  $k$ , existem somente símbolos terminais, portanto toda variável  $A_i$  no nível  $k-1$  será adicionado a  $V_1$ , no primeiro passo, na etapa 2 do algoritmo. Qualquer variável no nível  $k-2$  será adicionado a  $V_1$ , no segundo passo da etapa 2. Na terceira vez da etapa 2, todas as variáveis no nível  $k-3$  serão adicionados, e assim por diante. O algoritmo não pode terminar enquanto existirem variáveis na árvore que não estejam ainda em  $V_1$ . Portanto,  $A$  será, mais cedo ou mais tarde, colocado em  $V_1$ .

Na segunda parte da construção, obtemos a resposta final, que é  $\widehat{G}$  de  $G$ . Desenhamos o grafo de dependência para  $G_1$  e dele achamos todas as variáveis que não podem ser atingidas de  $S$ . Essas são removidas do conjunto de variáveis, assim como as correspondentes produções. Podemos também eliminar qualquer terminal que não ocorre em alguma produção útil. O resultado é a gramática  $\widehat{G} = \langle \widehat{V}, \widehat{T}, S, \widehat{P} \rangle$ .

Por construção,  $\widehat{G}$  não contém qualquer símbolo ou produção inútil. Além disso, para cada  $w \in L(G)$  temos a derivação

$$S \xrightarrow{*} xAy \xrightarrow{*} w.$$

Como a construção retém  $A$  e todas as produções associadas, temos tudo que precisamos para fazer a derivação

$$S \xrightarrow{*_{\widehat{G}}} xAy \xrightarrow{*_{\widehat{G}}} w.$$

Logo,  $w \in L(\widehat{G})$  e portanto  $L(G) \subseteq L(\widehat{G})$ . Por outro lado, a gramática  $\widehat{G}$  é construída de  $G$  removendo produções, isto é  $\widehat{P} \subseteq P$ . Portanto,  $L(\widehat{G}) \subseteq L(G)$ . Colocando os dois resultados juntos, vemos que  $G$  e  $\widehat{G}$  são equivalentes. ■

#### 5.5.4 Remoção de $\lambda$ -Produções

Um espécie de produção as vezes indesejada, é aquela na qual o lado direito é a cadeia vazia, isto é  $\lambda$ -produções. Por exemplo  $\lambda$ -produções, como mencionado na seção 5.3, não permitem estabelecer um critério de parada geral para algoritmos de pertinência.

**Definição 5.5.8** Qualquer produção de uma gramática livre do contexto da forma

$$A \longrightarrow \lambda$$

é chamada uma  **$\lambda$ -produção**. Se para uma variável  $A$  a derivação

$$A \xrightarrow{*} \lambda$$

é possível, ela é chamada **anulável**.

Uma gramática pode gerar uma linguagem não contendo  $\lambda$ , chamada **linguagem livre de  $\lambda$** , ainda que a gramática possua  $\lambda$ -produções e variáveis anuláveis. Nesses casos, as  $\lambda$ -produções podem ser removidas

**Exemplo 5.5.9** Considere a gramática

$$\begin{aligned} S &\longrightarrow aS_1b, \\ S_1 &\longrightarrow aS_1b \mid \lambda. \end{aligned}$$

Esta gramática gera a linguagem livre de  $\lambda$ ,  $\{a^n b^n / n \geq 1\}$ . A  $\lambda$ -produção  $S_1 \longrightarrow \lambda$  pode ser removida após adicionar-se novas produções obtidas substituindo-se  $\lambda$  por  $S_1$ , onde ela ocorre no lado direito. Fazendo isso, obtemos a gramática

$$\begin{aligned} S &\longrightarrow aS_1b \mid ab, \\ S_1 &\longrightarrow aS_1b \mid ab. \end{aligned}$$

É fácil mostrar que esta nova gramática gera a mesma linguagem que a original.

Em situações mais gerais, as substituições para as  $\lambda$ -produções podem ser feitas de modo similar.

**Teorema 5.5.10** Seja  $G = \langle V, S, T, P \rangle$  uma gramática livre do contexto tal que  $\lambda \notin L(G)$ . Então existe uma gramática equivalente,  $\widehat{G} = \langle V, T, S, \widehat{P} \rangle$ , sem  $\lambda$ -produções.

**DEMONSTRAÇÃO:** Primeiro encontraremos  $V_N$ , o conjunto de todas as variáveis anuláveis de  $G$ , usando as seguintes etapas:

1. Para toda produção  $A \longrightarrow \lambda$ , ponha  $A$  em  $V_N$ .

## 5.5. Simplificações de Gramáticas Livres do Contexto

---

2. Repita o seguinte passo até não existir variável a ser colocada em  $V_N$ .

- (a) Para todas as produções em  $P$

$$B \longrightarrow w,$$

onde  $w \in V_N^*$ , ponha  $B$  em  $V_N$ .

Uma vez obtido o conjunto  $V_N$ , estamos pronto para construir  $\widehat{P}$ . Para fazer isso, colocamos em  $\widehat{P}$  todas as produções que não são  $\lambda$ -produções, assim como todas aquelas geradas trocando as variáveis anuláveis, isto é variáveis em  $V_N$ , por  $\lambda$ , em todas as combinações possíveis. Por exemplo, se  $x_i$  e  $x_j$  são anuláveis, existirá uma produção em  $\widehat{P}$ , com  $x_i$  trocado por  $\lambda$ , uma na qual  $x_j$  é trocada por  $\lambda$  e uma na qual ambas são trocadas com  $\lambda$ . Existe uma exceção: se todos os  $x_i$  são anuláveis, a produção  $A \longrightarrow \lambda$  não será colocada em  $\widehat{P}$ . É claro que  $\widehat{G}$  é equivalente a  $G$ . ■

**Exemplo 5.5.11** Ache uma gramática livre do contexto sem  $\lambda$ -produções, equivalente à gramática definida por

$$\begin{aligned} S &\longrightarrow ABaC, \\ A &\longrightarrow BC, \\ B &\longrightarrow b \mid \lambda, \\ C &\longrightarrow D \mid \lambda, \\ D &\longrightarrow d. \end{aligned}$$

Da primeira etapa da construção do teorema 5.5.10, achamos que as variáveis anuláveis são  $A, B$  e  $C$ . Então, seguindo a segunda etapa da construção, primeiro colocamos aquelas produções da gramática original que não são  $\lambda$ -produções, ou seja

$$\begin{aligned} S &\longrightarrow ABaC, \\ A &\longrightarrow BC, \\ B &\longrightarrow b \\ C &\longrightarrow D \\ D &\longrightarrow d. \end{aligned}$$

Depois adicionamos a ela, todas aquelas produções resultantes de eliminar as variáveis anuláveis (neste caso  $A, B$  e  $C$ ) que aparecem no lado direito em todas as combinações possíveis. Ou seja,

$$\begin{aligned} S &\longrightarrow BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a, \\ A &\longrightarrow C \mid B. \end{aligned}$$

Juntando os dois conjuntos de produções, temos a seguinte gramática:

$$\begin{aligned} S &\longrightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a, \\ A &\longrightarrow BC \mid C \mid B, \\ B &\longrightarrow b, \\ C &\longrightarrow D, \\ D &\longrightarrow d. \end{aligned}$$

### 5.5.5 Remoção de Produções Unitárias

Assim como as  $\lambda$ -produções, produções em que o lado direito é composto por uma única variável são indesejáveis.

**Definição 5.5.12** Qualquer produção numa gramática livre do contexto da forma,

$$A \longrightarrow B,$$

onde  $A, B \in V$ , é chamada **produção unitária**.

**Teorema 5.5.13** Seja  $G = \langle V, S, T, P \rangle$  uma gramática livre do contexto sem  $\lambda$ -produções. Então, existe uma gramática livre do contexto, equivalente  $\widehat{G} = \langle \widehat{V}, \widehat{T}, S, \widehat{P} \rangle$ , que não contém produções unitárias.

**DEMONSTRAÇÃO:** Obviamente que qualquer produção unitária da forma  $A \longrightarrow A$  pode ser removida sem nenhum efeito sobre a gramática. Consideremos as produções unitárias da forma  $A \longrightarrow B$ , com  $A \neq B$ . A primeira vista, pode parecer que podemos usar o teorema 5.5.1, diretamente com  $x_1 = x_2 = \lambda$ , trocando

$$A \longrightarrow B$$

com

$$A \longrightarrow y_1 \mid y_2 \mid \dots \mid y_n,$$

onde  $B \longrightarrow y_1 \mid \dots \mid y_n$ .

Mas isso nem sempre funcionará, pois vai depender da escolha da produção a ser substituída. Por exemplo, no caso especial da gramática

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow B \mid aA \\ B &\longrightarrow C \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

se substituirmos  $A \longrightarrow B$  usando o teorema 5.5.1, obteríamos a gramática

## 5.5. Simplificações de Gramáticas Livres do Contexto

---

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow C \mid bB \mid BB \mid aA \\ B &\longrightarrow C \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

Agora substituindo  $A \longrightarrow C$  obteríamos a gramática

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow B \mid ab \mid bB \mid BB \mid aA \\ B &\longrightarrow C \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

Agora substituindo  $A \longrightarrow B$  obteríamos a gramática

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow C \mid ab \mid bB \mid BB \mid aA \\ B &\longrightarrow C \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

Substituindo  $A \longrightarrow C$  obteríamos a gramática anterior a ela, ficando em um ciclo, que só poderia ser quebrado escolhendo outra produção unitária para eliminar. Por exemplo poderíamos substituir  $B \longrightarrow C$  e obteríamos

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow C \mid ab \mid bB \mid BB \mid aA \\ B &\longrightarrow B \mid ab \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

Eliminando  $B \longrightarrow B$  teríamos a gramática

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow C \mid ab \mid bB \mid BB \mid aA \\ B &\longrightarrow ab \mid bB \mid BB \\ C &\longrightarrow B \mid ab \end{aligned}$$

Substituindo agora  $C \longrightarrow B$  teríamos a gramática

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow C \mid ab \mid bB \mid BB \mid aA \\ B &\longrightarrow ab \mid bB \mid BB \\ C &\longrightarrow ab \mid bB \mid BB \end{aligned}$$

Finalmente, substituindo  $A \longrightarrow C$ , teríamos a gramática

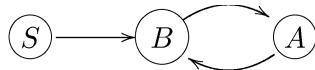


Figura 5.9: Exemplo de grafo de dependência para remoção de produções unitárias.

$$\begin{aligned} S &\longrightarrow aA \\ A &\longrightarrow ab \mid bB \mid BB \mid aA \\ B &\longrightarrow ab \mid bB \mid BB \\ C &\longrightarrow ab \mid bB \mid BB \end{aligned}$$

que não possui produções unitárias (embora tenha algumas produções inúteis) e que é equivalente à original.

Uma maneira mais rápida e sem perigo de ficar em um ciclo é a seguinte: primeiro achamos para cada  $A$ , todas as variáveis  $B$  tais que

$$A \xrightarrow{*} B. \quad (5.4)$$

Podemos fazer isso desenhando um grafo de dependência, com uma aresta  $(C, D)$  sempre quando a gramática tem a produção  $C \longrightarrow D$ , então (5.4) se verifica quando existir um caminho entre  $A$  e  $B$ . A nova gramática  $\hat{G}$  é gerada primeiro colocando-se em  $\hat{P}$  todas as produções não unitárias de  $P$ . Em seguida, para todo  $A$  e  $B$  satisfazendo (5.4), acrescentamos a  $\hat{P}$  as produções

$$A \longrightarrow y_1 \mid y_2 \mid \cdots \mid y_n,$$

onde  $B \longrightarrow y_1 \mid y_2 \mid \cdots \mid y_n$  são todas as produções em  $\hat{P}$ , com  $B$  à esquerda. Observe que uma vez que  $B \longrightarrow y_1 \mid y_2 \mid \cdots \mid y_n$  é tirado de  $\hat{P}$ , nenhum dos  $y_i$  pode ser uma simples variável, desse modo nenhuma produção unitária é criada na etapa.

Para mostrar que a gramática resultante é equivalente à original segue-se a mesma linha de raciocínio do teorema 5.5.10. ■

Observe que o algoritmo acima pode gerar produções inúteis. Portanto, depois, caso seja necessário, devemos aplicar a  $\hat{G}$  o algoritmo do teorema 5.5.7.

**Exemplo 5.5.14** *Remover todas as produções unitárias de*

$$\begin{aligned} S &\longrightarrow Aa \mid B, \\ A &\longrightarrow a \mid bc \mid B, \\ B &\longrightarrow A \mid bb. \end{aligned}$$

*O grafo de dependência para produções unitárias é dado na figura 5.9. Vemos dele que*

$$S \xrightarrow{*} A, \quad S \xrightarrow{*} B, \quad B \xrightarrow{*} A \text{ e } A \xrightarrow{*} B.$$

*Portanto, adicionemos às produções não unitárias originais*

## 5.6. Formas Normais

---

$$S \longrightarrow Aa, \quad A \longrightarrow a \mid bc, \quad B \longrightarrow bb,$$

as novas regras

$$S \longrightarrow a \mid bc \mid bb; \quad A \longrightarrow bb; \quad B \longrightarrow a \mid bc,$$

para obter a gramática desejada, isto é:

$$\begin{aligned} S &\longrightarrow Aa \mid a \mid bc \mid bb \\ A &\longrightarrow a \mid bc \mid bb \\ B &\longrightarrow bb \mid a \mid bc, \end{aligned}$$

Podemos colocar todos esses resultados juntos para mostrar que as gramáticas para linguagens livres do contexto podem se tornar livres de produções inúteis,  $\lambda$ -produções e produções unitárias.

## 5.6 Formas Normais

Existem muitas tipos de formas normais que podemos estabelecer para gramáticas livres do contexto. Algumas dessas, devido as suas amplas utilidades, foram estudadas exaustivamente. Consideremos, brevemente, duas delas.

### 5.6.1 Forma Normal de Chomsky

Uma espécie de forma normal que podemos ver é aquela na qual o número de símbolos à direita da produção são estritamente limitados. Em particular, podemos pedir que as cadeias no lado direito das produções não tenham mais que dois símbolos.

**Definição 5.6.1** Uma gramática livre do contexto está na **forma normal de Chomsky** se todas as produções são da forma

$$A \longrightarrow BC \text{ ou } A \longrightarrow a,$$

onde  $A, B, C \in V$  e  $a \in T$ .

**Teorema 5.6.2** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto tal que  $\lambda \notin L(G)$ . Então, existe uma gramática equivalente,  $\hat{G} = \langle \hat{V}, T, S, \hat{P} \rangle$ , na forma normal de Chomsky.

**DEMONSTRAÇÃO:** Devido ao teorema 5.5.10 podemos assumir, sem perda de generalidade, que  $G$  não tem  $\lambda$ -produções nem produções unitárias. A construção de  $\hat{G}$  será dada em duas etapas.

**Etapa1:** Construir a partir de  $G$  uma gramática,  $G_1 = \langle V_1, T, S, P_1 \rangle$ , considerando todas as produções em  $P$  na forma

$$A \longrightarrow x_1 x_2 \cdots x_n,$$

onde cada  $x_i$  é um símbolo ou em  $V$  ou  $T$ . Introduzir uma nova variável  $B_a$ , para cada  $a \in T$ , ou seja  $V_1 = V \cup \{B_a / a \in T\}$  e adicionar a produção  $B_a \longrightarrow a$  a  $P_1$ . Se  $n = 1$ , então  $x_1$  deve ser um símbolo terminal, pois não temos produções unitárias. Neste caso, colocar a produção em  $P_1$ . Se  $n \geq 2$ , colocar em  $P_1$  a produção

$$A \longrightarrow C_1 C_2 \cdots C_n,$$

onde  $C_i = x_i$  se  $x_i \in V$  e  $C_i = B_a$  se  $x_i = a$ .

Com isso removemos todos os símbolos terminais das produções de comprimento maior do que um, trocando-os com as novas variáveis introduzidas.

No fim dessa etapa, temos uma gramática  $G_1$ , cujas produções têm a forma

$$A \longrightarrow a, \tag{5.5}$$

ou

$$A \longrightarrow C_1 C_2 \cdots C_n, \tag{5.6}$$

onde  $C_i \in V_1$ . Logo, por consequência do teorema 5.5.1, é fácil ver que  $L(G_1) = L(G)$ . Gramáticas livres do contexto onde todas suas produções têm uma das formas das equações (9.3) ou (5.6) serão ditas estarem na **forma normal pre-Comsky**.

**Etapa 2:** Na segunda etapa, introduzimos variáveis adicionais para reduzir o comprimento do lado direito das produções onde necessário. Primeiro colocar todas as produções da forma (9.3) e todas da forma (5.6), com  $n = 2$  em  $\widehat{P}$ . Para  $n > 2$ , introduzir as novas variáveis  $D_1, D_2, \dots, D_{n-2}$  e colocar em  $\widehat{P}$  as produções

$$A \longrightarrow C_1 D_1; \quad D_1 \longrightarrow C_2 D_2; \dots; \quad D_{n-2} \longrightarrow C_{n-1} C_n.$$

É claro que a gramática,  $\widehat{G}$ , resultante estará na forma normal de Chomsky. Repetidas aplicações do teorema 5.5.1 mostra que  $L(\widehat{G}) = L(G_1)$  e, consequentemente,

$$L(\widehat{G}) = L(G).$$

■

## 5.6. Formas Normais

---

**Exemplo 5.6.3** Converter a gramática, com produções

$$S \longrightarrow ABa; \quad A \longrightarrow aab; \quad B \longrightarrow Ac,$$

para a forma normal de Chomsky.

Pela exigência na construção do teorema 5.6.2, a gramática não tem qualquer  $\lambda$ -produção ou qualquer produção unitária.

Na etapa 1, introduzimos novas variáveis  $B_a$ ,  $B_b$ ,  $B_c$  e usamos o algoritmo para obter

$$\begin{aligned} S &\longrightarrow ABB_a, \\ A &\longrightarrow B_aB_aB_b, \\ B &\longrightarrow AB_c, \\ B_a &\longrightarrow a, \\ B_b &\longrightarrow b, \\ B_c &\longrightarrow c. \end{aligned}$$

Na segunda etapa, introduzimos variáveis adicionais para obter as duas primeiras produções em forma normal, obtendo o resultado final.

$$\begin{aligned} S &\longrightarrow AD_1, \\ D_1 &\longrightarrow BB_a, \\ A &\longrightarrow B_aD_2, \\ D_2 &\longrightarrow B_aB_b, \\ B &\longrightarrow AB_c, \\ B_a &\longrightarrow a, \\ B_b &\longrightarrow b, \\ B_c &\longrightarrow c. \end{aligned}$$

**Exemplo 5.6.4** Seja a gramática livre do contexto seguinte:

$$S \longrightarrow ABaCaC; \quad A \longrightarrow baC \mid ABbA; \quad B \longrightarrow Ac \quad C \longrightarrow bA.$$

Esta gramática não têm nem  $\lambda$ -produções nem produções unitárias, pelo que para obter sua forma normal de Chomsky basta aplicar diretamente o algoritmo do teorema 5.6.2.

Na etapa 1, forma normal pre-Chomsky, para evitar eventuais confusões e simplificar a notação, usaremos  $X$ ,  $Y$  e  $Z$  como as novas variáveis introduzidas nesta etapa em vez das usuais  $B_a$ ,  $B_b$  e  $B_c$ , respectivamente.

$$\begin{aligned} S &\longrightarrow ABXCXC, \\ A &\longrightarrow YXC \mid ABYA, \\ B &\longrightarrow AZ, \\ C &\longrightarrow YA, \\ X &\longrightarrow a, \\ Y &\longrightarrow b, \\ Z &\longrightarrow c. \end{aligned}$$

Na segunda etapa, introduzimos variáveis adicionais para obter as duas primeiras produções em forma normal, obtendo o resultado final.

$$\begin{aligned}
 S &\longrightarrow AD_1, \\
 D_1 &\longrightarrow BD_2, \\
 D_2 &\longrightarrow XD_3, \\
 D_3 &\longrightarrow CD_4, \\
 D_4 &\longrightarrow XC, \\
 A &\longrightarrow YD_5 \mid AD_6, \\
 D_5 &\longrightarrow XC, \\
 D_6 &\longrightarrow BD_7, \\
 D_7 &\longrightarrow YA, \\
 B &\longrightarrow AZ, \\
 C &\longrightarrow YA, \\
 X &\longrightarrow a, \\
 Y &\longrightarrow b, \\
 Z &\longrightarrow c.
 \end{aligned}$$

Um outra maneira de decompor uma produção da forma 5.6 é usando o seguinte algoritmo:  
Seja a produção:  $p = A \longrightarrow C_1 \dots C_n$

Procedimento  $Decompor(p)$

Faça  $k = [n/2]$ , isto é a parte inteira de  $n/2$ .

Caso  $k > 1$  faça:

escreva " $A \longrightarrow DE$ " onde  $D$  e  $E$  são variáveis novas,

$Decompor(D \longrightarrow C_1 \dots C_k)$ ,

$Decompor(E \longrightarrow C_{k+1} \dots C_n)$ .

Caso  $k = 1$  e  $n = 2$  faça:

escreva  $p$ .

Caso Contrário faça:

escreva " $A \longrightarrow C_1 D$ " e " $D \longrightarrow C_2 C_3$ " onde  $D$  é uma variável nova.

Enquanto no algoritmo original são geradas  $n-2$  variáveis e a produção original é substituída por  $n-1$  produções, com este algoritmo seriam geradas  $[n \log n] - 1$  novas variáveis e a produção original é substituída por  $[n \log n]$  produções.

Por exemplo, seguindo usando este algoritmo em vez do outro na etapa 2 do exemplo 5.6.4, obteríamos a seguinte gramática:

## 5.6. Formas Normais

---

$$\begin{aligned}
 S &\longrightarrow D_1D_2, \\
 D_1 &\longrightarrow D_3X, \\
 D_2 &\longrightarrow D_4C, \\
 D_3 &\longrightarrow AB, \\
 D_4 &\longrightarrow CX, \\
 A &\longrightarrow D_5C \mid D_6D_7, \\
 D_5 &\longrightarrow YX, \\
 D_6 &\longrightarrow AB, \\
 D_7 &\longrightarrow YA, \\
 B &\longrightarrow AZ, \\
 C &\longrightarrow YA, \\
 X &\longrightarrow a, \\
 Y &\longrightarrow b, \\
 Z &\longrightarrow c.
 \end{aligned}$$

Comparando, ambas gramáticas vemos que têm exatamente o mesmo número de produções e de variáveis. O segundo algoritmo só começa a ser melhor que o primeiro quando a gramática têm produções com uma quantidade grande de símbolos no lado direito. Porém, quando fazemos manualmente é possível diminuir ainda mais a quantidade de produções, fazendo alguns casamentos. Por exemplo, na gramática da etapa 1 do exemplo 5.6.4 há três ocorrências da subcadeia  $XC$  e duas de  $AB$  e  $YA$ . Tendo isto em conta, podemos obter a seguinte gramática:

$$\begin{aligned}
 S &\longrightarrow D_1D_2, \\
 D_1 &\longrightarrow AB, \\
 D_2 &\longrightarrow D_3D_3, \\
 D_3 &\longrightarrow XC, \\
 A &\longrightarrow YD_3 \mid D_1C, \\
 B &\longrightarrow AZ, \\
 C &\longrightarrow YA, \\
 X &\longrightarrow a, \\
 Y &\longrightarrow b, \\
 Z &\longrightarrow c.
 \end{aligned}$$

### 5.6.2 Forma Normal de Greibach

Como vimos na subseção 5.3.2, as gramáticas simples ou S-gramáticas permitem uma análise de alta eficiência (tempo linear), porém são pouco expressivas, isto é nem toda linguagem livre do contexto pode ser gerada por uma gramática linear. Assim, para cada gramática livre do contexto seria desejável achar uma gramática livre do contexto equivalente o mais próximo possível de uma gramática simples.

**Definição 5.6.5** Uma gramática livre do contexto diz-se na **forma normal de Greibach** se todas as produções têm a forma

$$A \longrightarrow ax,$$

onde  $a \in T$  e  $x \in V^*$ .

Note que devido às formas das produções, uma gramática na forma normal de Greibach não pode ser esquerda recursiva.

Note ainda que toda S-gramática é uma gramática livre do contexto na forma normal de Greibach, porém a inversa não é verdadeira.

**Teorema 5.6.6** *Seja  $G$  uma gramática livre do contexto, com  $\lambda \notin L(G)$ . Então existe uma gramática equivalente,  $\widehat{G}$ , na forma normal de Greibach.*

**DEMONSTRAÇÃO:** O algoritmo para a construção de  $\widehat{G}$ , e a prova do teorema baseado nessa construção é bastante longo. Desde que nosso interesse é na existência de uma forma normal de Greibach, só esboçaremos sua construção.

O algoritmo para reescrever uma gramática na forma normal de Greibach consiste em várias etapas.

**Etapa 1:** Reescrever a gramática na forma normal pre-Chomsky.

**Etapa 2:** Renomear as variáveis de  $V$  por  $A_1, A_2, \dots, A_n$ , de tal forma que  $A_1 = S$ .

**Etapa 3:** Usar os teoremas 5.5.1 e 5.5.3 para reescrever a gramática tal que todas as produções tenham uma das seguintes possíveis formas

$$\begin{aligned} A_i &\longrightarrow A_j x_j, \quad j > i, \\ Z_i &\longrightarrow A_j x_j, \quad j \leq n, \\ A_i &\longrightarrow a x_i, \end{aligned}$$

onde  $a \in T$ ,  $x_i \in V^*$  e  $Z_i$  são as variáveis introduzidas pelo teorema 5.5.3, para eliminar produções esquerda-recursivas. Para ver que isso é possível, olhemos as produções cujos lados esquerdos são  $A_1$ . Tais produções satisfazem as exigências, exceto aquelas da forma

$$A_1 \longrightarrow A_1 x.$$

Para remover essas produções esquerda recursivas aplicamos o teorema 5.5.3. Isto introduz a nova variável  $Z_1$  no processo. Agora, considere as produções  $A_2 \longrightarrow A_1 x$ . Se substituirmos  $A_1$ , usando o teorema 5.5.1, geramos ou produções aceitáveis ou produções da forma  $A_2 \longrightarrow A_2 x$  em cujo caso aplicamos novamente o teorema 5.5.3. Para  $A_3$ , primeiro eliminamos produções da forma  $A_3 \longrightarrow A_1 x$ , pelo teorema 5.5.1, e em seguidas as produções da forma  $A_3 \longrightarrow A_2 x$ . Finalmente, removemos as produções esquerda-recursivas, usando o teorema 5.5.3. Continuando assim até todas as produções estarem na forma requerida.

**Etapa 4:** Após a etapa 3, todas as produções tendo  $A_n$  à esquerda devem ser da forma

$$A_n \longrightarrow a x_n$$

e estão, portanto, na forma correta. As produções cujo lado esquerdo é  $A_{n-1}$  ou são do tipo  $A_{n-1} \longrightarrow a x$ , em cujo caso está certa ou é do tipo  $A_{n-1} \longrightarrow A_n x$  em cujo caso usamos

## 5.6. Formas Normais

---

o teorema 5.5.1 para substituir a (primeira) ocorrência de  $A_n$  na produção  $A_{n-1} \rightarrow A_n x$  pelas produções de  $A_n$ . Após este passo, todas as produções com  $A_n$  e  $A_{n-1}$  no lado esquerdo estarão na forma desejada ( $A \rightarrow ax$ ). Em seguida fazemos o mesmo com as produções de  $A_{n-2}$ , e assim por diante. Ao fim desse processo, todas as produções com  $A_i$ , no lado esquerdo, estarão na forma normal de Greibach. Finalmente, para transformar as produções do tipo  $Z_i \rightarrow A_j x_j$  na forma apropriada, realizamos uma substituição, usando o teorema 5.5.1. ■

**Exemplo 5.6.7** Considere

$$\begin{aligned} A_1 &\longrightarrow A_2 A_2 \mid a, \\ A_2 &\longrightarrow A_1 A_2 \mid b. \end{aligned}$$

A gramática está na forma normal de Chomsky, e as variáveis estão rotuladas apropriadamente, de modo que podemos começar na etapa 3. As produções

$$A_2 \longrightarrow b \text{ e } A_1 \longrightarrow A_2 A_2 \mid a$$

estão na forma exigida na etapa 3. Para transformar a produção  $A_2 \rightarrow A_1 A_2$ , para produções equivalentes na forma normal de Greibach, usamos o teorema 5.5.1, substituindo  $A_1$ , para obter

$$A_2 \longrightarrow A_2 A_2 A_2 \mid a A_2 \mid b.$$

Agora, usamos o teorema 5.5.3, para remover produções esquerda-recursivas, com o que obtemos:

$$\begin{aligned} A_2 &\longrightarrow a A_2 \mid a A_2 Z \mid b \mid b Z, \\ Z &\longrightarrow A_2 A_2 \mid A_2 A_2 Z. \end{aligned}$$

Finalmente, substituímos, usando o teorema 5.5.1, a primeira ocorrência de  $A_2$ , nas produções com  $A_1$  e  $Z$  no lado esquerdo, chegando à forma normal de Greibach:

$$\begin{aligned} A_1 &\longrightarrow a A_2 A_2 \mid a A_2 Z A_2 \mid b A_2 \mid b Z A_2 \mid a, \\ A_2 &\longrightarrow a A_2 \mid a A_2 Z \mid b \mid b Z, \\ Z &\longrightarrow a A_2 A_2 \mid a A_2 Z A_2 \mid b A_2 \mid b Z A_2 \mid a A_2 A_2 Z \mid a A_2 Z A_2 Z \mid b A_2 Z \mid b Z A_2 Z \end{aligned}$$

Como este exemplo indica, transformar uma gramática livre do contexto, embora muito simples, na sua forma normal de Greibach tende a ser um processo muito extenso. Porém, há alguns casos em que um algoritmo mais direto pode ser usado.

Se todas as produções de uma gramática livre do contexto  $G = \langle V, T, S, P \rangle$  tiverem a forma

$$A \longrightarrow aw$$

onde  $A \in V$ ,  $a \in T$  e  $w \in (V \cup T)^*$ , então podemos substituir cada símbolo terminal  $a$  em  $w$  pela variável  $B_a$  e acrescentar a produção  $B_a \rightarrow a$  à gramática, ou seja similar ao artifício usado no algoritmo para transformar gramática livre do contexto na sua forma normal pre-Chomsky.

## 5.7 Exercícios

1. Considere a gramática, com produções

$$S \longrightarrow aSb \mid SS \mid \lambda.$$

Mostre que a linguagem gerada por essa gramática é

$$\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) = \mathcal{N}_b(w) \text{ e } \mathcal{N}_a(v) \geq \mathcal{N}_b(v) \text{ para todo prefixo } v \text{ de } w\},$$

onde  $\mathcal{N}_a(w)$  é o número de  $a$ 's, que ocorrem em  $w$ . Sugestão: testar as seguintes cadeias:  $abaabb$ ,  $aabbaabb$  e  $ababab$  que fazem parte da linguagem e a cadeia  $aabbabba$  que não faz parte.

2. Desenhe a árvore de derivação correspondente ao exercício anterior.
3. Achar as gramáticas livres do contexto para as seguintes linguagens (com  $m \geq 1$ ,  $n \geq 1$ ,  $k \geq 1$  e  $p \geq 1$ ).
  - $\mathcal{L} = \{a^n b^m / n \neq m - 1\}$ ,
  - $\mathcal{L} = \{a^n b^m / n \neq 2m\}$ ,
  - $\mathcal{L} = \{a^n b^m / n \leq m + 3\}$ ,
  - $\mathcal{L} = \{a^n b^m / 2m \leq n + 1\}$ ,
  - $\mathcal{L} = \{a^n b^m c^k / n = m \text{ ou } m \leq k\}$ ,
  - $\mathcal{L} = \{a^n b^m c^k / n = k \text{ ou } n \neq m\}$ ,
  - $\mathcal{L} = \{a^n b^m c^k / n = m \text{ ou } m \neq k\}$ ,
  - $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \geq \mathcal{N}_b(w)\}$ ,
  - $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \text{ é ímpar }\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n / m = 2n + k\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n / 2k = n + m\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n / k = 2n + m + 2\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n / m = 2(n + k) + 1\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n / n \text{ é par e } k = m + \frac{n}{2}\}$ ,
  - $\mathcal{L} = \{a^k b^m a^n b^p / m = k \text{ ou } n = p\}$
  - $\mathcal{L} = \{a^k b^m a^n b^p / m \neq k \text{ ou } n = p\}$
4. Desenhe uma gramática livre de contexto que gere todas as expressões regulares válidas sobre o alfabeto  $\{a, b\}$ . Por exemplo deve permitir gerar as cadeias  $(a + aa^*b)^*$  +  $(\lambda + aa)(bb)^*b$  e  $(ab + ba)^*(aa + bb)^*$ .
5. Achar uma gramática livre do contexto para a linguagem

$$\mathcal{L} = \{a^n w w^R b^n / w \in \{a, b\}^*, n \geq 1\}$$

## 5.7. Exercícios

---

6. Defina uma gramática livre do contexto para a linguagem de todas cadeias no alfabeto  $\{a, b\}$  que não são palíndromos. Isto é, para a linguagem

$$\mathcal{L} = \{w \in \{a, b\}^* / w \neq w^R\}$$

7. Seja  $\mathcal{L} = \{a^n b^n / n \geq 0\}$

- (a) Mostre que  $\mathcal{L}^2$  é livre do contexto.
- (b) Mostre que  $\mathcal{L}^k$  é livre do contexto, para cada  $k \leq 1$ .

8. Mostre uma árvore de derivação para a cadeia  $aabbba$ , com a gramática

$$\begin{aligned} S &\longrightarrow AB \mid \lambda, \\ A &\longrightarrow aB, \\ B &\longrightarrow Sb. \end{aligned}$$

Dê uma descrição da linguagem gerada por essa gramática.

9. Considere a gramática com produções

$$\begin{aligned} S &\longrightarrow aaB, \\ A &\longrightarrow bBb \mid \lambda, \\ B &\longrightarrow Aa. \end{aligned}$$

Mostre que a cadeia  $aabbabba$  não está na linguagem gerada por tal gramática.

10. Achar uma gramática livre do contexto para o conjunto de todas as expressões regulares sobre o alfabeto  $\{a, b\}$ .

11. Mostre que a gramática seguinte é ambígua

$$\begin{aligned} S &\longrightarrow AB \mid aaB, \\ A &\longrightarrow a \mid Aa, \\ B &\longrightarrow b. \end{aligned}$$

12. Construa uma gramática não ambígua, equivalente à gramática do exercício anterior.

13. Encontre uma S-gramática para  $L(aaa^*b + b)$  e para  $\mathcal{L} = \{a^n b^n / n \geq 1\}$ .

14. Mostre que toda S-gramática é não-ambígua.

15. Mostre que a linguagem  $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}$  é livre do contexto.

16. Use o teorema 5.5.3, para remover produções esquerda-recursivas nas gramáticas

- (a)  $S \longrightarrow abS \mid SS \mid \lambda$
- (b)  $S \longrightarrow Sb \mid A,$   
 $A \longrightarrow Aa \mid a$

$$(c) \quad S \longrightarrow aZ \mid SbS, \\ Z \longrightarrow aZb \mid \lambda.$$

17. Dê uma gramática sem produções esquerda-recursivas para a linguagem

$$\mathcal{L} = \{w \in \{a, b\}^* \mid \mathcal{N}_a(w) > \mathcal{N}_b(w)\}$$

18. Elimine as produções inúteis na seguinte gramática

$$\begin{aligned} S &\longrightarrow a \mid aA \mid B \mid C, \\ A &\longrightarrow aB \mid \lambda, \\ B &\longrightarrow Aa, \\ C &\longrightarrow cCD, \\ D &\longrightarrow dd \mid ddD. \end{aligned}$$

19. Elimine as  $\lambda$ -produções de

$$\begin{aligned} S &\longrightarrow AaB \mid aCaB, \\ A &\longrightarrow aBa \mid \lambda \mid B, \\ B &\longrightarrow bbA \mid \lambda, \\ C &\longrightarrow bbbC \mid AB. \end{aligned}$$

20. Elimine as produções unitárias nas duas gramáticas anteriores.

21. Elimine produções unitárias,  $\lambda$ -produções, produções inúteis e produções esquerda-recursivas nas seguintes gramáticas livres do contexto:

- (a)  $S \longrightarrow Sa \mid aAB$   
 $A \longrightarrow B \mid aa \mid \lambda$   
 $B \longrightarrow AbB \mid \lambda \mid C$   
 $C \longrightarrow aCCa \mid ABC$
- (b)  $S \longrightarrow ASAS \mid B$   
 $A \longrightarrow aAa \mid \lambda$   
 $B \longrightarrow bB \mid C \mid Db$   
 $C \longrightarrow aCb \mid ab$   
 $D \longrightarrow Da \mid bD$
- (c)  $S \longrightarrow aA \mid aB \mid SaA$   
 $A \longrightarrow B \mid aa \mid \lambda$   
 $B \longrightarrow bB \mid \lambda \mid C$   
 $C \longrightarrow aCa \mid ABC$
- (d)  $S \longrightarrow AbAaBa \mid ABSa \mid aAAbC,$   
 $A \longrightarrow aAa \mid \lambda,$   
 $B \longrightarrow bbB \mid BCa \mid aD \mid \lambda,$   
 $C \longrightarrow cAE \mid aCCa,$

## 5.7. Exercícios

---

$$\begin{aligned} D &\longrightarrow E \mid F, \\ E &\longrightarrow aCb, \\ F &\longrightarrow FaF \mid FSaA, \\ G &\longrightarrow SaH, \\ H &\longrightarrow aA \mid \lambda. \end{aligned}$$

22. Suponha que  $G$  é uma gramática livre do contexto, para a qual  $\lambda \in L(G)$ . Mostre que se aplicarmos o teorema 5.5.10, obteremos uma nova gramática,  $\hat{G}$ , tal que  $L(\hat{G}) = L(G) - \{\lambda\}$ .
23. Converter as seguintes gramáticas à forma normal de Chomsky
- (a)  $S \longrightarrow aSb \mid ab$
  - (b)  $S \longrightarrow aSbA \mid aA,$   
 $A \longrightarrow abA \mid b.$
  - (c)  $S \longrightarrow abAB,$   
 $A \longrightarrow bAB \mid \lambda,$   
 $B \longrightarrow BAa \mid A \mid \lambda.$
  - (d)  $S \longrightarrow AaA \mid AbA,$   
 $A \longrightarrow AaC \mid bAAa \mid \lambda,$   
 $B \longrightarrow bC \mid A,$   
 $C \longrightarrow D \mid a \mid ba,$   
 $D \longrightarrow bAb \mid ba,$
  - (e)  $S \longrightarrow aA \mid aB \mid SaAB,$   
 $A \longrightarrow B \mid aa \mid \lambda,$   
 $B \longrightarrow bB \mid Ca \mid \lambda,$   
 $C \longrightarrow aCa \mid ABC,$
  - (f)  $S \longrightarrow SaS \mid ABAA,$   
 $A \longrightarrow bAAa \mid \lambda,$   
 $B \longrightarrow bCb \mid C,$   
 $C \longrightarrow aC \mid ba,$
  - (g)  $S \longrightarrow AaD \mid C,$   
 $A \longrightarrow AAab \mid bBa \mid \lambda,$   
 $B \longrightarrow bB \mid C \mid b,$   
 $C \longrightarrow B \mid aC,$   
 $D \longrightarrow baD \mid A$
  - (h)  $S \longrightarrow aC,$   
 $A \longrightarrow ACb \mid bA \mid \lambda,$   
 $B \longrightarrow bAa,$   
 $C \longrightarrow BA \mid ba$
24. Transformar as gramáticas resultantes do exemplo anterior à forma normal de Greibach.

25. Converter as seguintes gramáticas na forma normal de Greibach.

- (a)  $S \rightarrow aaS \mid SS \mid aa,$
- (b)  $S \rightarrow CA \mid CB,$   
 $A \rightarrow BB \mid BS \mid a,$   
 $B \rightarrow AA \mid CS \mid b,$   
 $C \rightarrow a \mid b,$
- (c)  $S \rightarrow aA \mid Bbb,$   
 $A \rightarrow aBa \mid aa,$   
 $B \rightarrow Abb \mid Cab,$   
 $C \rightarrow Aba \mid Bab \mid aba$
- (d)  $S \rightarrow SaS \mid AB,$   
 $A \rightarrow bAa \mid \lambda,$   
 $B \rightarrow ab \mid C,$   
 $C \rightarrow Ba.$

## **5.7. Exercícios**

---

# Capítulo 6

## Autômatos com Pilha

A descrição das linguagens livres do contexto, por meio de gramáticas livre do contexto é conveniente, como se observa no uso de BNF, em linguagens de programação. Uma pergunta é se existe uma classe de autômatos que possamos associar com as linguagens livres do contexto. Vimos, anteriormente, que os autômatos finitos não podem reconhecer todas as linguagens livres do contexto. Intuitivamente, entendemos que isso acontece devido aos autômatos finitos possuírem memórias estritamente finitas, enquanto o reconhecimento de linguagens livre do contexto podem requerer o armazenamento de uma quantidade ilimitada de informação. Por exemplo, quando verificamos uma cadeia da linguagem  $\mathcal{L} = \{a^n b^n / n \geq 0\}$ , devemos checar não somente que todos os  $a$ 's precede o primeiro  $b$ , mas também que a quantidade de  $a$ 's é a mesma que a de  $b$ 's, ou seja, precisamos contar. Como  $n$  é ilimitado, esta contagem não pode ser feita com uma memória finita. Assim, precisamos de uma máquina que possa contar sem limite. Mas, como podemos ver de exemplos como  $\{ww^R / w \in \Sigma^*\}$ , precisamos mais que a habilidade de contar sem limite: precisamos também da habilidade de armazenar e comparar uma seqüência de símbolos na ordem reversa. Isso sugere que tentemos uma pilha como mecanismo de armazenamento, permitindo armazenamento ilimitado restrito a operações como uma pilha. Isto nos fornece uma classe de máquinas chamadas autômatos com pilha.

### 6.1 Autômatos com Pilha Determinísticos (APD)

Na figura 6.1, é apresentado uma representação esquemática de um autômato com pilha.

Cada movimento da unidade de controle de um autômato com pilha lê um símbolo da fita de entrada, enquanto ao mesmo tempo troca o conteúdo da pilha através das operações usuais na pilha. Quando cada movimento da unidade de controle é completamente determinado pelo símbolo de entrada corrente assim como pelo símbolo no topo da pilha, o autômato com pilha é chamado de determinístico. Neste caso como resultado do movimento o estado da unidade de controle pode mudar assim como o topo da pilha pode ser substituído por uma cadeia de símbolos da pilha, mudando consequentemente o topo da pilha.

## 6.1. Autômatos com Pilha Determinísticos (APD)

---

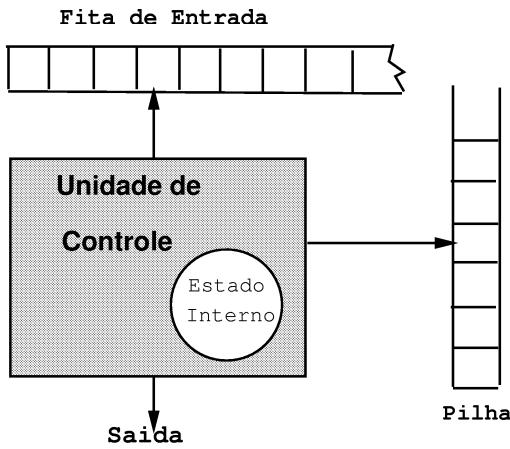


Figura 6.1: Representação esquemática de um autômato com pilha.

**Definição 6.1.1** Um autômato com pilha determinístico (APD) é definido pela sete-tupla  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$ , onde

- $Q$  é um conjunto finito de estados internos da unidade de controle,
- $\Sigma$  é um conjunto finito de símbolos, chamado o **alfabeto de entrada**,
- $\Gamma$  é o conjunto finito de símbolos, chamado o **alfabeto da pilha**,
- $q_0$  é o **estado inicial**,
- $z \in \Gamma$  é o **símbolo de início da pilha**,
- $F \subseteq Q$  é o **conjunto de estados finais**,
- $\delta : Q \times \Sigma \times \Gamma \longrightarrow Q \times \Gamma^*$  é a **função de transição**.

Esta definição de APD é diferente à usual, a qual pode ser encontrada em textos como [HU79] e [Har78], porém é equivalente, no sentido que uma linguagem é reconhecida por um APD (no nosso sentido) se e somente se é reconhecida por um APD no sentido usual. A vantagem desta nossa definição, é que ela estende de maneira natural a definição de AFD. Na subseção 6.4.1, veremos a definição usual de APD.

Dizer que  $z$  é o símbolo inicial da pilha e que  $q_0$  é o estado inicial, significa que o APD ao iniciar uma execução (com alguma cadeia de símbolos de  $\Sigma$  na fita de entrada), terá na pilha  $z$  como único símbolo e o estado interno será  $q_0$ .

Uma transição  $\delta(q_i, a, A) = (q_j, A_1 \dots A_n)$  indica que se o APD está no estado  $q_i$ , o topo da pilha é  $A$  e o símbolo da fita de entrada que o cabeçote de leitura lê for  $a$  então o estado do APD deve mudar para  $q_j$ , o topo ser substituído pela cadeia  $A_1 \dots A_n$  e, como em todo autômato, o cabeçote de leitura se deve deslocar uma célula para a direita. Colocar a cadeia  $A_1 \dots A_n$  substituindo o topo da pilha significa que  $A_n$  substitui o topo,  $A_{n-1}$  é colocado em cima de  $A_n$ ,  $A_{n-2}$  é colocado em cima de  $A_{n-1}$ , e assim por diante. Logo, o novo topo da pilha seria neste caso  $A_1$ .

Um APD **aceita uma cadeia** se após ler a cadeia toda o estado do APD é um estado final.

**Exemplo 6.1.2** O seguinte APD aceita qualquer cadeia da forma  $a^n b^n$  com  $n \geq 1$ .

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$ ,
  - $\Sigma = \{a, b\}$ ,
  - $\Gamma = \{A, B\}$ ,
  - $z = B$ ,
  - $F = \{q_3\}$  e
  - $\delta$  :
- |  |  |
|--|--|
| $\delta(q_0, a, B) = (q_1, B)$ ,       | $\delta(q_2, a, A) = (q_4, \lambda)$ , |
| $\delta(q_0, b, B) = (q_4, B)$ ,       | $\delta(q_2, b, A) = (q_2, \lambda)$ , |
| $\delta(q_0, a, A) = (q_4, \lambda)$ , | $\delta(q_3, a, B) = (q_4, B)$ ,       |
| $\delta(q_0, b, A) = (q_4, \lambda)$ , | $\delta(q_3, b, B) = (q_4, B)$ ,       |
| $\delta(q_1, a, B) = (q_1, AB)$ ,      | $\delta(q_3, a, A) = (q_4, \lambda)$ , |
| $\delta(q_1, b, B) = (q_3, B)$ ,       | $\delta(q_3, b, A) = (q_4, \lambda)$ , |
| $\delta(q_1, a, A) = (q_1, AA)$ ,      | $\delta(q_4, a, B) = (q_4, B)$ ,       |
| $\delta(q_1, b, A) = (q_2, \lambda)$ , | $\delta(q_4, b, B) = (q_4, B)$ ,       |
| $\delta(q_2, a, B) = (q_4, B)$ ,       | $\delta(q_4, a, A) = (q_4, \lambda)$ , |
| $\delta(q_2, b, B) = (q_3, B)$ ,       | $\delta(q_4, b, A) = (q_4, \lambda)$ . |

Note que para qualquer AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , o APD  $M_P = \langle Q, \Sigma, \Gamma, \delta_P, q_0, z, F \rangle$ , onde  $\Gamma = \{z\}$  e  $\delta_P(q, a, z) = \delta(q, a)$  para cada  $a \in \Sigma$  e  $q \in Q$ , reconhece a mesma linguagem que  $M$ , isto é:  $L(M) = L(M_P)$ . Assim, autômatos com pilha determinísticos reconhecem qualquer linguagem regular e como mostrado pelo exemplo 6.1.2 também reconhecem algumas linguagens livres do contexto que não são regulares. Porém, eles não reconhecem qualquer linguagem livre do contexto. Por exemplo, não reconhecem a linguagem

$$\mathcal{L}_P = \{ww^R / w \in \{0,1\}^*\}.$$

Um APD que reconheça esta linguagem deveria fazer o seguinte: a cada símbolo da primeira metade da cadeia (isto é de  $w$ ) deve-se pôr um símbolo na pilha (um  $A$  para cada 0 e um  $B$  para cada 1, por exemplo) sem mudar de estado, quando chegar o primeiro símbolo da segunda metade ( $w^R$ ) deve-se verificar se casa ou não com o topo da pilha (isto é, se o símbolo for 0 então o topo só pode ser  $A$ , e se for 1 então só pode ser  $B$ ). Se casar, então apaga o topo da pilha (o substitui pela cadeia vazia  $\lambda$ ) e muda de estado, se não vai para um estado de morte. Se casaram então o APD continua lendo nesse novo estado o seguinte símbolo da segunda metade e verifica se casa ou não com o topo da pilha, se eles casarem então apaga o topo da pilha e continua no mesmo estado, se não casarem então vai para um estado de morte, e assim por diante. Porém existem dois problemas com este “APD”: 1) quando ele vai para o estado final? e 2) como ele decide em que momento termina a primeira metade da cadeia e começa a segunda?.

A primeira destas objeções pode ser naturalmente permitindo o uso de  $\lambda$ -transições já a segunda poderia ser superada permitindo uma escolha não-determinística a cada passo. Na verdade, como veremos na seção 6.4, só bastaria uma de suas características adicionais.

## 6.2 Autômato com Pilha Não-Determinístico (APN)

**Definição 6.2.1** Um autômato com pilha não-determinístico (APN) é definido pela sete-tupla  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$ , onde

- $Q$  é um conjunto finito de estados interno da unidade de controle,
- $\Sigma$  é um conjunto finito de símbolos, chamado o **alfabeto de entrada**,
- $\Gamma$  é o conjunto finito de símbolos, chamado o **alfabeto da pilha**,
- $q_0$  é o **estado inicial**,
- $z$  é o **símbolo de início da pilha**,
- $F \subseteq Q$  é o **conjunto de estados finais**,
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \longrightarrow \wp^{\text{fin}}(Q \times \Gamma^*)$  é a **função de transição**.

Uma aplicação de  $\delta$  tem a seguinte interpretação: o argumento de  $\delta$  é o estado corrente da unidade de controle, o símbolo de entrada corrente, e o símbolo corrente no topo da pilha. O resultado é um conjunto de pares  $(q, x)$ , dentre os quais o APD fará uma escolha não-determinística de um deles. Se  $(q, x)$  for o escolhido então  $q$  será o próximo estado da unidade de controle e  $x$  será a cadeia posta no topo da pilha, no lugar do único símbolo que estava lá antes. Observe que o segundo argumento de  $\delta$  pode ser  $\lambda$ , indicando que o movimento não consome um símbolo de entrada. Denominaremos tal movimento  **$\lambda$ -transição**. Observe, também, que o  $\delta$  é definido de tal modo que ele precisa de um símbolo na pilha. Não é possível nenhum movimento se a pilha estiver vazia. Note, ademais que podemos diminuir a quantidade de símbolos na pilha se  $x$  for  $\lambda$ . Assim, suponha que o conjunto de regras de transição de um APN contém

$$\delta(q_1, a, x) = \{(q_2, yz), (q_3, \lambda)\}.$$

Se em um determinado momento a unidade de controle está no estado  $q_1$ , o símbolo de entrada lido é  $a$ , e o símbolo sobre o topo da pilha é  $x$ , então uma de duas coisas pode acontecer:

1. a unidade de controle vai para o estado  $q_2$  e a cadeia  $yz$  substitui  $x$  no topo da pilha, ou
2. a unidade de controle vai para o estado  $q_3$ , com o símbolo  $x$  removido do topo da pilha.

**Exemplo 6.2.2** Considere o APN com

- $Q = \{q_0, q_1, q_2, q_3\}$ ,
- $\Sigma = \{a, b\}$ ,
- $\Gamma = \{0, 1\}$ ,
- $z = 0$ ,

- $F = \{q_3\}$  e

- $\delta$  :

$$\begin{aligned}\delta(q_0, a, 0) &= \{(q_1, 10), (q_3, \lambda)\}, & \delta(q_1, b, 1) &= \{(q_2, \lambda)\}, \\ \delta(q_0, \lambda, 0) &= \{(q_3, \lambda)\}, & \delta(q_2, b, 1) &= \{(q_2, \lambda)\}, \\ \delta(q_1, a, 1) &= \{(q_1, 11)\}, & \delta(q_2, \lambda, 0) &= \{(q_3, \lambda)\}.\end{aligned}$$

*Que podemos afirmar sobre a ação deste autômato?*

Primeiro, observe que não são especificadas transições para todas as combinações possíveis de entrada e símbolos da pilha. Por exemplo,  $\delta(q_0, b, 0)$  não está definida. A interpretação deste fato é o mesmo usado para autômato finito não-determinístico: uma transição não especificada vai para o conjunto vazio e representa uma configuração de morte do APN. Ou seja,  $\delta(q_0, b, 0) = \emptyset$ .

As transições cruciais são  $\delta(q_1, a, 1) = \{(q_1, 11)\}$  que adiciona 1 à pilha quando é lido um  $a$ , e  $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$ , o qual remove um 1 quando um  $b$  é encontrado. Essas duas etapas contam o número de  $a$ 's e compara com o número de  $b$ 's. A unidade de controle está no estado  $q_1$  até ser encontrado o primeiro  $b$  quando ele entra no estado  $q_2$ . Isto assegura que nenhum  $b$  precede o último  $a$ . Após analisar as transições restantes, veremos que o APN terminará no estado final  $q_3$  se e somente se a cadeia de entrada está na linguagem

$$\mathcal{L} = \{a^n b^n / n \geq 0\} \cup \{a\}.$$

*Como uma analogia com autômatos finitos, devemos dizer que o APN aceita a linguagem acima.*

Para melhor visualizar a função de transição  $\delta$ , esta será mostrada como uma tabela, onde a primeira coluna é rotulada com os pares estado corrente-símbolo no topo da pilha, e a primeira fila com os símbolos de entrada junto com a entrada  $\lambda$ . Cada elemento  $(i, j)$  da matriz que não esteja na primeira coluna (portanto,  $j > 1$ ) nem na primeira fila (portanto,  $i > 1$ ), representa a saída da função quando avaliada no estado corrente, símbolo do topo da pilha (descritos na posição  $(i, 1)$  da matriz) e para a entrada descrita na posição  $(1, j)$  da matriz. Para simplificar a notação, usaremos  $(q_i, a_i), (q_j, a_j), \dots, (q_k, a_k)$  em vez de  $\{(q_i, a_i), (q_j, a_j), \dots, (q_k, a_k)\}$  e não escreveremos nada quando a saída for  $\emptyset$ . Assim, por exemplo, a função de transição  $\delta$  do exemplo anterior seria representada pela tabela

	$a$	$b$	$\lambda$
$(q_0, 0)$	$(q_1, 10), (q_3, \lambda)$		$(q_3, \lambda)$
$(q_1, 1)$	$(q_1, 11)$	$(q_2, \lambda)$	
$(q_2, 1)$		$(q_2, \lambda)$	
$(q_2, 0)$			$(q_3, \lambda)$

Para simplificar a discussão, introduziremos uma notação conveniente para descrever as configurações sucessivas de um APN durante o processamento de uma cadeia. Os fatores relevantes em qualquer momento após um passo computacional do APN são o estado corrente da unidade de controle, a parte não lida da cadeia de entrada, e o conteúdo corrente da pilha. Juntos

## 6.2. Autômato com Pilha Não-Determinístico (APN)

---

esses elementos determinam completamente todas as maneiras possíveis no qual um APN pode proceder. A tripla

$$(q, w, u),$$

onde  $q$  é o estado corrente da unidade de controle,  $w$  é a parte não lida da cadeia e  $u$  é o conteúdo da pilha (com o símbolo mais à esquerda indicando o topo da pilha) é chamado uma **descrição instantânea** do autômato com pilha. Um movimento de uma descrição instantânea para outro será denotado pelo símbolo  $\vdash$ , portanto

$$(q_1, aw, bu) \vdash (q_2, w, yu)$$

é possível se e somente se

$$(q_2, y) \in \delta(q_1, a, b).$$

O movimento envolvendo um número arbitrário de etapas será denotado por  $\vdash^*$  e  $\vdash^+$ , este último indicando que, no mínimo, um movimento é feito. Quando vários autômatos estão sob consideração usaremos  $\vdash_M$  para enfatizar que o movimento é feito pelo autômato particular  $M$ .

Se  $\delta(q, a, A) = \emptyset$  para algum  $a \in \Sigma$  e  $A \in \Gamma$  então para qualquer  $w \in \Sigma^*$  e  $v \in \Gamma^*$  não existe  $q'$  nem  $y \in \Gamma^*$  tal que

$$(q, aw, Av) \vdash (q', w, yv)$$

Denotaremos isto por,  $(q, aw, Av) \vdash \perp$ . Analogamente, se  $(q, w, v) \vdash^* (q', w', v')$  e  $(q', w', v') \vdash \perp$  então

$$(q, w, v) \vdash^* \perp.$$

**Exemplo 6.2.3** Tome o APN do exemplo anterior. Então

$$(q_0, aaabbb, 0) \vdash (q_3, aabbb, \lambda),$$

e como não tem nenhum movimento possível o APN pararia, isto é  $(q_0, aaabbb, 0) \vdash^* \perp$ . Mesmo  $q_3$  sendo um estado final essa computação não nós permitiria concluir que a cadeia aaabbb é aceita pelo autômato, pois não consumiu toda a cadeia de entrada.

É claro que também poderia ter ocorrido o seguinte:

$$(q_0, aaabbb, 0) \vdash (q_1, aabbb, 10) \vdash (q_1, abbb, 110) \vdash (q_1, bbb, 1110) \vdash (q_2, bb, 110) \vdash (q_2, b, 10) \vdash (q_2, \lambda, 0) \vdash (q_3, \lambda, \lambda)$$

Aqui sim, o APN pára no estado final mas só após ter consumido toda a cadeia. Por tanto o APN reconheceria aaabbb.

### 6.2.1 A Linguagem Aceita por um Autômato com Pilha Não-Determinístico

Assim como os autômatos finitos reconhecem linguagens (regulares), cada autômato com pilha reconhece, também, uma linguagem.

**Definição 6.2.4** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um autômato com pilha não-determinístico. A linguagem aceita por  $M$  é o conjunto

$$L(M) = \{w \in \Sigma^* / (q_0, w, z) \vdash^* (q_f, \lambda, u), \text{ para algum } q_f \in F \text{ e } u \in \Gamma^*\}.$$

Assim, uma **cadeia é reconhecida** por  $M$  somente se está em  $L(M)$ .

Em palavras, a linguagem aceita por  $M$  é o conjunto de todas as cadeias sobre o alfabeto  $\Sigma$  para as quais  $M$ , começando no estado inicial  $q_0$  e com a pilha contendo somente o símbolo  $z$ , pára num estado final após consumir a cadeia toda. Note que o fato de existir um situação em que  $(q_0, w, z) \vdash^* (q_i, \lambda, u)$  para algum  $q_i \notin F$  e  $u \in \Gamma^*$  ou que  $(q_0, w, z) \vdash^* \perp$  não impede que  $(q_0, w, z) \vdash^* (q_f, \lambda, v)$  para algum  $q_f \in F$ . Ou seja, para  $w \in L(M)$  basta que exista pelo menos um caminho que leve com êxito a um estado final após consumir todos os símbolos da fita de entrada.

**Exemplo 6.2.5** Construir uma APN para a linguagem

$$\mathcal{L} = \{a^{n+1}b^n / n \geq 0\}.$$

Como no exemplo anterior, para cada a lido, colocaremos um símbolo na pilha, de tal modo que quando começamos a ler os b's, retiremos cada um desses símbolos da pilha. Porém como te-se mais um a que b's, só começaremos a colocar símbolos na pilha a partir do segundo a lido. Se ao terminar de ler a cadeia estiver a pilha com o símbolo de início da pilha (que é diferente do colocado quando lê um a) então vamos para um estado final. Logo, definamos o APN  $M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{0, 1\}, \delta, q_0, 0, \{q_3\} \rangle$ , onde  $\delta$  é definido pela seguinte tabela:

	$a$	$b$	$\lambda$
$(q_0, 0)$	$(q_1, 0)$		
$(q_1, 0)$	$(q_1, 10)$		$(q_3, 0)$
$(q_1, 1)$	$(q_1, 11)$	$(q_2, \lambda)$	
$(q_2, 1)$		$(q_2, \lambda)$	
$(q_2, 0)$			$(q_3, 0)$

Quando  $M$  processa a cadeia aaabb, faz os seguintes movimentos:

$$(q_0, aaabb, 0) \vdash (q_1, aabb, 0) \vdash (q_1, abb, 10) \vdash (q_1, bb, 110) \vdash (q_2, b, 10) \vdash (q_2, \lambda, 0) \vdash (q_3, \lambda, 0)$$

e, portanto, a cadeia é aceita. Claramente, este APN aceita a linguagem  $\mathcal{L}$ , isto é,  $L(M) = \mathcal{L}$ .

## 6.2. Autômato com Pilha Não-Determinístico (APN)

---

**Exemplo 6.2.6** Construir um APN para a linguagem

$$\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}.$$

Como anteriormente, a solução para este problema envolve contar o número de a's e b's, o que é feito com uma pilha. Não precisamos aqui nos preocupar com a ordem dos a's e b's. Podemos inserir um símbolo de contagem, digamos 0, na pilha se o a for lido, e então tirar um símbolo da pilha quando o b for lido. A única dificuldade com isto é se existe um prefixo de w com mais b's que a's, não acharemos um 0 para retirar. Para isso deveremos usar um símbolo de contagem negativa, digamos 1, para contar os b's que estão sendo lidos além dos a's. Assim, quando vier novos a's retiramos estes 1's da pilha até ficar novamente equiparada às quantidades. A solução completa é um APN

$$M = \langle \{q_0, q_1\}, \{a, b\}, \{0, 1, z\}, \delta, q_0, z, \{q_1\} \rangle,$$

com  $\delta$  definido por

	a	b	$\lambda$
$(q_0, z)$	$(q_0, 0z)$	$(q_0, 1z)$	$(q_1, z)$
$(q_0, 0)$	$(q_0, 00)$	$(q_0, \lambda)$	
$(q_0, 1)$	$(q_0, \lambda)$	$(q_0, 11)$	

Em processando a cadeia baab, o APN faz o seguinte movimento

$$(q_0, baab, z) \vdash (q_0, aab, 1z) \vdash (q_0, ab, z) \vdash (q_0, b, 0z) \vdash (q_0, \lambda, z) \vdash (q_1, \lambda, z),$$

e portanto a cadeia é aceita.

**Exemplo 6.2.7** Construir um APN que aceite a linguagem

$$\mathcal{L} = \{ww^R / w \in \{a, b\}^*\}.$$

Usamos o fato de que os símbolos são recuperados na pilha na ordem reversa de sua inserção. Quando lemos a primeira parte da cadeia, colocamos os símbolos consecutivos na pilha. Para a segunda parte, comparamos o símbolo de entrada corrente com o topo da pilha, continuando enquanto os dois casarem. Como os símbolos são recuperados da pilha na ordem reversa que foram colocados, um casamento completo se dá se e somente se a entrada é da forma  $ww^R$ .

Uma aparente dificuldade ao se usar esta sugestão é que não sabemos o meio da cadeia, isto é, quando w termina e quando começa  $w^R$ . Mas, a natureza não-determinística do autômato nos ajuda. O APN adivinha corretamente qual é o meio e muda de estado nesse ponto. Uma solução para o problema é dado por

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle,$$

onde  $Q = \{q_0, q_1, q_2\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{A, B, z\}$ ,  $F = \{q_2\}$ . A função de transição pode ser visualizada como tendo várias partes: um conjunto para colocar  $w$  na pilha,

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_0, Az)\}, & \delta(q_0, b, A) &= \{(q_0, BA)\}, \\ \delta(q_0, b, z) &= \{(q_0, Bz)\}, & \delta(q_0, a, B) &= \{(q_0, AB)\}, \\ \delta(q_0, a, A) &= \{(q_0, AA)\}, & \delta(q_0, b, B) &= \{(q_0, BB)\},\end{aligned}$$

um conjunto para adivinhar o meio da cadeia, onde o APN muda de estado de  $q_0$  para  $q_1$ ,

$$\delta(q_0, \lambda, A) = \{(q_1, A)\}, \quad \delta(q_0, \lambda, B) = \{(q_1, B)\},$$

um conjunto para comparar  $w^R$  com o conteúdo da pilha,

$$\delta(q_1, a, A) = \{(q_1, \lambda)\}, \quad \delta(q_1, b, B) = \{(q_1, \lambda)\},$$

e, finalmente,

$$\delta(q_1, \lambda, z) = \{(q_2, \lambda)\},$$

para reconhecer o casamento bem sucedido.

Esta função de transição pode ser melhor visualizada na tabela

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_0, Bz)$	$(q_2, \lambda)$
$(q_0, A)$	$(q_0, AA)$	$(q_0, BA)$	$(q_1, A)$
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB)$	$(q_1, B)$
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, B)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, \lambda)$

A seqüência de movimentos que aceita  $abba$  é

$$(q_0, abba, z) \vdash (q_0, bba, Az) \vdash (q_0, ba, BAz) \vdash (q_1, ba, BAz) \vdash (q_1, a, Az) \vdash (q_1, \lambda, z) \vdash (q_2, \lambda, \lambda).$$

A alternativa não-determinística para localizar o meio da cadeia é tomada no terceiro movimento. Naquele estágio, o APN tem a descrição instantânea  $(q_0, ba, BAz)$  e tem escolhas para o próximo movimento. Uma é usar  $\delta(q_0, b, b) = \{(q_0, BB)\}$  e fazer o movimento

$$(q_0, ba, BAz) \vdash (q_0, a, BBAz).$$

O segundo é o movimento que usamos, ou seja, usar  $\delta(q_0, \lambda, b) = \{(q_1, b)\}$ . Somente este último leva à aceitação da entrada.

## 6.2. Autômato com Pilha Não-Determinístico (APN)

---

Até o momento não usamos o não-determinismo puro, ou seja uma situação em que  $|\delta(q, a, A)| \geq 2$  para algum  $a \in \Sigma$  e  $A \in \Gamma$ . O seguinte exemplo, mostra uma linguagem onde aparentemente uma tal situação seria essencial.

**Exemplo 6.2.8** Seja a linguagem

$$\mathcal{L} = \{a^n b^m \mid 1 \leq n \leq m \leq 3n\}.$$

Cada  $a$  em uma cadeia de  $\mathcal{L}$  tem o peso de um, dois ou três  $b$ 's. Por exemplo, uma cadeia com três  $a$ 's pode ter entre três a nove  $b$ 's. Ou seja se a cadeia for  $aaabbbbb$  então o primeiro  $a$  poderia casar com os dois últimos  $b$ 's (lembre que a ordem da retirada, pela estrutura da pilha, é a reversa), o segundo  $a$  com o segundo e terceiro  $b$  e o último  $a$  com o primeiro  $b$ . Uma outra possibilidade, seria que o primeiro  $a$  case com o último  $b$ , o segundo com penúltimo  $b$  e o último  $a$  com os três primeiros  $b$ 's. Ou seja haveria várias formas de casar as  $a$ 's com  $b$ 's a escolha de quantos  $b$  casa um  $a$  pode ser feita deterministicamente. Assim um APN para esta linguagem seria

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_2\} \rangle,$$

com  $\delta$  definido por

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az), (q_0, AAz), (q_0, AAAz)$		
$(q_0, A)$	$(q_0, AA), (q_0, AAA), (q_0, AAAA)$	$(q_1, \lambda)$	
$(q_1, A)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, z)$

Assim, o reconhecimento da cadeia  $aaabbbbb$  pode ser obtida como segue:

$$\begin{aligned} (q_0, aaabbbbb, z) &\vdash (q_0, aabbbb, Az) \vdash (q_0, abbbb, AAz) \vdash (q_0, bbbb, AAAAz) \\ &\vdash (q_1, bbbb, AAAAz) \vdash (q_1, bbb, AAAz) \vdash (q_1, bb, AAz) \vdash (q_1, b, Az) \\ &\vdash (q_1, \lambda, z) \vdash (q_2, \lambda, z), \end{aligned}$$

ou por

$$\begin{aligned} (q_0, aaabbbbb, z) &\vdash (q_0, aabbbb, AAz) \vdash (q_0, abbbb, AAAz) \vdash (q_0, bbbb, AAAAz) \\ &\vdash (q_1, bbbb, AAAAz) \vdash (q_1, bbb, AAAz) \vdash (q_1, bb, AAz) \vdash (q_1, b, Az) \\ &\vdash (q_1, \lambda, z) \vdash (q_2, \lambda, z). \end{aligned}$$

## 6.3 Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto

Nos exemplos da seção anterior, vimos que os autômatos com pilha existem para algumas linguagens familiares livres do contexto. Isto não é coincidência. Existe uma relação geral entre as linguagens livres do contexto e os autômatos com pilha não-determinísticos. Veremos que a toda linguagem livre do contexto está associado um APN que a aceita e, inversamente, que toda linguagem aceita por um APN é livre do contexto.

### 6.3.1 Transformando Gramáticas Livre do Contexto para Autômatos com Pilha Não-Determinísticos

Primeiro, mostraremos que para toda linguagem livre do contexto existe um APN que a aceita. A idéia subjacente é construir um APN que possa, em algum sentido, efetuar uma derivação mais a esquerda de qualquer cadeia na linguagem. Para simplificar um pouco o argumento, assumiremos que a linguagem é gerada por uma gramática onde o lado direito de cada produção tem a seguinte forma

$$A \longrightarrow aw$$

parta algum  $A \in V$ ,  $a \in T \cup \{\lambda\}$  e  $w \in V^*$ . Observe que a única diferença entre esta forma normal e a de Greibach, é que aqui o lado direito pode ser composto por uma cadeia de variáveis, enquanto que na forma normal de Greibach o símbolo mais à esquerda sempre é um símbolo terminal.

Para obter esta nova forma normal, que chamaremos aqui de **forma normal pseudo-Greibach**, basta associar a cada símbolo terminal  $a$  uma variável, digamos  $B_a$ , e substituir toda ocorrência de um símbolo terminal  $a$  no lado direito de uma produção, mas desde que não seja o símbolo mais à esquerda, pela variável  $B_a$  e ao final desse processo adicionar as produções  $B_a \longrightarrow a$  para cada  $a \in T$ . Claramente, a forma normal pseudo-Greibach obtida desta forma é equivalente à original. Portanto, toda linguagem livre do contexto pode ser gerada por uma gramática na forma normal pseudo-Greibach.

O APN que vamos construir terá  $T$  como alfabeto de entrada e  $V \cup \{z\}$  como alfabeto da pilha, com  $z \notin V$  sendo o símbolo de início da pilha. Representaremos a derivação mantendo as variáveis na parte direita da forma sentencial como a cadeia que substituirá a variável no topo da pilha, enquanto o terminal da parte esquerda é identificado com o símbolo de entrada que é lido. Sendo assim, para simular a aplicação de uma produção  $A \longrightarrow ax$ , devemos ter a variável  $A$  no topo da pilha e o terminal  $a$  como símbolo de entrada. A variável sobre a pilha é removida e trocada pela cadeia de variáveis  $x$ . O que  $\delta$  deve atingir é fácil de ver. Antes de apresentar o argumento geral, veremos um exemplo simples.

**Exemplo 6.3.1** Construir um APN que aceite a linguagem gerada pela seguinte gramática:

### 6.3. Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto

---

$$\begin{aligned} S &\longrightarrow ABC \\ A &\longrightarrow aAa \mid a, \\ B &\longrightarrow bbB \mid bb, \\ C &\longrightarrow abC \mid \lambda \end{aligned}$$

Primeiro obtemos a forma normal pseudo-Greibach:

$$\begin{aligned} S &\longrightarrow ABC \\ A &\longrightarrow aAB_a \mid a, \\ B &\longrightarrow bB_bB \mid bB_b, \\ C &\longrightarrow aB_bC \mid \lambda, \\ B_a &\longrightarrow a, \\ B_b &\longrightarrow b. \end{aligned}$$

O autômato correspondente terá três estados,  $\{q_0, q_1, q_2\}$ , com estado inicial  $q_0$  e estado final  $\{q_2\}$ . Primeiro, o símbolo de início  $S$  é posto sobre a pilha por

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}.$$

A produção  $S \longrightarrow ABC$  será simulada no APN removendo  $S$  da pilha e trocando com  $ABC$  sem consumir qualquer símbolo da fita de entrada, ou seja aplicando uma  $\lambda$ -transição. A produção  $A \longrightarrow aAB_a$  deve possibilitar ao APN ler um  $a$  e substituir  $A$  por  $AB_a$ . Já a produção  $A \longrightarrow a$  deve simplesmente permitir ao APN consumir  $a$  e remover  $A$  do topo da pilha. Analogamente, o APN para simular as produções  $B \longrightarrow bB_bB$  e  $B \longrightarrow bB_b$  deverá ler o símbolo  $b$  da fita de entrada e substituir o  $B$  no topo da pilha por  $B_bB$  e  $B_b$ , respectivamente. Analogamente para simular as produção  $C \longrightarrow aB_bC$  o APN deve ler  $a$  e trocar o  $C$  no topo da pilha por  $B_bC$ . Já para simular  $C \longrightarrow \lambda$  o APN deve apagar  $C$  do topo da pilha sem consumir qualquer símbolo, ou seja aplicando uma  $\lambda$ -transição. Analogamente, para o APN simular as produções  $B_a \longrightarrow a$  e  $B_b \longrightarrow b$  deve apagar  $B_a$  e  $B_b$ , respectivamente, do topo da pilha sem consumir qualquer símbolo da fita de entrada.

Portanto, as produções da gramática na forma normal pseudo-Greibach são representadas no APN por

$$\begin{aligned} \delta(q_1, \lambda, S) &= \{(q_1, ABC)\} \\ \delta(q_1, a, A) &= \{(q_1, AB_a), (q_1, \lambda)\} \\ \delta(q_1, b, B) &= \{(q_1, B_bB), (q_1, B_b)\} \\ \delta(q_1, a, C) &= \{(q_1, B_bC)\} \\ \delta(q_1, \lambda, C) &= \{(q_1, \lambda)\} \\ \delta(q_1, a, B_a) &= \{(q_1, \lambda)\} \\ \delta(q_1, b, B_b) &= \{(q_1, \lambda)\} \end{aligned}$$

O aparecimento do símbolo de início da pilha no topo da pilha sinaliza a completação da derivação e o APN é posto no estado final por

$$\delta(q_1, \lambda, z) = \{(q_2, z)\}.$$

A construção, nesse exemplo, pode ser generalizada para outros casos, levando para um resultado geral.

**Teorema 6.3.2** *Para qualquer linguagem livre do contexto  $\mathcal{L}$  existe um APN  $M$  tal que*

$$\mathcal{L} = L(M).$$

**DEMONSTRAÇÃO:** Se  $\mathcal{L}$  é uma linguagem livre do contexto, livre de  $\lambda$ , existe uma gramática livre do contexto para ela na forma normal pseudo-Greibach. Seja  $G = \langle V, T, S, P \rangle$  tal gramática. Construiremos então um APN que simulará as derivações mais à esquerda nesta gramática. Como foi sugerido, a simulação será feita tal que a parte não processada da forma sentencial estará na pilha, enquanto o prefixo terminal de qualquer forma sentencial casa com o correspondente prefixo da cadeia de entrada (o símbolo de entrada corrente).

Especificamente o APN será

$$M = \langle \{q_0, q_1, q_2\}, T, V \cup \{z\}, \delta, q_0, z, \{q_2\} \rangle,$$

onde  $z \notin V$ . Observe que o alfabeto de entrada de  $M$  é idêntico ao conjunto de terminais de  $G$  e que o alfabeto da pilha contém o conjunto das variáveis da gramática (mais o símbolo de início da pilha). A função de transição incluirá

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\} \quad (6.1)$$

de modo que após o primeiro movimento de  $M$ , a pilha contém o símbolo de início  $S$  da derivação (o símbolo de início  $z$  da pilha é uma marca que nos permite detectar o fim da derivação). Além disso, o conjunto de regras de transição é tal que

$$(q_1, w) \in \delta(q_1, a, A), \quad (6.2)$$

sempre que  $A \longrightarrow aw$  está em  $P$  para algum  $A \in V$ ,  $a \in T \cup \{\lambda\}$  e  $w \in V^*$ . Isto é, se  $a \in T$ , lê a entrada  $a$  e remove a variável  $A$  do topo da pilha, trocando-a pela cadeia de variáveis  $w$ . Desse modo ela gera as transições que permite o APN simular todas as derivações. Caso  $a \in \lambda$  então sem consumir qualquer símbolo de entrada, substitui  $A$  no topo da pilha por  $w$ . Finalmente, temos

$$\delta(q_1, \lambda, z) = \{(q_2, z)\}, \quad (6.3)$$

para colocar  $M$  no estado final.

Para mostrar que  $M$  aceita qualquer  $w \in L(G)$ , considere a derivação parcial mais à esquerda

$$S \xrightarrow{*} a_1 a_2 \cdots a_n A_1 A_2 \cdots A_m \implies a_1 a_2 \cdots a_n b B_1 \cdots B_k A_2 \cdots A_m.$$

Se  $M$  simula esta derivação, então após ler  $a_1 a_2 \cdots a_n$ , a pilha deve conter  $A_1 A_2 \cdots A_m$ . Para tomar o próximo passo na derivação,  $G$  deve ter a produção

### 6.3. Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto

---

$$A_1 \longrightarrow bB_1 \cdots B_k.$$

mas a construção é tal que  $M$  tem uma regra de transição na qual

$$(q_1, B_1 \cdots B_k) \in \delta(q_1, b, A_1),$$

e tal que a pilha agora contém  $B_1 \cdots B_k A_2 \cdots A_m$ , após ter lido  $a_1 \cdots a_n b$ .

Um argumento de indução simples sobre o número de etapas na derivação mostra que se  $S \xrightarrow{*} w$ , então

$$(q_1, w, Sz) \vdash^* (q_1, \lambda, z).$$

Usando 6.1 e 6.3 temos

$$(q_0, w, z) \vdash (q_1, w, Sz) \vdash^* (q_1, \lambda, z) \vdash (q_2, \lambda, z),$$

de modo tal que  $L(G) \subseteq L(M)$ .

Para provar que  $L(M) \subseteq L(G)$ , seja  $w \in L(M)$ . Por definição

$$(q_0, w, z) \vdash^* (q_2, \lambda, u).$$

Mas existe somente uma maneira para ir de  $q_0$  para  $q_1$  e somente uma de  $q_1$  para  $q_2$ . Portanto, devemos ter

$$(q_1, w, Sz) \vdash^* (q_1, \lambda, z).$$

Agora, vamos escrever  $w = a_1 a_2 \cdots a_n$ . Então a primeira etapa em

$$(q_1, a_1 a_2 \cdots a_n, Sz) \vdash^* (q_1, \lambda, z) \tag{6.4}$$

deve ser uma regra da forma (6.2) para obter

$$(q_1, a_1 a_2 \cdots a_n, Sz) \vdash (q_1, a_2 a_3 \cdots a_n, u_1 z).$$

Mas, então, a gramática tem uma regra da forma  $S \longrightarrow a_1 u_1$ , tal que  $S \xrightarrow{*} a_1 u_1$ .

Repetindo isto, escrevendo  $u_1 = A u_2$ , temos

$$(q_1, a_2 \cdots a_n, A u_2 z) \vdash (q_1, a_3 \cdots a_n, u_3 u_2 z),$$

implicando que  $A \longrightarrow a_2 u_3$  está na gramática e que

$$S \xrightarrow{*} a_1 a_2 u_3 u_2.$$

Isto torna claro que em qualquer ponto o conteúdo da pilha (excluindo  $z$ ) é idêntico à parte não casada da forma sentencial, de modo que ( 6.4) implica

$$S \xrightarrow{*} a_1 a_2 \cdots a_n.$$

Em consequência,  $L(M) \subseteq L(G)$ , completando a prova. ■

**Exemplo 6.3.3** Considere a gramática

$$\begin{aligned} S &\longrightarrow aA, \\ A &\longrightarrow aABC \mid bB \mid a, \\ B &\longrightarrow b, \\ C &\longrightarrow c. \end{aligned}$$

Uma vez que a gramática já está na forma normal de Greibach, podemos usar as construções do teorema, acima, imediatamente.

Além das regras

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\} \quad e \quad \delta(q_1, \lambda, z) = \{(q_2, z)\},$$

o APN terá as regras de transição.

$$\begin{aligned} \delta(q_1, a, S) &= \{(q_1, A)\}, & \delta(q_1, b, A) &= \{(q_1, B)\}, \\ \delta(q_1, a, A) &= \{(q_1, ABC)\}, \quad \delta(q_1, b, B) = \{(q_1, \lambda)\}, \\ \delta(q_1, c, C) &= \{(q_1, \lambda)\}. \end{aligned}$$

Ao processar  $aaabc$ ,  $M$  faz a seguinte seqüência de movimentos

$$\begin{aligned} (q_0, aaabc, z) \vdash (q_1, aaabc, Sz) \vdash (q_1, aabcAz) \vdash (q_1, abc, ABCz) \vdash \\ (q_1, bc, BCz) \vdash (q_1, c, Cz) \vdash (q_1, \lambda, z) \vdash (q_2, \lambda, z). \end{aligned}$$

Isto corresponde à derivação

$$S \implies aA \implies aaABC \implies aaaBC \implies aaabC \implies aaabc.$$

### 6.3.2 Transformando Autômatos com Pilha Não-Determinísticos em Gramáticas Livre do Contexto

A inversa do teorema 6.3.2 é também verdadeira. A construção é reverter o processo de construção deste teorema, de modo que a gramática simule os movimentos do APN. Isto significa que o conteúdo da pilha deve estar refletida na parte das variáveis da forma sentencial, enquanto a entrada processada é o prefixo terminal da forma sentencial. Para fazer isso é preciso um pouco de detalhe.

De modo a manter a discussão tão simples quanto possível, assumiremos que o APN em discussão satisfaz as seguintes condições:

### 6.3. Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto

---

1. Ele tem um único estado final no qual o autômato só entra nele se e somente se a pilha estiver vazia.
2. Todas as transições devem ter a forma  $\delta(q_i, a, A) = \{C_1, C_2, \dots, C_n\}$ , onde

$$C_i = (q_j, \lambda) \quad (6.5)$$

ou

$$C_i = (q_j, BC). \quad (6.6)$$

Isto é, cada movimento ou aumenta ou diminui o conteúdo da pilha de um único símbolo.

Essas restrições podem parecer muito severas, o que não é verdade. É possível mostrar que para qualquer APN existe um equivalente, no sentido que reconhecem a mesma linguagem, satisfazendo as condições 1. e 2. Ou seja, essas restrições constituem uma **forma normal para APN**.

Transformar um APN num APN equivalente satisfazendo a primeira condição é trivial, uma vez que se há mais de um estado final, é suficiente tornar os estados finais em não-finais, criar um novo estado final  $q_{nf}$  e para cada antigo estado final  $q_f$  e para cada símbolo  $A$  da pilha adicionar a seguinte transição:

$$\delta(q_f, \lambda, A) = (\{q_f\}, \lambda) \text{ se } A \neq z \text{ e}$$

$$\delta(q_f, \lambda, z) = (\{q_{nf}\}, \lambda).$$

Já uma transição que não satisfaz 2., ou é da forma  $\delta(q_i, a, A) = (q_j, A_1 \dots A_n)$  para algum  $n > 2$ , em cujo caso pode ser trocada pelas transições:

$$\delta(q_i, a, A) = (q_{j1}, XA_n), \delta(q_{j1}, \lambda, X) = (q_{j2}, XA_{n-1}), \dots, \delta(q_{j(n-1)}, \lambda, X) = (q_j, A_1A_2)$$

onde  $q_{j1}, \dots, q_{jn}$  são estados novos e  $X \in \Gamma - \{z\}$ , ou é da forma  $\delta(q_i, a, A) = (q_j, B)$ , em cujo caso pode ser trocada pelas transições:

$$\delta(q_i, a, A) = (q_{j1}, BB) \text{ e } \delta(q_{j1}, \lambda, B) = (q_j, \lambda)$$

onde  $q_{j1}$  é um novo estado.

**Exemplo 6.3.4** Seja a linguagem  $\mathcal{L} = \{a^n b^m \mid 2n \leq m \leq 3n\}$ . Um APN que reconhece  $\mathcal{L}$  é descrito a seguir:

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, AAz)(q_0, AAAz)$		
$(q_0, A)$	$(q_0, AAA)(q_0, AAAA)$	$(q_1, \lambda)$	
$(q_1, A)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, \lambda)$

onde  $q_2$  é o estado final.

Este autômato não está na forma normal para APN, mas transformaremos eles usando o algoritmo anterior. Note que ele já satisfaz a primeira condição, pelo que só aplicaremos a segunda parte do algoritmo, resultando no seguinte APN:

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_a, Az)(q_b, Az)$		
$(q_a, A)$			$(q_0, AA)$
$(q_b, A)$			$(q_c, AA)$
$(q_c, A)$			$(q_0, AA)$
$(q_0, A)$	$(q_d, AA)(q_e, AA)$	$(q_1, \lambda)$	
$(q_d, A)$			$(q_0, AA)$
$(q_e, A)$			$(q_f, AA)$
$(q_f, A)$			$(q_0, AA)$
$(q_1, A)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, \lambda)$

Obviamente, este autômato pode ser enxugado. Por exemplo, o seguinte APN que também está na forma normal é equivalente ao APN resultante do algoritmo e portanto equivalente ao original:

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_a, Az)(q_b, Az)$		
$(q_a, A)$			$(q_0, AA)$
$(q_b, A)$			$(q_a, AA)$
$(q_0, A)$	$(q_a, AA)(q_b, aA)$	$(q_1, \lambda)$	
$(q_1, A)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, \lambda)$

Construiremos, agora, a gramática livre do contexto para a linguagem aceita pelo APN. Sem perda de generalidade, podemos assumir que o APN está na forma normal para APN.

Para iniciar, devemos ter uma forma sentencial para indicar o conteúdo da pilha, mas observemos que a configuração de um APN também envolve um estado interno que deve ser relembrado na forma sentencial. Para isso, devemos usar variáveis na gramática da forma  $(q_i A q_j)$ , com a interpretação de que  $(q_i A q_j) \xrightarrow{*} w$  se e somente se o APN apaga  $A$  da pilha, indo do estado  $q_i$  para o  $q_j$ , enquanto lê a cadeia  $w$ . “Apagar”, aqui, significa que  $A$  e seus efeitos após (isto é, a cadeia pela qual é trocada) desaparece da fita, trazendo o símbolo originalmente abaixo de  $A$  para o topo.

Usando esta interpretação, não é difícil ver que cada produção da gramática precisa corresponder com um desses dois tipos de transições. Como ( 6.5) envolve uma “remoção” imediata de  $A$ , a gramática terá a correspondente produção

$$(q_i A q_j) \longrightarrow a.$$

### 6.3. Equivalência entre Autômatos com Pilha Não-Determinísticos e Gramáticas Livre do Contexto

---

Produções do tipo ( 6.6) geram a produção

$$(q_i A q_l) \longrightarrow a(q_j B q_k)(q_l C q_l),$$

onde  $q_k$  e  $q_l$  tomam todos os valores possíveis em  $Q$ . Isto é devido ao fato de que para apagar  $A$  primeiramente lemos  $a$ , trocamos  $A$  com  $BC$  e vamos para o estado  $q_j$ . Logo, devemos apagar  $B$  e depois  $C$ . Podemos fazer isto indo até um estado  $q_k$  e apagar  $B$ , ou seja,  $(q_j B q_k)$  e depois indo até um estado  $q_l$  e apagar  $C$ , ou seja,  $(q_l C q_l)$ . Como não sabemos exatamente que estados são esses  $q_k$  e  $q_l$ , colocamos todos.

Na última etapa, pode parecer que adicionamos em demasia, uma vez que podem existir estados  $q_k$  que não podem ser alcançados por  $q_j$  enquanto apaga  $B$ . Isto é verdade, mas isto não afeta a gramática. Neste caso, a variável resultante  $(q_j B q_k)$  é inútil e não afeta a linguagem aceita pela gramática. De fato podemos aplicar o algoritmo de remoção de produções inúteis para tornar a gramática mais enxuta.

Finalmente, como variável de início tomamos  $(q_0 z q_f)$ , onde  $q_f$  é o único estado final do APN.

**Exemplo 6.3.5** Considere o APN  $M = \langle \{q_0, \dots, q_3\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_2\} \rangle$  onde

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_0, Az)\}, \\ \delta(q_3, \lambda, z) &= \{(q_0, Az)\}, \\ \delta(q_0, a, A) &= \{(q_3, \lambda)\}, \\ \delta(q_0, b, A) &= \{(q_1, \lambda)\}, \\ \delta(q_1, \lambda, z) &= \{(q_2, \lambda)\}.\end{aligned}$$

As três últimas são da forma ( 6.5) de modo que elas geram as produções correspondentes.

$$(q_0 A q_3) \longrightarrow a, \quad (q_0 A q_1) \longrightarrow b, \quad (q_1 z q_2) \longrightarrow \lambda.$$

A partir das duas primeiras transições obtemos o seguinte conjunto de produções.

$$\begin{array}{l|l|l|l|l}(q_0 z q_0) \longrightarrow a(q_0 A q_0)(q_0 z q_0) & | & a(q_0 A q_1)(q_1 z q_0) & | & a(q_0 A q_2)(q_2 z q_0) & | & a(q_0 A q_3)(q_3 z q_0), \\ (q_0 z q_1) \longrightarrow a(q_0 A q_0)(q_0 z q_1) & | & a(q_0 A q_1)(q_1 z q_1) & | & a(q_0 A q_2)(q_2 z q_1) & | & a(q_0 A q_3)(q_3 z q_1), \\ (q_0 z q_2) \longrightarrow a(q_0 A q_0)(q_0 z q_2) & | & a(q_0 A q_1)(q_1 z q_2) & | & a(q_0 A q_2)(q_2 z q_2) & | & a(q_0 A q_3)(q_3 z q_2), \\ (q_0 z q_3) \longrightarrow a(q_0 A q_0)(q_0 z q_3) & | & a(q_0 A q_1)(q_1 z q_3) & | & a(q_0 A q_2)(q_2 z q_3) & | & a(q_0 A q_3)(q_3 z q_3), \\ (q_3 z q_0) \longrightarrow (q_0 A q_0)(q_0 z q_0) & | & (q_0 A q_1)(q_1 z q_0) & | & (q_0 A q_2)(q_2 z q_0) & | & (q_0 A q_3)(q_3 z q_0), \\ (q_3 z q_1) \longrightarrow (q_0 A q_0)(q_0 z q_1) & | & (q_0 A q_1)(q_1 z q_1) & | & (q_0 A q_2)(q_2 z q_1) & | & (q_0 A q_3)(q_3 z q_1), \\ (q_3 z q_2) \longrightarrow (q_0 A q_0)(q_0 z q_2) & | & (q_0 A q_1)(q_1 z q_2) & | & (q_0 A q_2)(q_2 z q_2) & | & (q_0 A q_3)(q_3 z q_2), \\ (q_3 z q_3) \longrightarrow (q_0 A q_0)(q_0 z q_3) & | & (q_0 A q_1)(q_1 z q_3) & | & (q_0 A q_2)(q_2 z q_3) & | & (q_0 A q_3)(q_3 z q_3).\end{array}$$

Eliminando produções inúteis temos a seguinte gramática livre do contexto:

$$\begin{aligned}
 (q_0 z q_2) &\longrightarrow a(q_0 A q_1)(q_1 z q_2) \mid a(q_0 A q_3)(q_3 z q_2), \\
 (q_3 z q_2) &\longrightarrow (q_0 A q_1)(q_1 z q_2) \mid (q_0 A q_3)(q_3 z q_2), \\
 (q_0 A q_3) &\longrightarrow a, \\
 (q_0 A q_1) &\longrightarrow b, \\
 (q_1 z q_2) &\longrightarrow \lambda.
 \end{aligned}$$

A variável de início será  $(q_0 z q_2)$ . A cadeia  $aab$  é aceita pelo APN, com as sucessivas configurações

$$\begin{aligned}
 (q_0, aab, z) &\vdash (q_0, ab, Az) \\
 &\vdash (q_3, b, z) \\
 &\vdash (q_0, b, Az) \\
 &\vdash (q_1, \lambda, z) \\
 &\vdash (q_2, \lambda, \lambda).
 \end{aligned}$$

A correspondente derivação com  $G$  é

$$\begin{aligned}
 (q_0 z q_2) &\implies a(q_0 A q_3)(q_3 z q_2) \\
 &\implies aa(q_3 z q_2) \\
 &\implies aa(q_0 A q_1)(q_1 z q_2) \\
 &\implies aab(q_1 z q_2) \\
 &\implies aab.
 \end{aligned}$$

As etapas na prova do seguinte teorema serão fáceis de entender se se observar a correspondência entre as sucessivas descrições instantâneas do APN e as formas sentenciais na derivação. O primeiro  $q_i$ , na variável mais à esquerda de cada forma sentencial é o estado corrente do APN, enquanto a seqüência de símbolos do meio é a mesma do conteúdo da pilha.

Embora a construção gere uma gramática muito complicada, ela pode ser aplicada a qualquer APN cujas regras de transição satisfazem as condições dadas.

**Teorema 6.3.6** Se  $\mathcal{L} = L(M)$  para algum APN  $M$ , então  $\mathcal{L}$  é uma linguagem livre do contexto.

**DEMONSTRAÇÃO:** Seja  $\widehat{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, z, \{q_f\} \rangle$  a forma normal da APN  $M$ . Assim,  $\widehat{M}$  satisfaz as condições (6.5) e (6.6). Construiremos uma gramática  $G = \langle V, \Sigma, (q_0 z q_f), P \rangle$ , onde  $V$  consiste dos elementos da forma  $(q_i c q_j)$  e para cada

$$(q_j, BC) \in \delta(q_i, a, A)$$

incluímos em  $P$  a produção

$$(q_i A q_l) \longrightarrow a(q_j B q_k)(q_k C q_l),$$

para cada  $q_k, q_l \in Q$ . Por outro lado, para cada

## 6.4. $\lambda$ -Transições vs Não-determinismo Explícito

---

$$(q_j, \lambda) \in \delta(q_i, a, A)$$

incluímos em  $P$  a produção

$$(q_i A q_j) \longrightarrow a.$$

Mostraremos que a gramática assim obtida é tal que para cada  $q_i, q_j \in Q$ ,  $A \in \Gamma$ ,  $x \in \Gamma^*$  e  $u, v \in \Sigma^*$ ,

$$(q_i, uv, Ax) \vdash^* (q_j, v, x), \text{ se e somente se } (q_i A q_j) \xrightarrow{*} u.$$

Pela maneira como foram construídas as produções de  $G$ , podemos, facilmente, mostrar isto. Basta aplicar uma indução no tamanho da seqüência de movimentos, para obter a implicação, e outra no tamanho da derivação, para obter a inversa. Assim, em particular, podemos concluir que

$$(q_0, w, z) \vdash^* (q_f, \lambda, \lambda), \text{ se e somente se } (q_0 z q_f) \xrightarrow{*} w,$$

onde  $q_f \in F$ . Portanto,  $L(M) = L(G)$ . ■

## 6.4 $\lambda$ -Transições vs Não-determinismo Explícito

Harrison em [Har78] mostrou que para qualquer APN  $P$  é possível obter um APN sem  $\lambda$ -transições  $P'$ , tal que  $P$  e  $P'$  reconhecem a mesma linguagem. Ou seja, ele mostrou que  $\lambda$ -transições não acrescentam poder ao modelo de autômatos com pilha.

Por outro lado, observe que a necessidade de usar não-determinismo puro ou explícito no primeiro APN no exemplo 6.3.4 é só aparente, perfeitamente poderíamos definir um APN onde o  $\delta$  não contenha qualquer escolha não-determinística explícita, ou seja  $|\delta(q, a, A)| \leq 1$  para todo  $a \in \Sigma \cup \{\lambda\}$  e  $A \in \Gamma$ .

Um tal APN seria

$$M = \langle \{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_4\} \rangle,$$

com  $\delta$  definido por

	$a$	$b$	$\lambda$
$(q_0, z)$			$(q_1, Az)$
$(q_0, A)$		$(q_3, \lambda)$	$(q_1, AA)$
$(q_1, A)$	$(q_0, AA)$		$(q_2, AA)$
$(q_2, A)$	$(q_0, AA)$		
$(q_3, A)$		$(q_3, \lambda)$	
$(q_3, z)$			$(q_4, z)$

Dizer que uma transição é não-determinística é dizer que ela possui mais de uma opção. Formalmente,

**Definição 6.4.1** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um APN. Para cada  $(q, a, A) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ ,  $\delta(q, a, A)$  é explicitamente não-determinística se  $|\delta(q, a, A)| \geq 2$ . O **grau de não-determinismo explícito** de  $M$ , denotado por  $\deg(M)$ , é

$$\deg(M) = \max\{|\delta(q, a, A)| / (q, a, A) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma\}.$$

**Definição 6.4.2** Um APN  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  é **livre de não-determinismo explícito** se  $\deg(M) = 1$

A seguir veremos que não-determinismo explícito, isto é quando uma transição permite mais de uma opção, também pode ser eliminado sem diminuir o poder do modelo. Ou seja, podemos usar  $\lambda$ -transições para simular não-determinismo.

**Teorema 6.4.3** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um APN. Então existe um apn livre de não-determinismo explícito  $M'$  tal que  $L(M) = L(M')$ .

**DEMONSTRAÇÃO:** Uma vez que Harrison em [Har78] mostrou como retirar  $\lambda$ -transições de um APN sem alterar a linguagem aceita por ele, podemos assumir sem perda de generalidade, que  $M$  não possui  $\lambda$ -transições. Também sem perda de generalidade podemos assumir que os estados do APN são seqüenciais, ou seja vão de  $q_0$  ate  $q_{n-1}$ , onde  $n = |Q|$ .

O seguinte algoritmo mostra a forma como podemos transformar um APN  $M$  qualquer num APN livre de não-determinismo explícito  $M'$  equivalente.

Começo

$Q' := Q$ ;

Enquanto  $\deg(M) \geq 2$  Faça

Começo

Para cada  $q \in Q$  Faça

Começo

$n := |Q'|$ ;

$m := 0$ ;

Se  $|\delta(q, a, A)| \geq 2$  para algum  $a \in \Sigma$  e  $A \in \Gamma$  Então

Começo

$m := 1$ ;

$\delta'(q, \lambda, A) := \{(q_n, A)\}$ ;

Para cada  $b \in \Sigma$  Faça

Começo

$(q', w') := \varsigma(\delta(q, b, A))$ ; %% onde  $\varsigma$  é uma função escolha

$\delta'(q, b, A) := \{(q', w')\}$ ;

$\delta'(q_n, b, A) := \delta(q, b, A) - \{(q', w')\}$ ;

Para cada  $B \in \Gamma - \{A\}$  Faça  $\delta'(q_n, b, B) := \emptyset$ ;

Fim

Para cada  $A \in \Gamma$  Faça  $\delta'(q_n, \lambda, A) := \emptyset$ ;

Fim

## 6.4. $\lambda$ -Transições vs Não-determinismo Explícito

Senão Para cada  $b \in \Sigma$  Faça  $\delta'(q, b, A) := \delta(q, b, A)$ ;  
 Se  $m = 1$  Então  $Q' := Q' \cup \{q_n\}$ ;

Fim

$Q := Q'$ ;

$\delta := \delta'$ ;

Fim

Fim

A figura 6.2 mostra como age este algoritmo.

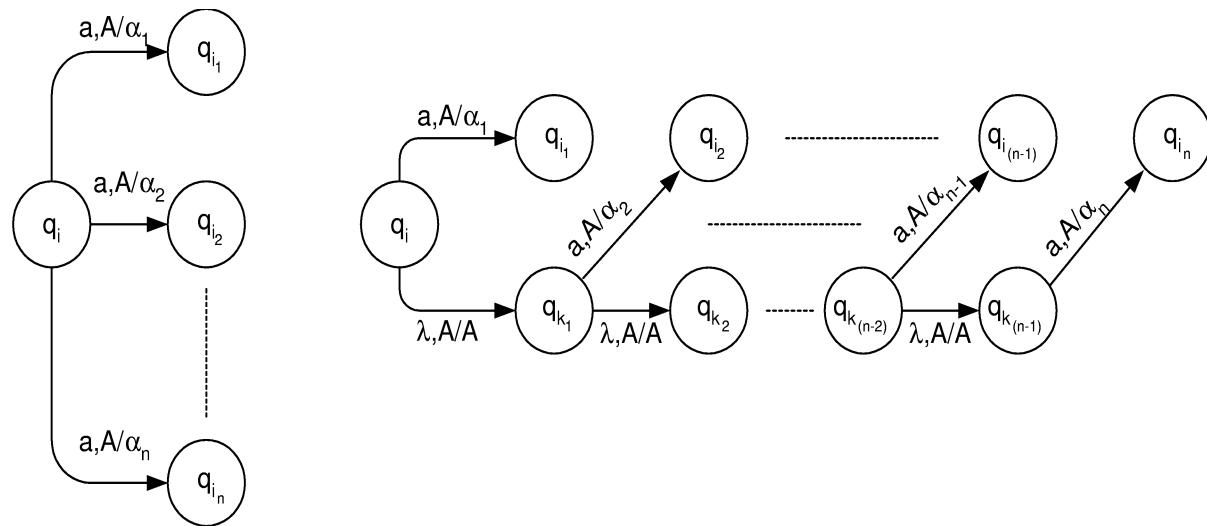


Figura 6.2: Esquema básico da transformação de um APN em um APN livre de não-determinismo explícito.

Note que a cada final do bloco de programa do primeiro “Para cada”,  $M' = \langle Q', \Sigma, \Gamma, \delta', q_0, z, F \rangle$  satisfaz

1. Se  $|\delta'(q, a, A)| \geq 2$  para algum  $(q, a, A) \in Q' \times \Sigma \times \Gamma$  então  $\delta'(q, \lambda, A) = \emptyset$  e
2.  $\deg(M') = \deg(M) - 1$ .

Uma vez que  $M$  nesse ponto é  $M'$  anterior, a segunda propriedade garante que o algoritmo sempre pára.

Falta provar que o  $M'$  ao final da execução do programa e o  $M$  do inicio são equivalentes, mas deixamos para o leitor se convencer disso. ■

Observe que o APN livre de não-determinismo explícito construído no início desta seção a partir do APN do exemplo 6.2.8 não seguiu fielmente este algoritmo. Para isso precisaríamos partir de uma APN livre de  $\lambda$ -transições.

**Exemplo 6.4.4** Por exemplo,

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

delta	a	b	λ
( $q_0, z$ )	( $q_0, z$ ), ( $q_0, Az$ ), ( $q_0, AAz$ )		
( $q_0, A$ )	( $q_0, AA$ ), ( $q_0, AAA$ ), ( $q_0, AAAA$ )	( $q_1, \lambda$ )	
( $q_1, A$ )		( $q_1, \lambda$ )	
( $q_1, z$ )		( $q_2, \lambda$ )	

Claramente,

$$\deg(M) = 3 \text{ e } \mathcal{L}(M) = \{a^n b^m \mid 1 \leq n \leq m \leq 3n\}.$$

Ou seja é equivalente ao APN do exemplo 6.2.8. Seguindo fielmente o algoritmo obteríamos o seguinte APN livre de não-determinismo explícito:

$$M' = \langle \{q_0, \dots, q_6\}, \{a, b\}, \{A, z\}, \delta', q_0, z, \{q_2\} \rangle, \text{ onde}$$

$\delta'$	a	b	$\lambda$
( $q_0, z$ )	( $q_0, z$ )		( $q_3, z$ )
( $q_0, A$ )	( $q_0, AA$ )	( $q_1, \lambda$ )	( $q_3, A$ )
( $q_1, A$ )		( $q_1, \lambda$ )	
( $q_1, z$ )		( $q_2, \lambda$ )	
( $q_3, z$ )	( $q_0, Az$ )		( $q_4, z$ )
( $q_3, A$ )	( $q_0, AAA$ )		( $q_4, A$ )
( $q_4, z$ )	( $q_0, AAz$ )		
( $q_4, A$ )	( $q_0, AAAA$ )		

#### 6.4.1 Medidas de Não-Determinismo Implícito em APN's Livres de Não-determinismo Explícito

Assim, o APN  $M'$  do exemplo 6.4.4 não contém não-determinismo explícito. Porém, uma vez que reconhece uma linguagem intrinsecamente não-determinística, o não-determinismo de alguma forma continua dentro de  $M'$ , só que de uma forma oculta.

**Definição 6.4.5** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um APN livre de não-determinismo explícito.  $\delta(q, a, A)$  é uma **transição determinística** se

$$|\delta(q, \lambda, A) \cup \delta(q, a, A)| \leq 1. \quad (6.7)$$

$M$  é um **autômato com pilha determinístico** (APD) se para cada  $(q, a, A) \in Q \times \Sigma \times \Gamma$ ,  $\delta(q, a, A)$  é uma transição determinística.

## 6.4. $\lambda$ -Transições vs Não-determinismo Explícito

---

Observe que:

1. A condição 6.7 poderia ser re-escrita como: “Se  $\delta(q, \lambda, A) \neq \emptyset$  então  $\delta(q, a, A) = \emptyset$ ”. Portanto esta definição coincide com a definição usual de APD; e
2. Esta definição de APD e a definição 6.1.1 são equivalentes, no sentido que aceitam a mesma classe de linguagens (as linguagens livres do contexto determinísticas).

**Definição 6.4.6** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um APD livre de não-determinismo explícito e  $a \in \Sigma$ . Dizemos que  $\delta(q, a, A)$  é uma **transição implicitamente não-determinística** se  $\delta(q, a, A) \neq \emptyset$  e existe  $q' \in Q - \{q\}$  tal que  $\delta(q', a, B) \neq \emptyset$  e já seja  $(q, \lambda, A) \vdash^*_M (q', \lambda, Bu)$  ou  $(q', \lambda, B) \vdash^*_M (q, \lambda, Au)$  para algum  $B \in \Gamma$  e  $u \in \Gamma^*$ . Neste caso, dizemos que  $\delta(q', a, B)$  é uma **transição não-determinística co-implícita** de  $\delta(q, a, A)$ .

Para compreendermos melhor esta definição, considere o caso onde  $\delta(q, a, A) = \{(q_i, w)\}$ ,  $\delta(q, \lambda, A) = \{(q', Bv)\}$  e  $\delta(q', a, B) = \{(q_j, u)\}$ . Portanto,  $(q, ax, Aw') \vdash_M (q_i, x, ww')$  e  $(q, ax, Aw') \vdash_M (q', ax, Bvw') \vdash_M (q_j, x, uvw')$ . Assim, a partir da descrição instantânea  $(q, ax, Aw')$  nos teremos uma escolha não-determinística quando  $a$  for lido. Porém, se  $\delta(q, \lambda, A) = \emptyset$  ou  $\delta(q', a, B) = \emptyset$  então nos teríamos uma única possibilidade de escolha a partir de  $(q, ax, Aw')$  quando  $a$  for lido:  $(q, ax, Aw') \vdash_M (q_i, x, ww')$ , uma vez que a outra possibilidade ( $(q, ax, Aw') \vdash_M (q', ax, Bvw')$ ) não seria aplicável e portanto a transição  $\delta(q, a, A)$  seria determinística.

**Definição 6.4.7** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, z, F \rangle$  um APN livre de não-determinismo explícito. A quantidade de **não-determinismo implícito** de  $M$ , denotado por  $\mu(M)$ , é

$$\mu(M) = |\{(q, a, A) \in Q \times \Sigma \times \Gamma / \delta(q, a, A) \text{ é uma transição implicitamente não-determinística}\}|.$$

Claramente, um APN livre de não-determinismo explícito  $M$  é determinístico se e somente se  $\mu(M) = 0$ . Observe que isto não implica que não tenha  $\lambda$ -transições, mas implica que caso se tenha uma  $\lambda$ -transição para um estado  $q$  e símbolo da pilha  $A$  (isto é,  $\delta(q, \lambda, A) \neq \emptyset$ ) então necessariamente  $\delta(q, a, A) = \emptyset$  para cada  $a \in \Sigma$ . A medida de não-determinismo implícito foi introduzida e usada por [Bed07] para descrever uma hierarquia infinita de classes de linguagens as quais variam da classe das linguagens livres do contexto determinísticas até as livres do contexto. Obviamente, essa não é a única forma de se estabelecer uma hierarquia dessa natureza, e de fato na literatura existem outros trabalhos sobre este assunto, veja por exemplo [GLW05, Her97, SY94, VS81].

## 6.5 Exercícios

1. Construa um APN que aceite a linguagem de todos os palíndromos no alfabeto  $\{a, b\}$ . Um palíndromo é uma cadeia que quando lida da esquerda à direita é a mesma que quando lida da direita à esquerda.
2. Dado um AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , descreva como construir, diretamente, um APN equivalente a ele.
3. Construa APN's que reconheçam as seguintes linguagens, sobre  $\Sigma = \{a, b\}$  (considere  $k, m, n, p \geq 1$ )
  - (a)  $\mathcal{L} = \{a^n b^{2n} / n \geq 0\}$
  - (b)  $\mathcal{L} = \{a^m b^n / n \leq m \leq 3n\}$
  - (c)  $\mathcal{L} = \{a^m b^n / m \leq n \leq 3m\}$
  - (d)  $\mathcal{L} = \{a^m b^n a^{2m+1} / n, m \geq 1\}$
  - (e)  $\mathcal{L} = \{a^m b^n / n = m + 1 \text{ ou } m = n + 1\}$
  - (f)  $\mathcal{L} = \{w \in \{a, b\}^* / N_a(w) \text{ é múltiplo de } 3\}$
  - (g)  $\mathcal{L} = \{w \in \Sigma^* / N_a(w) = N_b(w) + 1\}$
  - (h)  $\mathcal{L} = \{w \in \Sigma^* / N_a(w) = 2N_b(w)\}$
  - (i)  $\mathcal{L} = \{w \in \Sigma^* / 2N_a(w) \leq N_b(w) \leq 3N_a(w)\}$
  - (j)  $\mathcal{L} = \{a^m b^n / 2m = 3n\}$
  - (k)  $\mathcal{L} = \{a^m b^n / n = \lfloor \frac{m}{2} \rfloor\}$
  - (l)  $\mathcal{L} = \{a^m b^n / n = \lceil \frac{m}{2} \rceil\}$
  - (m)  $\mathcal{L} = \{a^k b^m a^n b^p / k = 2n \text{ ou } m = 2p\}$
  - (n)  $\mathcal{L} = \{a^k b^m a^n b^p / n = 2k \text{ ou } p = 2m\}$
  - (o)  $\mathcal{L} = \{a^k b^m a^n b^p / k = m \text{ ou } n \neq p\}$
  - (p)  $\mathcal{L} = \{a^k b^m a^n b^p / k = m \text{ ou } n \neq p\}$
  - (q)  $\mathcal{L} = \{a^k b^m a^n b^p / k - m = n - p\}$
4. Construa APN's que reconheçam as seguintes linguagens sobre  $\Sigma = \{a, b, c\}$ 
  - (a)  $\mathcal{L} = \{wcw^R / w \in \{a, b\}^*\}$
  - (b)  $\mathcal{L} = \{a^m b^n c^{m+n} / n \geq 0, m \geq 0\}$
  - (c)  $\mathcal{L} = \{a^m b^{m+n} c^n / n \geq 0, m \geq 0\}$
  - (d)  $\mathcal{L} = \{w \in \Sigma^* / N_a(w) \leq N_b(w) - N_c(w)\}$
  - (e)  $\mathcal{L} = \{w \in \Sigma^* / N_a(w) + N_b(w) = N_c(w)\}$
  - (f)  $\mathcal{L} = \{a^k b^m c^n / m \text{ é par e } k = n + \frac{m}{2}\}$
  - (g)  $\mathcal{L} = \{a^k b^m c^n / m = 2n + k - 1\}$
  - (h)  $\mathcal{L} = \{a^k b^m c^n / k = 2n + m + 2\}$

## 6.5. Exercícios

---

5. Que linguagem é aceita pelo APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_1, A), (q_2, \lambda)\}, \\ \delta(q_1, b, A) &= \{(q_1, B)\}, \\ \delta(q_1, b, B) &= \{(q_1, B)\}, \\ \delta(q_1, a, B) &= \{(q_2, \lambda)\} ?\end{aligned}$$

6. Que linguagem aceita o APN anterior se  $F = \{q_0, q_1, q_2\}$ ?  
 7. Mostre que o APN construído no exemplo 6.3.1 aceita a cadeia aaabbbb.  
 8. Prove que o APN construído no exemplo 6.3.1 reconhece a linguagem

$$\mathcal{L} = \{a^{n+1}b^{2n} / n \geq 0\}.$$

9. Construa um APN que reconheça a linguagem gerada pelas seguintes gramáticas livres de contexto:
- (a)  $S \rightarrow aABB \mid aAA,$   
 $A \rightarrow aBB \mid a,$   
 $B \rightarrow bBB \mid A.$
  - (b)  $S \rightarrow AA \mid a,$   
 $A \rightarrow SA \mid b.$
  - (c)  $S \rightarrow baB \mid aABA \mid ab,$   
 $A \rightarrow aA \mid a,$   
 $B \rightarrow bBb \mid baAbB \mid \lambda$
  - (d)  $S \rightarrow AaB \mid SaS,$   
 $A \rightarrow AAa \mid B,$   
 $B \rightarrow bBb \mid \lambda$

10. No exemplo 6.3.5 mostre que as variáveis  $(q_0Bq_0)$  e  $(q_0zq_1)$  são inúteis.  
 11. Encontre uma gramática livre do contexto que gere a linguagem aceita pelo APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$		
$(q_0, A)$	$(q_1, \lambda)$	$(q_0, AA)$	
$(q_1, A)$			$(q_1, \lambda)$
$(q_1, z)$			$(q_2, \lambda)$

12. Encontre uma gramática livre do contexto que gere a linguagem aceita pelo APN

$$M = \langle \{q_0, \dots, q_3\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_23\} \rangle, \text{ onde}$$

	<i>a</i>	<i>b</i>	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_0, Bz)$	
$(q_0, A)$	$(q_0, AA)(q_1, \lambda)$	$(q_0, BB)$	
$(q_0, B)$	$(q_0, AA)$	$(q_0, BB)(q_1, \lambda)$	
$(q_1, A)$	$(q_1, \lambda)$	$(q_2, A)$	
$(q_1, B)$	$(q_2, B)$	$(q_1, \lambda)$	
$(q_2, A)$		$(q_2, \lambda)$	
$(q_2, B)$	$(q_2, \lambda)$		
$(q_2, z)$			$(q_3, \lambda)$

13. Seja o seguinte APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

	<i>a</i>	<i>b</i>	$\lambda$
$(q_0, z)$	$(q_0, Xz) (q_2, z)$	$(q_0, Yz) (q_2, z)$	
$(q_0, X)$	$(q_0, AX) (q_1, X) (q_2, \lambda)$	$(q_0, BX)$	
$(q_0, A)$	$(q_0, AA) (q_1, A) (q_1, \lambda)$	$(q_0, BA)$	
$(q_0, Y)$	$(q_0, AY)$	$(q_0, BY) (q_1, Y) (q_2, \lambda)$	
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB) (q_1, B) (q_1, \lambda)$	
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, X)$	$(q_2, \lambda)$		
$(q_1, B)$		$(q_1, \lambda)$	
$(q_1, Y)$		$(q_2, \lambda)$	

Observe que  $M$  reconhece a linguagem de todos os palíndromos diferentes de  $\lambda$ .

Determine o grau de não-determinismo explícito de  $M$  e construa, usando o algoritmo do teorema 6.4.3 um APN livre de não-determinismo explícito equivalente a  $M$ .

14. Seja o seguinte APN

$$M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_3\} \rangle, \text{ onde}$$

	<i>a</i>	<i>b</i>	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_0, Az)$	
$(q_0, A)$	$(q_0, AA)(q_1, \lambda)(q_2, A)$	$(q_0, AA)(q_1, A)(q_2, \lambda)$	
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, z)$			$(q_3, \lambda)$
$(q_2, A)$		$(q_2, \lambda)$	
$(q_2, z)$			$(q_3, \lambda)$

## 6.5. Exercícios

---

Observe que  $M$  reconhece a seguinte linguagem

$$L(M) = \{uv \in \{a,b\}^* \mid \mathcal{N}_a(u) + \mathcal{N}_b(u) = \mathcal{N}_a(v) \text{ ou } \mathcal{N}_a(u) + \mathcal{N}_b(u) = \mathcal{N}_b(v)\}$$

Determine o grau de não-determinismo explícito de  $M$  e construa, usando o algoritmo do teorema 6.4.3 um APN livre de não-determinismo explícito equivalente a  $M$ .

15. Demonstre que para todo APD segundo a definição 6.1.1 existe um APD segundo a definição 6.4.5 que aceita a mesma linguagem e vice-versa.

## Capítulo 7

# Propriedades das Linguagens Livres do Contexto

As linguagens livres do contexto ocupam uma posição central na hierarquia das linguagens formais. Por um lado, as linguagens livres do contexto incluem importantes, mas restritas, famílias de linguagens como as linguagens regulares e as linguagens de programação usuais pertencem a esta classe de linguagens. Por outro lado, existem famílias de linguagens mais amplas das quais as linguagens livres do contexto são casos especiais. Como anteriormente, para as linguagens regulares, veremos as propriedades de fecho para várias operações, algoritmos para determinar propriedades de membros da família, e resultados estruturais como os lemas do bombeamento.

### 7.1 O Lema do Bombeamento para Linguagens Livres do contexto

O lema do bombeamento dado para linguagens regulares é uma ferramenta efetiva para mostrar que certas linguagens não são regulares. Existem lemas do bombeamento análogos para outras classes de linguagens, veja por exemplo [Lin90] um lema do bombeamento para linguagens lineares e em [Yu89] um lema do bombeamento para linguagens livres do contexto determinísticas, isto é, reconhecidas por APD's<sup>1</sup>. Porém, aqui, discutiremos um desses lemas de bombeamento para linguagens livres do contexto, em geral.

**Teorema 7.1.1 (Lema do bombeamento)** *Seja  $\mathcal{L}$  uma linguagem livre do contexto infinita. Então existe algum inteiro positivo  $m$  tal que qualquer  $w \in \mathcal{L}$ , com  $|w| \geq m$ , pode ser decomposto como  $w = uvxyz$  com  $u, v, x, y, z \in \Sigma^*$  satisfazendo*

$$|vxy| \leq m, \tag{7.1}$$

$$|vy| \geq 1 \tag{7.2}$$

---

<sup>1</sup>Também existem outras ferramentas para este mesmo propósito, veja por exemplo [HU79, LV95].

## 7.1. O Lema do Bombeamento para Linguagens Livres do contexto

---

e

$$uv^i xy^i z \in \mathcal{L}, \text{ para todo } i = 0, 1, 2, \dots \quad (7.3)$$

DEMONSTRAÇÃO: Ver [Lin90]. ■

**Exemplo 7.1.2** Seja a linguagem

$$\mathcal{L} = \{a^n b^{2n} / n \geq 1\}.$$

Mostraremos usando o lema do bombeamento que  $\mathcal{L}$  é livre do contexto.

Considere,  $m = 3$ . Então todo  $w \in \mathcal{L}$  satisfaz  $|w| \geq m$ . Seja,  $n \geq 1$ .

Claramente,  $u = a^{n-1}$ ,  $v = a$ ,  $x = \lambda$ ,  $y = bb$  e  $z = b^{2(n-1)}$  é uma decomposição de  $w$  satisfazendo as condições (7.1) e (7.2) do lema. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = a^{n-1} a^i (bb)^i b^{2(n-1)} = a^{n-1+i} b^{2i+2(n-1)} = a^{n-1+i} b^{2(n-1+i)} \in \mathcal{L}$$

Logo,  $\mathcal{L}$  é livre do contexto.

Porém, este lema é mais útil para mostrar que uma linguagem não pertence à família das linguagens livre do contexto. Seu uso é análogo aquele visto para linguagens regulares.

**Exemplo 7.1.3** Mostre que a linguagem  $\mathcal{L} = \{a^n b^n c^n / n \geq 0\}$  não é livre do contexto.

Suponha que  $\mathcal{L}$  é livre do contexto. Então, pelo lema do bombeamento, existe  $m$  satisfazendo as condições do teorema 7.1.1. A cadeia  $a^m b^m c^m$  está em  $\mathcal{L}$ . Se escolhermos  $vxy$  como sendo uma cadeia só de  $a$ 's, então, pelo lema do bombeamento, a cadeia  $uv^2 xy^2 z$ , por exemplo, deveria estar em  $\mathcal{L}$ , mas como  $|vy| \leq 1$ , ela claramente não é da forma  $a^n b^n c^n$ , pois tem mais  $a$ 's que  $b$ 's (e  $c$ 's). Analogamente, se escolhermos as subcadeias  $v$  composta só de  $a$ 's, digamos  $j$   $a$ 's, e  $y$  composta só de  $b$ 's, digamos  $k$   $b$ 's, então a cadeia bombeada  $uv^2 xy^2 z$ , terá a forma  $a^{m+j} b^{m+k} c^m$ , e portanto, novamente geramos uma cadeia que não está em  $\mathcal{L}$ . De fato, a única maneira da escolha das subcadeias  $u, v, x, y$  e  $z$  impedir esse fato é tomar  $vxy$  tal que  $vy$  tenha o mesmo número de  $a$ 's,  $b$ 's e  $c$ 's. Mas isso não é possível pela restrição (7.2). Portanto,  $\mathcal{L}$  não é livre do contexto.

Uma forma direta de se provar isso é dado no seguinte corolário.

**Corolário 7.1.4** Seja  $\mathcal{L}$  uma linguagem infinita.  $\mathcal{L}$  não é livre do contexto se para cada inteiro positivo  $m$  existe  $w \in \mathcal{L}$ , tal que  $|w| \geq m$  e para cada decomposição  $uvxyz$  de  $w$ , se

$$|vxy| \leq m \quad (7.4)$$

e

$$|vy| \geq 1 \quad (7.5)$$

então  $uv^i xy^i z \notin \mathcal{L}$ , para algum  $i = 0, 1, 2, \dots$

DEMONSTRAÇÃO: Direto do lemma do bombeamento. ■

**Exemplo 7.1.5** Considere a linguagem

$$\mathcal{L} = \{ww \mid w \in \{a, b\}^*\}.$$

Apesar desta linguagem parecer muito similar à linguagem livre do contexto do exemplo 5.1.2, ela não é livre do contexto.

Seja  $m \in \mathbb{N}$  qualquer e considere a cadeia  $w = a^m b^m a^m b^m$ . Nos teríamos que analisar todas as possíveis decomposições  $uvxyz$  de  $w$ , e se a decomposição satisfazer (7.4) e (7.5) então necessariamente algum bombeamento de  $v$  e  $y$  deverá gerar uma cadeia não pertencente à linguagem. Mesmo considerando só as decomposições que satisfazem (7.4) e (7.5) ainda são muitas (além do mais que nós não conhecemos o valor real de  $m$ ). Mas como a restrição (7.4) força que  $|vxy| \leq m$ , então ela só pode estar em:

$$\begin{array}{c} \underbrace{a \cdots a}_{u} \underbrace{a \cdots a}_{vxy} \underbrace{a \cdots ab^m}_{z} a^m b^m \text{ ou } \underbrace{a^m b}_{u} \underbrace{\cdots b}_{vxy} \underbrace{b \cdots b}_{z} \underbrace{b \cdots ba^m}_{b^m} \text{ ou} \\ \underbrace{a^m b^m a}_{u} \underbrace{\cdots a}_{vxy} \underbrace{a \cdots a}_{z} \underbrace{a \cdots ab^m}_{b^m} \text{ ou } \underbrace{a^m b^m a^m b}_{u} \underbrace{\cdots b}_{vxy} \underbrace{b \cdots b}_{z} \underbrace{b \cdots b}_{b^m} \end{array} \quad (7.6)$$

ou ainda

$$\begin{array}{c} \underbrace{a \cdots a}_{u} \underbrace{a \cdots ab}_{vxy} \underbrace{\cdots b}_{z} \underbrace{b \cdots ba^m}_{b^m} \text{ ou } \underbrace{a^m b}_{u} \underbrace{\cdots b}_{vxy} \underbrace{b \cdots ba}_{z} \underbrace{\cdots a}_{a} \underbrace{a \cdots ab^m}_{b^m} \text{ ou} \\ \underbrace{a^m b^m a}_{u} \underbrace{\cdots a}_{vxy} \underbrace{a \cdots a}_{z} \underbrace{a \cdots ab}_{b^m} \underbrace{\cdots b}_{b} \end{array} \quad (7.7)$$

Note que todas as decomposições em (7.6) e em (7.7) são análogas, e portanto é suficiente considerar um caso de cada.

Assim, no caso de (7.6) consideraremos somente a primeira decomposição. Assim,  $u = a^{k_1}$ ,  $v = a^{k_2}$ ,  $x = a^{k_3}$ ,  $y = a^{k_4}$  e  $z = a^{k_5} b^m a^m b^m$  para algum  $k_1, k_2, k_3, k_4, k_5 \in \mathbb{N}$  tal que  $k_1 + k_2 + k_3 + k_4 + k_5 = m$  e  $k_2 + k_4 \geq 1$ . Assim,

$$uv^2 xy^2 z = a^{k_1} a^{2k_2} a^{k_3} a^{2k_4} a^{k_5} b^m a^m b^m = a^{k_1} a^{k_2} a^{k_2} a^{k_3} a^{k_4} a^{k_4} a^{k_5} b^m a^m b^m = a^{m+k_2+k_4} b^m a^m b^m.$$

Como  $k_2 + k_4 \geq 1$ , então  $uv^2 xy^2 z \notin \mathcal{L}$ .

No caso de (7.7) consideraremos somente a primeira decomposição. Mas, aqui novamente teremos várias possibilidades, por exemplo  $v$  e  $x$  podem ser só composto de  $a$ 's e  $y$  de  $b$ 's ou  $y$  de  $a$ 's e de  $b$ 's, ou ainda,  $v$  pode ser composto de  $a$ 's e  $x$  de  $a$ 's e de  $b$ 's e  $y$  de  $b$ 's, etc.

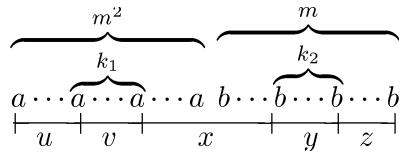
## 7.2. Propriedades de Fecho para Linguagens Livres do Contexto

---

*Mas independente de qual for decomposição ao bombeiar o v e o y (que satisfazem  $|vy| \geq 1$ ) será alterada a quantidade de a's e/ou de b's da primeira metade da cadeia w pelo que a cadeia resultante já não terá mais a forma  $a^m b^m a^m b^m$  e portanto não pertencerá à linguagem. Logo,  $\mathcal{L}$  não é livre do contexto.*

**Exemplo 7.1.6** Mostre que a linguagem  $\mathcal{L} = \{a^j b^k / j = k^2\}$  não é livre do contexto.

Dado um m qualquer, tomamos nossa cadeia como sendo  $a^{m^2} b^m$ . Claramente, temos várias escolhas para u, v, x, y, z. A única que requer mais cuidado é a seguinte:



Bombeando as subcadeias v e y, i vezes, geraremos a nova cadeia com  $m^2 + (i - 1)k_1$  a's e  $m + (i - 1)k_2$  b's. Se  $k_1 = 0$ , então, pela restrição (7.5),  $k_2 > 1$ . Para o caso  $i = 0$ , teremos  $m^2$  a's e  $m - k_2$  b's o qual obviamente não está em  $\mathcal{L}$ . Usando um raciocínio análogo, obteremos que a escolha  $k_2 = 0$  e  $k_1 > 1$ , também falha. Consideraremos, portanto, o caso  $k_1 \neq 0$  e  $k_2 \neq 0$  e tomemos  $i = 0$ . Como

$$\begin{aligned} (m - k_2)^2 &\leq (m - 1)^2 \\ &= m^2 - 2m + 1 \\ &< m^2 - k_1, \end{aligned}$$

o resultado não está em  $\mathcal{L}$ . Portanto,  $\mathcal{L}$  não é livre do contexto.

## 7.2 Propriedades de Fecho para Linguagens Livres do Contexto

No capítulo 4, analisamos algumas propriedades de fecho para certas operações e alguns algoritmos de decisão sobre as propriedades da família de linguagens regulares. Ambas questões, eram facilmente provadas. No entanto, as mesmas questões para linguagens livres do contexto são ligeiramente mais difíceis de se responder. De fato, algumas propriedades de fecho que são satisfeitas por linguagens regulares não o são por linguagens livres do contexto. Nesta seção, somente analisaremos as propriedades de fecho das operações mais tradicionais entre linguagens.

**Teorema 7.2.1** A família de linguagens livres do contexto é fechada sobre união, concatenação e fecho estrela (\*).

**DEMONSTRAÇÃO:** Sejam  $L_1$  e  $L_2$  duas linguagens livres do contexto qualquer e  $G_1 = \langle V_1, T_1, S_1, P_1 \rangle$  e  $G_2 = \langle V_2, T_2, S_2, P_2 \rangle$  duas gramáticas livres do contexto que geram elas. Por simplicidade sempre podemos considerar que  $V_1 \cap V_2 = \emptyset$ .

As gramáticas livres do contexto

1.  $G_{\cup} = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\} \rangle$

## Capítulo 7. Propriedades das Linguagens Livres do Contexto

2.  $G_\circ = \langle V_1 \cup V_2, S_1, (P_1 - P) \cup P_2 \cup \{A \rightarrow wS_2 / A \rightarrow w \in P\} \rangle$  onde  $P = \{A \rightarrow w \in P_1 / w \in T_1^*\}$  e
3.  $G_* = \langle V_1, T_1, S_1, P_1 \cup \{A \rightarrow wS_1 / A \rightarrow w \in P \text{ e } w \in T_1^*\} \cup \{S_1 \rightarrow \lambda\} \rangle$ ,

geram as linguagens  $L_1 \cup L_2$ ,  $L_2 \circ L_1$  e  $L_1^*$ , respectivamente. ■

**Teorema 7.2.2** *A família de linguagens livres do contexto não é fechada sobre intersecção e complementação.*

DEMONSTRAÇÃO: Considere as linguagens

$$\mathcal{L}_1 = \{a^n b^n c^m / n \geq 0, m \geq 0\}$$

e

$$\mathcal{L}_2 = \{a^n b^m c^m / n \geq 0, m \geq 0\}.$$

Claramente  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são livres do contexto. Mas, como vimos no exemplo 7.1.3, a linguagem

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \{a^n b^n c^n / n \geq 0\},$$

não é livre do contexto. Logo, as linguagens livres do contexto não são fechadas sobre a intersecção.

Por outro lado, seja a identidade de conjuntos

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}. \quad (7.8)$$

Se as linguagens livres do contexto fossem fechadas sobre a complementação, então, como pelo teorema anterior elas são fechadas sobre a união, a linguagem em (7.8) também seria livre do contexto, o qual contradiziria o resultado anterior. Conseqüentemente, a família das linguagens livres do contexto não é fechada sobre a complementação. ■

### 7.3 Propriedades Decidíveis de Linguagens Livres do Contexto

No capítulo 4 já foi visto a existência de um algoritmo de pertinência para linguagens livres do contexto. Isto é uma característica essencial de qualquer família de linguagens útil na prática. Outras propriedades simples de linguagens livres do contexto podem ser determinadas. Mas, algumas questões intuitivamente simples de se responder para linguagens regulares via um algoritmo, não podem ser respondidas algorítmicamente para livres do contexto. Para os propósitos desta discussão, assumiremos que a linguagem é descrita pela sua gramática.

**Teorema 7.3.1** *Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Então existe um algoritmo para decidir quando  $L(G)$ , for vazio ou não.*

### 7.3. Propriedades Decidíveis de Linguagens Livres do Contexto

---

**DEMONSTRAÇÃO:** Por simplicidade assuma que  $\lambda \notin L(G)$ . Use o algoritmo para remover símbolos e produções inúteis. Se  $S$  for inútil, então  $L(G)$  é vazio, senão  $L(G)$  contém ao menos um elemento. ■

**Teorema 7.3.2** *Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto. Então existe um algoritmo para decidir quando  $L(G)$ , for infinito ou não.*

**DEMONSTRAÇÃO:** Assuma que  $G$  não contém nenhuma  $\lambda$ -produção, nenhuma produção unitária e nenhum símbolo inútil. Se existe na gramática uma variável se repetindo,  $A \in V$ , isto é, uma variável tal que

$$A \xrightarrow{*} xAy,$$

então, como  $A$  não é inútil, deve existir  $w \in T^*$  tal que

$$S \xrightarrow{*} uAv \xrightarrow{*} uxAyv \xrightarrow{*} w,$$

Logo,  $u \xrightarrow{*} x_1$ ,  $x \xrightarrow{*} x_2$ ,  $A \xrightarrow{*} x_3$ ,  $y \xrightarrow{*} x_4$  e  $v \xrightarrow{*} x_5$  com  $x_1, x_2, x_3, x_4, x_5 \in T^*$  são tais que  $x_1x_2x_3x_4x_5 = w$ . Portanto, para todo  $n = 1, 2, \dots$

$$S \xrightarrow{*} uAv \xrightarrow{*} ux^nAy^nv \xrightarrow{*} ux^nzy^nv \xrightarrow{*} x_1x_2^n x_3 x_4^n x_5$$

e portanto  $L(G)$  seria infinita.

Se nenhuma variável pode ser repetida, então o comprimento de qualquer derivação não excede  $|V|$ . Em cujo caso  $L(G)$  é finita.

Assim, para obter um algoritmo para determinar quando  $L(G)$  é infinita, somente precisamos determinar quando a gramática tem uma variável repetindo. Isto pode ser conseguido, simplesmente, desenhandando o grafo de dependência das variáveis. Se existe um ciclo, então a gramática tem uma variável se repetindo, pois não temos variáveis inúteis, e portanto sempre podemos chegar a ela desde  $S$ . ■

## 7.4 Exercícios

1. Mostre usando o lema do bombeamento que as seguintes linguagens são livres do contexto
  - (a)  $\mathcal{L} = \{a^n b^n / n \text{ é um número ímpar}\}$
  - (b)  $\mathcal{L} = \{a^n b^{2n} / n \text{ é múltiplo de } 5\}$
  - (c)  $\mathcal{L} = \{a^n b^p / n \geq p\}$
  - (d)  $\mathcal{L} = \{a^n b^p / n \geq 1 \text{ e } n \leq p \leq 3n\}$
  - (e)  $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \text{ é par}\}$
  - (f)  $\mathcal{L} = \{w \in \{a, b\}^* / w = w^R\}$
  - (g)  $\mathcal{L} = \{w \in \{a, b\}^* / w \neq w^R\}$
  
2. Mostre que as seguintes linguagens, sobre o alfabeto  $\{a, b\}$ , não são livres do contexto
  - (a)  $\mathcal{L} = \{a^m b^n / n = m^2\}$
  - (b)  $\mathcal{L} = \{a^m b^n / m \leq n^2\}$
  - (c)  $\mathcal{L} = \{a^m b^n / m \geq (n - 1)^3\}$
  - (d)  $\mathcal{L} = \{uv \in \{a, b\}^* / \mathcal{N}_a(u) + \mathcal{N}_b(u) = \max\{\mathcal{N}_a(v), \mathcal{N}_b(v)\}\}$
  
3. Mostre que as seguintes linguagens, sobre o alfabeto  $\{a, b, c\}$ , não são livres do contexto
  - (a)  $\mathcal{L} = \{a^k b^m c^n / n = km\}$
  - (b)  $\mathcal{L} = \{a^k b^m c^n / k < m \text{ e } k \leq m \leq n\}$
  - (c)  $\mathcal{L} = \{a^k b^m c^n / m > k \text{ e } n > k\}$
  - (d)  $\mathcal{L} = \{w / \mathcal{N}_a(w) < \mathcal{N}_b(w) < \mathcal{N}_c(w)\}$
  
4. Determine quais das seguintes linguagens, sobre o alfabeto  $\{a, b\}$ , são livres do contexto e quais não
  - (a)  $\mathcal{L} = \{a^n w w^R a^n / n \geq 0, w \in \{a, b\}^*\}$
  - (b)  $\mathcal{L} = \{a^m b^n a^m b^n / m \geq 0, n \geq 0\}$
  - (c)  $\mathcal{L} = \{a^m b^n a^n b^m / m \geq 0, n \geq 0\}$
  - (d)  $\mathcal{L} = \{a^m b^n a^j b^k / m + n \leq j + k\}$
  - (e)  $\mathcal{L} = \{a^m b^n a^j b^k / m \leq j, n \leq k\}$
  
5. Mostre que a classe das linguagens livres do contexto é fechada sobre os seguintes operadores:
  - (a) Homomorfismos,
  - (b) Diferença de conjuntos,
  - (c) O operador reverso

## 7.4. Exercícios

---

(d) Seja  $\mathcal{L}$  uma linguagem qualquer. Define  $\widehat{\mathcal{L}}_a$  como sendo

$$\widehat{\mathcal{L}}_a = \{awa / w \in \mathcal{L}\}$$

(e) Seja  $\mathcal{L}$  um linguagem sobre o alfabeto  $\{0, 1\}$  e  $\neg : \Sigma^* \longrightarrow \Sigma^*$  definido por  $\neg(\lambda) = \lambda$ ,  $\neg(w0) = 1\neg(w)$  e  $\neg(w1) = 0\neg(w)$ . Define  $\neg\mathcal{L}$  como sendo

$$\neg\mathcal{L} = \{\neg(w) / w \in \mathcal{L}\}$$

(f) Seja  $\mathcal{L}$  uma linguagem qualquer. Define  $\widetilde{\mathcal{L}}_a$  como sendo

$$\widetilde{\mathcal{L}}_a = \{waw / w, v \in \mathcal{L}\}$$

(g) Seja  $\mathcal{L}$  uma linguagem qualquer sobre um alfabeto  $\Sigma$  e  $\rho : \Sigma^* \longrightarrow \Sigma^*$  definido por  $\rho(\lambda) = \lambda$  e  $\rho(wa) = \rho(w)aa$  para todo  $w \in \Sigma^*$  e  $a \in \Sigma$ . Define  $\mathcal{L}^\rho$  como sendo

$$\mathcal{L}^\rho = \{\rho(w) / w \in \mathcal{L}\}$$

(h) Seja  $\mathcal{L}$  uma linguagem qualquer sobre um alfabeto  $\Sigma$  e  $\varrho : \Sigma^* \longrightarrow \Sigma^*$  definido por  $\varrho(\lambda) = \lambda$  e  $\varrho(wa) = a\varrho(w)aa$  para todo  $w \in \Sigma^*$  e  $a \in \Sigma$ . Define  $\mathcal{L}^\varrho$  como sendo

$$\mathcal{L}^\varrho = \{\varrho(w) / w \in \mathcal{L}\}$$

6. Seja  $L(G)$  a linguagem gerada por uma gramática livre do contexto  $G$ . Dê um algoritmo para determinar quando  $\lambda \in L(G)$  ou não.
7. Mostre que existe um algoritmo que determina quando a linguagem gerada por alguma gramática livre do contexto contém qualquer palavra de comprimento menor que algum número  $n$ .

## Capítulo 8

# Máquinas de Turing

Dentre as idéias discutidas anteriormente se destacam as linguagens regulares e as livres do contexto, assim como sua associação com autômatos finitos e autômatos com pilhas, respectivamente. Observamos que a família das linguagens regulares forma um subconjunto próprio das linguagens livres do contexto e por isso os autômatos com pilha são, em algum sentido, mais poderosos do que os autômatos finitos. Vimos, também que as linguagens livres do contexto, embora fundamentais para estudar linguagens de programação são de escopo limitado. Isto porque certas linguagens simples, como  $\mathcal{L} = \{a^n b^n c^n / n \geq 0\}$  e  $\{ww / w \in \{a, b\}^*\}$ , não são livres do contexto. Daí procurarmos definir novas classes de linguagens que considerem esses casos. Para isso vamos retornar ao conceito de um autômato. Se compararmos os autômatos finitos com os autômatos com pilha observaremos que a natureza da memória temporária faz a diferença entre eles. Um autômato com pilha sem memória temporária é um autômato finito. É razoável esperar que existam famílias de linguagens mais poderosas se dermos mais flexibilidade de memória ao autômato. Por exemplo, o que aconteceria se usássemos duas pilhas, três pilhas, uma fila ou algum outro dispositivo de memória?. A cada novo dispositivo de memória corresponde definir uma nova espécie de autômato e através dele eventualmente uma nova família de linguagens. Esta abordagem suscita uma grande quantidade de questões, muitas das quais não são muito interessantes. É mais útil colocar questões mais ambiciosas e considerar até onde iria o conceito de autômato. O que podemos dizer sobre o autômato mais poderoso e os limites da computação? Isto leva ao conceito fundamental de **máquina de Turing** e por seu turno, a uma definição precisa da idéia de uma computação mecânica e algorítmica.

Começaremos com a definição formal de uma máquina de Turing e então procuraremos a intuição fazendo alguns programas simples. Após, argumentaremos que, embora rudimentar, o conceito é bastante abrangente para cobrir processos muito mais complexos. A discussão culmina na *tese de Turing*, que assegura que qualquer processo computacional, como aqueles efetuados pelos computadores atuais, pode ser efetuado por uma máquina de Turing.

### 8.1 A Máquina de Turing Padrão

Embora seja possível imaginar uma grande variedade de autômatos com dispositivos de armazenamento complexos e sofisticados, a memória de uma máquina de Turing é bastante simples. Ela pode ser visualizada como um vetor unidimensional de células, cada uma das quais

## 8.1. A Máquina de Turing Padrão

---

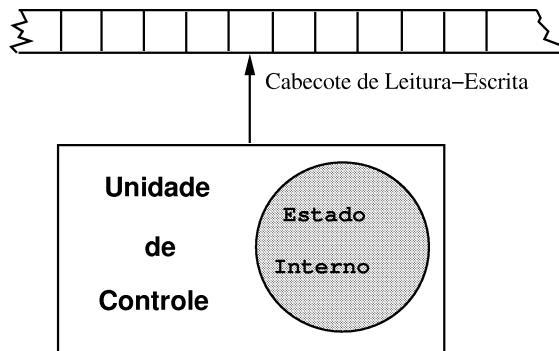


Figura 8.1: Representação esquemática de uma máquina de Turing.

pode manter um único símbolo. Este vetor se estende indefinidamente em ambas as direções e, portanto, é capaz de manter uma quantidade ilimitada de informações. A informação pode ser lida e trocada em qualquer ordem. Chamaremos tal dispositivo de armazenamento uma *fita*, porque ela é análoga às fitas magnéticas, tradicionalmente, usadas nos computadores atuais.

### 8.1.1 Definição de Máquina de Turing

Uma máquina de Turing é um autômato cuja memória temporária é uma fita a qual é dividida em células, cada uma das quais é capaz de manter um símbolo. Associado à fita existe um *cabeçote de leitura-escrita* que pode se mover uma célula para a direita ou para a esquerda da fita, podendo ler e escrever um único símbolo em cada movimento. Uma representação esquemática de uma máquina de Turing é mostrada na figura 8.1. Esta representação esquemática de uma máquina de Turing foge da representação esquemática geral de um autômato mostrada na figura 1.3, porém podemos ver uma máquina de Turing como sendo um autômato onde a fita de armazenamento se comporta como acima, e onde inicialmente o autômato transfere o conteúdo da fita entrada à fita de armazenamento. Mas usaremos a primeira noção de máquinas de Turing por ser mais simples e conhecida.

**Definição 8.1.1** Uma máquina de Turing  $M$  é uma séte-tupla  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$ , onde

- $Q$  é um conjunto finito de *estados internos*,
- $\Sigma$  é um conjunto finito de símbolos, chamado *alfabeto de entrada*,
- $\Gamma$  é um conjunto finito de símbolos, chamado *alfabeto da fita*,
- $\delta$  é uma função (parcial) de *transição*,
- $\square \in \Gamma$  é um símbolo especial, chamado *branco*,
- $q_0 \in Q$  é o estado inicial e
- $F \subseteq Q$  é o conjunto de *estados finais*.

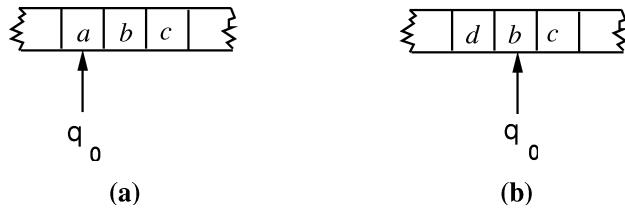


Figura 8.2: A situação (a) antes do movimento e (b) após.

Na definição, assumiremos que  $\Sigma \subseteq \Gamma - \{\square\}$ , isto é, que o alfabeto de entrada, não inclui o branco. O branco é tirado dos símbolos de entrada por razões que ficarão claras a seguir. A função de transição  $\delta$  é definida como:

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{E, D\}.$$

Em geral,  $\delta$  é uma função parcial sobre  $Q \times \Gamma$ , sua interpretação nos dá o princípio pelo qual uma máquina de Turing opera. Os argumentos de  $\delta$  são o estado corrente da unidade de controle e o símbolo corrente da fita sendo lido. O símbolo de movimento indica se o cabeçote de escrita-leitura move-se para a esquerda (E) ou para a direita (D) uma célula após o novo símbolo ter sido escrito na fita.

**Exemplo 8.1.2** A figura 8.2 mostra a situação antes e após o movimento causado pela transição

$$\delta(q_0, a) = (q_1, d, D)$$

Podemos pensar uma máquina de Turing como um computador muito simples. ele tem uma unidade de processamento, que tem uma memória finita, e na sua fita, ele tem uma memória secundária com capacidade ilimitada. As instruções que tal computador pode efetuar são muito limitadas: ele pode perceber um símbolo na sua fita e usá-lo como resultado para decidir o que fazer no próximo passo. As únicas ações que a máquina pode efetuar são: reescrever o símbolo corrente, trocar o estado do controle e mover o cabeçote uma célula para a direita ou esquerda. Este pequeno conjunto de instruções pode parecer inadequado para efetuar tarefas complicadas, porém isso não é verdade. As máquinas de Turing são, em princípio, muito poderosas. A função de transição,  $\delta$ , define como esse computador pode atuar, por isso chamamos essa função de “programa” da máquina.

Como sempre, o autômato começa num dado estado inicial com alguma informação na fita (cada célula inicialmente contém ou um símbolo de entrada em  $\Sigma$  ou o símbolo especial  $\square$ , mas a quantidade de símbolos não brancos é sempre finita). Então, ele executa uma seqüência de etapas controladas pela função de transição  $\delta$ . Durante esse processo, o conteúdo de qualquer célula na fita pode ser examinado e alterado em qualquer momento. Mais cedo ou mais tarde, o processo todo pode parar. Uma máquina de Turing **pára** de computar se atinge uma configuração para a qual  $\delta$  não está definida. Isto é possível porque  $\delta$  é uma função parcial. De fato, assumiremos que só existe um único estado final para o qual não são definidas transições. Portanto, a máquina de Turing parará sempre que ela entrar nesse estado final.

### 8.1. A Máquina de Turing Padrão

---

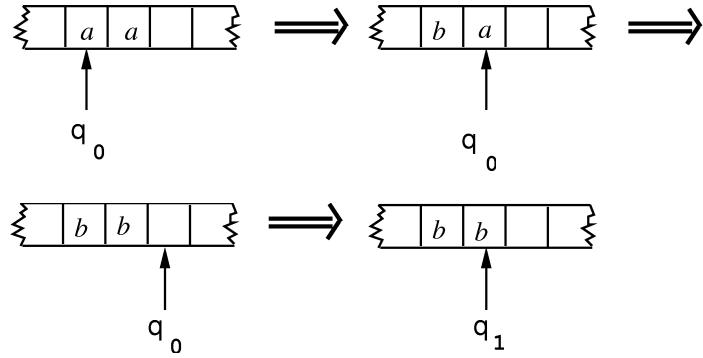


Figura 8.3: Parte da execução de uma máquina de Turing.

**Exemplo 8.1.3** Considere a máquina de Turing definida por

- $Q = \{q_0, q_1\}$ ,
- $\Sigma = \{a, b\}$ ,
- $\Gamma = \{a, b, \square\}$ ,
- $F = \{q_1\}$  e
- $\delta(q_0, a) = (q_0, b, D)$ ,  $\delta(q_0, b) = (q_0, b, D)$ ,  $\delta(q_0, \square) = (q_1, \square, E)$ .

Se esta máquina de Turing começa com o símbolo “a”, sob o cabeçote de leitura-escrita, a regra de transição aplicável é  $\delta(q_0, a) = (q_0, b, D)$ . Portanto, o cabeçote de leitura-escrita mudará o “a” pelo “b”, e então se moverá para a direita na fita. A máquina permanecerá no estado  $q_0$ . Qualquer “a” subsequente também será trocado, mas os b’s não serão modificados. Quando a máquina encontrar o primeiro branco, ela se moverá uma célula para a esquerda, e então parará no estado final  $q_1$ .

A figura 8.3 mostra vários estágios do processo para uma configuração inicial simples.

**Exemplo 8.1.4** Tome  $Q$ ,  $\Sigma$ ,  $\Gamma$  e  $F$ , como no exemplo anterior. Defina  $\delta$  por

$$\begin{aligned}\delta(q_0, a) &= (q_1, a, D), \quad \delta(q_0, b) = (q_1, b, D), \quad \delta(q_0, \square) = (q_1, \square, D), \\ \delta(q_1, a) &= (q_0, a, E), \quad \delta(q_1, b) = (q_0, b, E), \quad \delta(q_1, \square) = (q_0, \square, E).\end{aligned}$$

Para ver o que acontece aqui, podemos considerar um caso típico. Suponha que a fita inicialmente contém  $ab\dots$ , com o cabeçote de leitura-escrita apontando para “a”. A máquina lê o “a”, mas não o altera. Seu próximo estado é  $q_1$  e o cabeçote se move para a direita, de modo que ele, agora, aponta para “b”. Este símbolo é lido e deixado também inalterado. A máquina volta para o estado  $q_0$  e o cabeçote se move para a esquerda. Estamos, agora, exatamente na situação (configuração) original e a seqüência de movimentos começa novamente. É claro que a máquina, qualquer que seja a informação inicial na fita, executará para sempre, com o cabeçote se movendo para a direita e para a esquerda, alternadamente, mas não realizando qualquer modificação na fita. Este é um exemplo de uma máquina que não pára, em analogia com a terminologia, em programação, dizemos que a máquina de Turing está num “laço infinito”.

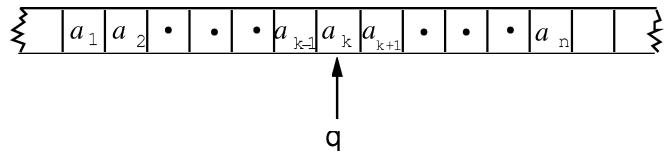


Figura 8.4: Configuração arbitrária de uma máquina de Turing

Como podemos dar definições diferentes de uma máquina de Turing é bom resumir as principais características do nosso modelo, o qual será chamado *máquina de Turing padrão*:

1. A máquina de Turing tem uma fita que é ilimitada em ambas as direções, permitindo qualquer número de movimentos para a direita ou para a esquerda.
2. A máquina é determinística no sentido de que  $\delta$  define, no máximo, um movimento para cada configuração.
3. Não existe fita de entrada especial. Assumimos que no início a fita tem um conteúdo especial. Alguns destes podem ser considerados entradas. Analogamente, não existe nenhum dispositivo de saída. Sempre que a máquina pára, algum ou todo o conteúdo da fita pode ser visto como saída.

Para exibir as configurações de uma máquina de Turing usamos a idéia de uma **descrição instantânea**. Qualquer configuração de uma máquina de Turing é completamente determinada pelo estado corrente da unidade de controle, o conteúdo da fita, e a posição do cabeçote de leitura-escrita. Usaremos a notação

$$a_1 a_2 \cdots a_{k-1} q a_k a_{k+1} \cdots a_n$$

para a descrição instantânea de uma máquina de Turing no estado  $q$ , com o conteúdo da fita como o da figura 8.4

Os símbolos  $a_1, \dots, a_n$  constituem o conteúdo da fita, enquanto  $q$  define o estado da unidade de controle. Esta convenção é escolhida de tal modo que a posição do cabeçote está sobre a célula contendo o símbolo imediatamente seguindo  $q$ .

A descrição instantânea nos dá somente uma quantidade finita de informações para a direita e esquerda do cabeçote. A parte não especificada da fita é assumida conter somente brancos. Normalmente tais brancos são irrelevantes e, não são mostrados explicitamente, na descrição instantânea. Se a posição do branco é relevante para a discussão ele aparecerá na descrição instantânea. Por exemplo, a descrição instantânea  $q \square w$  indica que o cabeçote está na célula imediatamente à esquerda do primeiro símbolo de  $w$  e que esta célula contém um branco.

**Exemplo 8.1.5** O desenho da figura 8.3 corresponde à seqüência de descrições instantâneas  $q_0 a a$ ,  $b q_0 a$ ,  $b b q_0 \square$  e  $b q_1 b$ .

## 8.1. A Máquina de Turing Padrão

---

Um movimento de uma configuração a outra será denotado por  $\vdash$ . Portanto, se

$$\delta(q_1, c) = (q_2, e, D),$$

um movimento possível é

$$abq_1cd \vdash abeq_2d.$$

Os símbolos  $\vdash^*$  e  $\vdash^+$  tem os significados usuais de um número arbitrário de movimentos. Sub-escritos  $\vdash_M$  são usados em argumentos para distinguir entre várias máquinas.

**Exemplo 8.1.6** A ação da máquina de Turing na figura 8.3 pode ser representada por

$$q_0aa \vdash bq_0a \vdash bbq_0\Box \vdash bq_1b$$

ou

$$q_0bb \vdash^* bq_1b.$$

A seguir resumiremos de um modo formal todas as observações feitas anteriormente.

**Definição 8.1.7** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \Box, F \rangle$  uma máquina de Turing. Uma **descrição instantânea** de  $M$  é uma cadeia  $a_1 \dots a_{k-1} q a_k \dots a_n$ , onde  $a_i \in \Gamma$  para cada  $i = 1, \dots, n$  e  $q \in Q$ . Um movimento

$$a_1 \dots a_{k-1} q_1 a_k a_{k+1} \dots a_n \vdash a_1 \dots a_{k-1} b q_2 a_{k+1} \dots a_n$$

é possível se e somente se

$$\delta(q_1, a_k) = (q_2, b, D).$$

Um movimento

$$a_1 \dots a_{k-1} q_1 a_k a_{k+1} \dots a_n \vdash a_1 \dots a_{k-2} q_2 a_{k-1} b a_{k+1} \dots a_n$$

é possível se e somente se

$$\delta(q_1, a_k) = (q_2, b, E).$$

Uma configuração  $y_1 q_p a y_2$  é uma **configuração de parada**, denotado por,  $y_1 q_p a y_2 \vdash$ , se  $\delta(q_p, a)$  não é definido. Assim, uma máquina de Turing  $M$  pára para a configuração inicial  $x_1 q_i x_2$  se

$$x_1 q_i x_2 \vdash^* y_1 q_j a y_2$$

para algum  $q_j \in Q$  e  $a \in \Gamma$ , para o qual  $\delta(q_j, a)$  não é definido. A seqüência de configurações levando a uma configuração de parada será chamada de **computação**.

O exemplo 8.1.4 mostra a possibilidade de que uma máquina de Turing pode nunca parar, permanecendo num laço sem fim do qual ela não pode escapar. Esta situação desempenha um papel fundamental na descrição de uma máquina de Turing, representemo-la por

$$x_1 q x_2 \vdash^* \infty$$

indicando que, iniciando da configuração inicial  $x_1 q x_2$  a máquina nunca pára.

### 8.1.2 Máquinas de Turing como Reconhecedoras de Linguagens

As máquinas de Turing podem ser vistas como reconhecedoras no seguinte sentido. Uma cadeia  $w$  é escrita sobre a fita, com brancos preenchendo as porções não usadas. A máquina começa no estado inicial  $q_0$ , com a posição do cabeçote no símbolo mais à esquerda de  $w$ . Se após uma seqüência de movimentos, a máquina de Turing entra num estado final e pára, então  $w$  é considerada reconhecida pela máquina de Turing. Lembre, que por convenção, quando uma execução de uma máquina de Turing atinge um estado final, ela pára.

**Definição 8.1.8** *Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  uma máquina de Turing. Então a linguagem reconhecida por  $M$  é*

$$L(M) = \{w \in \Sigma^+ / q_0 w \vdash^* x_1 q_f x_2 \text{ para algum } q_f \in F, x_1, x_2 \in \Gamma^*\}$$

Esta definição indica que a entrada  $w$  é escrita na fita com brancos em ambos os lados. A razão para excluir brancos da entrada se torna claro agora: Ela assegura-nos que todas as entradas estão restritas a uma região bem definida (finita) da fita, separadas por brancos à esquerda e à direita. Sem essa convenção, a máquina não poderia limitar a região na qual ela deveria olhar para a entrada. Não importa quantos brancos ela veja, ela nunca poderia estar segura de que não houvesse alguma entrada não branca em algum outro lugar da fita.

A definição 8.1.8 nos diz o que deve acontecer quando um  $w \in L(M)$ . Ela não diz nada sobre a saída para qualquer outra entrada. Quando  $w \notin L(M)$ , uma de duas coisas pode acontecer: a máquina pode parar num estado não final ou ela pode entrar num laço infinito e nunca parar. Qualquer cadeia para qual  $M$  não pára, por definição não está em  $L(M)$ .

**Exemplo 8.1.9** *Para  $\Sigma = \{0, 1\}$  projetar uma máquina de Turing que reconhece a linguagem denotada pela expressão regular  $0^*$ .*

*Este é um exercício fácil de programação em máquinas de Turing. Começando no lado esquerdo da entrada, lemos cada símbolo e checamos se ele é um 0. Se for o caso, continuamos movendo para a direita. Se atingirmos um branco sem encontrar nada além de 0, terminamos e reconhecemos a cadeia. Se a cadeia contém um 1 em algum lugar, a cadeia não está em  $L(0^*)$ , e paramos num estado não final. A máquina deve ter dois estados  $Q = \{q_0, q_1\}$  e um estado final  $F = \{q_1\}$ . A função de transição pode ser especificada por*

$$\delta(q_0, 0) = (q_0, 0, D), \delta(q_0, \square) = (q_1, \square, E).$$

## 8.1. A Máquina de Turing Padrão

---

Toda vez que 0 aparece sob o cabeçote este se moverá para a direita. Se em qualquer momento 1 é lido, a máquina parará no estado não final  $q_0$ , pois  $\delta(q_0, 1)$  não está definida.

O reconhecimento de uma linguagem mais complicada, em geral, é difícil. Como máquinas de Turing têm um conjunto de instruções muito primitivas, as computações que podemos programar numa linguagem de programação de nível mais alto são, em geral, difíceis de serem realizadas numa máquina de Turing. Mesmo assim, é possível e o conceito é fácil de entender, como o próximo exemplo mostra.

**Exemplo 8.1.10** Para  $\Sigma = \{a, b\}$ , projetar uma máquina de Turing que reconhece a linguagem  $\mathcal{L} = \{a^n b^n / n \geq 1\}$ .

Intuitivamente, resolvemos o problema do seguinte modo. Começamos no “a” mais à esquerda e o apagamos da fita, então movemos o cabeçote para a direita até encontrar o “b” mais à esquerda, que por seu turno também é apagado. Após isso, nos movemos para a esquerda novamente até o “a” mais à esquerda e o apagamos, então nos movemos até o “b” mais à direita e também o apagamos, e assim por diante. Indo para trás e para frente desta maneira, casamos cada “a” com o “b” correspondente. Se após algum tempo nenhum “a” ou “b” restar, então a cadeia deve estar em  $\mathcal{L}$ . Note que a máquina não somente não deve aceitar cadeias do tipo  $a^m b^n$  com  $n \neq m$  mas também cadeias onde um b preceda um a.

Desse modo a máquina será  $M = \langle Q, \Sigma, \Gamma, q_0, \square, F \rangle$ , onde

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{a, b, \square\}$
- $F = \{q_4\}$

As transições podem ser separadas em várias partes. O conjunto

$$\delta(q_0, a) = (q_1, \square, D), \quad \delta(q_1, a) = (q_1, a, D),$$

$$\delta(q_1, b) = (q_1, b, D), \quad \delta(q_1, \square) = (q_2, \square, E)$$

apaga o “a” mais à esquerda e então move o cabeçote para a direita até encontrar o “b” mais à direita (na verdade encontra o primeiro  $\square$  à direita e então retorna uma célula, a qual deveria ser um b)

O próximo conjunto de transições apaga o b mais à direita e então volta até achar o primeiro  $\square$  à esquerda, para se posicionar no a mais à esquerda e começar tudo de novo.

$$\delta(q_2, b) = (q_3, \square, E), \quad \delta(q_3, a) = (q_3, a, E), \quad \delta(q_3, b) = (q_3, b, E), \quad \delta(q_3, \square) = (q_0, \square, D).$$

Após um passo nesta parte da computação, a máquina terá efetuado uma computação parcial.

$$q_0 a^n b^n \vdash^* q_0 a^{n-1} b^{n-1}$$

de modo que um único “a” foi casado com um “b”. Após dois passos, teremos completado a computação parcial

$$q_0 a^{n-1} b^{n-1} \vdash^* q_0 a^{n-2} b^{n-2},$$

e assim por diante, indicando que o processo de casamento está sendo como o esperado.

Quando a entrada é a cadeia  $a^n b^n$ , a reescrita continua desta maneira, parando somente quando não houver mais a’s nem b’s para serem casados. Na procura do “a” mais à esquerda, o cabeçote vai para a esquerda no estado  $q_3$  e acha um  $\square$  para logo ir para o estado  $q_0$  esperando encontrar um “a” e recomeçar tudo de novo. Porém, agora, em vez de encontrar um a ele achará um outro  $\square$ . Logo, nesse ponto ele deveria enviar para o estado final, assim ainda devemos adicionar a transição

$$\delta(q_0, \square) = (q_4, \square, D).$$

**Exemplo 8.1.11** Projetar uma máquina de Turing que reconheça a linguagem

$$\mathcal{L} = \{a^n b^n c^n / n \geq 1\}.$$

As idéias usadas no exemplo anterior são facilmente adaptadas nesse caso. Casamos cada “a”, “b” e “c” apagando os a’s e c’s e marcando os b’s com x. Embora conceitualmente uma simples extensão do exemplo anterior, escrever esse programa é, como veremos a seguir, um pouco mais tedioso. Para facilitar a visualização da máquina de Turing, usaremos uma notação de tabela, onde a coluna representa o símbolo lido pelo cabeçote e as filas o estado interno da máquina. O conteúdo da tabela representa o movimento da máquina de Turing para esse símbolo e esse estado.

	$a$	$b$	$c$	$x$	$\square$
$q_0$	$(q_1, \square, D)$			$(q_5, \square, D)$	
$q_1$	$(q_1, a, D)$	$(q_2, x, D)$		$(q_1, x, D)$	
$q_2$		$(q_2, b, D)$	$(q_2, c, D)$		$(q_3, \square, E)$
$q_3$			$(q_4, \square, E)$		
$q_4$	$(q_4, a, E)$	$(q_4, b, E)$	$(q_4, c, E)$	$(q_4, x, E)$	$(q_0, \square, D)$
$q_5$				$(q_5, \square, D)$	$(q_6, \square, D)$
$q_6$					

O estado final é  $q_6$ .

Observe que mesmo  $\{a^n b^n\}$  sendo uma linguagem livre do contexto e  $\{a^n b^n c^n\}$  não sendo, elas podem ser reconhecidas por máquinas de Turing com estruturas similares.

Uma conclusão que podemos tirar deste exemplo é que uma máquina de Turing pode reconhecer linguagens que não são livres do contexto, uma primeira indicação de que as máquinas de Turing são mais poderosas do que os autômatos com pilha.

## 8.1. A Máquina de Turing Padrão

---

### 8.1.3 Máquinas de Turing como Tradutores (Computadores)

Tivemos poucos motivos até aqui para estudar os tradutores. Em teoria de linguagens reconhecedores são bastante adequados. Mas como veremos, as máquinas de Turing não são somente interessantes como reconhecedores, elas nos fornecem modelos abstratos simples de computadores digitais, em geral. Uma vez que o principal objetivo de um computador é transformar entradas em saídas, ele atua como um tradutor. Se pretendermos modelar computadores usando máquinas de Turing, temos de estudar esse aspecto com mais detalhes.

A entrada para uma computação é a cadeia, de todos os símbolos não brancos, dispostos na fita, de esquerda a direita, no início da computação. Na conclusão da computação num estado final, a saída será a cadeia que se encontrar na fita. Se a máquina pára num estado não final ou fica em laço infinito para uma determinada entrada, diremos que a máquina não está definida ou diverge para essa entrada. Portanto, podemos ver uma máquina de Turing tradutora  $M$  como uma implementação de uma função parcial  $f : \Sigma^* \rightarrow \Gamma^*$  definida por

$$f(w) = \hat{w}$$

uma vez que

$$q_0 w \vdash^* M \quad w_1 q_f w_2,$$

para algum estado final  $q_f \in F$  e  $\hat{w} = w_1 w_2 \in \Gamma^*$ .

**Definição 8.1.12** Uma função  $f$  com domínio  $D$  diz-se **Turing computável** ou, simplesmente, **computável** se existe alguma máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  tal que para todo  $w \in D$

$$q_0 w \vdash^* M \quad w_1 q_f w_2, \text{ com } q_f \in F \text{ e } f(w) = w_1 w_2$$

é claro que quando queremos computar uma função, digamos  $f : D \rightarrow E$ , usamos uma representação formal (como cadeias de uma linguagem formal) dos domínios  $D$  e  $E$ .

Como teremos oportunidade de ver, todas as funções matemáticas comuns, não importa quão complicadas elas sejam, são Turing computáveis. Começaremos olhando algumas operações simples, como a adição e a comparação aritmética.

**Exemplo 8.1.13** Dado dois inteiros positivos  $x$  e  $y$ , projetar uma máquina de Turing que compute  $x + y$ .

Primeiramente, temos que fazer alguma convenção para representar os inteiros positivos. Por simplicidade, usaremos a notação unária no qual qualquer inteiro positivo  $x$  é representado por  $\alpha(x) \in \{1\}^*$ , tal que

$$|\alpha(x)| = x.$$

Assim, 0 é representado pela cadeia vazia.

Devemos, também, decidir como  $x$  e  $y$  são dispostos inicialmente na fita e como sua soma deve aparecer no final da computação. Assumiremos que  $\alpha(x)$  e  $\alpha(y)$  estão na fita em notação unária, separados por um 0, com o cabeçote apontando para o símbolo mais à esquerda de  $\alpha(x)$ . Após a computação,  $\alpha(x + y)$  estará na fita seguido por um 0, e o cabeçote estará posicionado no lado esquerdo do resultado. Portanto, pretendemos projetar uma máquina de Turing para a computação

$$q_0\alpha(x)0\alpha(y)\vdash^* q_f\alpha(x+y)0,$$

onde  $q_f$  é um estado final. Construir um programa para isso é relativamente simples. Tudo que precisamos é mover o separador 0 para a direita, até o final de  $\alpha(y)$ , de modo que a adição nada mais seja do que concatenar as duas cadeias. Para isso, construímos

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle, \text{ com}$$

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$ ,
- $\Sigma = \{0, 1\}$ ,
- $\Gamma = \{0, 1, \square\}$ ,
- $F = \{q_4\}$ ,

	0	1	$\square$
$q_0$	$(q_1, 1, D)$	$(q_0, 1, D)$	
$q_1$		$(q_1, 1, D)$	$(q_2, \square, E)$
$q_2$		$(q_3, 0, E)$	
$q_3$		$(q_3, 1, E)$	$(q_4, \square, D)$
$q_4$			

Observe que ao mover o 0 para a direita criamos temporariamente um 1 extra. A transição  $\delta(q_2, 1) = (q_3, 0, D)$  é necessária para remover o 1 extra (que foi relembrado colocando a máquina no estado  $q_1$ ) para o fim da computação.

Isto pode ser visto na seqüência de descrições instantâneas para somar 111 com 11:

$$\begin{aligned} q_0111011 &\vdash 1q_011011 \vdash 11q_01011 \vdash 111q_0011 \vdash 1111q_111 \vdash 11111q_11 \vdash \\ &111111q_1\square \vdash 11111q_21 \vdash 1111q_310 \vdash 111q_3110 \vdash 11q_31110 \vdash 1q_311110 \vdash \\ &q_3111110 \vdash q_3\square111110 \vdash q_4111110. \end{aligned}$$

A notação unária, embora inconveniente para computações práticas, é bastante adequada para programação em máquinas de Turing. O programa resultante é mais curto e mais simples daquele se usarmos uma outra notação, tal como binária ou decimal.

## 8.1. A Máquina de Turing Padrão

---

Adicionar números é uma das operações fundamentais de qualquer computador, aquela que desempenha uma parte na síntese de instruções mais complicadas. Outras operações básicas são copiar e comparar cadeias. Essas também podem ser facilmente programadas em máquinas de Turing.

**Exemplo 8.1.14** *Projetar uma máquina de Turing que multiplique um número natural por dois. Isto em notação unária é o mesmo que projetar uma máquina de Turing a qual copie ou duplique cadeias de 1's. Mais precisamente, uma máquina de Turing que efetue a computação*

$$q_0 w \vdash^* q_f w w$$

para qualquer  $w \in \{1\}^*$ .

Para resolver este problema devemos implementar o seguinte processo intuitivo:

1. Trocar todo 1 por  $x$ .
2. Achar o  $x$  mais à direita e trocá-lo por 1
3. Mover-se para o extremo direito da região não branca e criar um 1
4. Repetir a etapa 2 e 3 até não existir mais  $x$ 's

Uma versão de máquina de Turing desse processo intuitivo é

	1	$x$	$\square$
$q_0$	$(q_0, x, D)$		$(q_1, \square, E)$
$q_1$	$(q_1, 1, E)$	$(q_2, 1, D)$	$(q_3, \square, D)$
$q_2$	$(q_2, 1, D)$		$(q_1, 1, E)$
$q_3$			

onde  $q_3$  é o único estado final. Isto pode parecer um pouco difícil de ver a primeira vista, por isso devemos testar o programa com uma cadeia simples como 11. A computação efetuada neste caso é

$$\begin{aligned} q_0 11 &\vdash x q_0 1 \vdash x x q_0 \square \vdash x q_1 x \vdash x 1 q_2 \square \vdash x q_1 11 \vdash q_1 x 11 \vdash 1 q_2 11 \vdash 11 q_2 1 \vdash 111 q_2 \square \vdash \\ &11 q_1 11 \vdash 1 q_1 111 \vdash q_1 1111 \vdash q_1 \square 1111 \vdash q_3 1111. \end{aligned}$$

**Exemplo 8.1.15** Sejam  $x$  e  $y$  dois números naturais em notação unária. Construir uma máquina de Turing que páre no estado final  $q_f$  se  $x \geq y$ , e páre no estado não final  $q_n$  se  $x < y$ . Mais especificamente, a máquina deve efetuar a computação

$$q_0 \alpha(x) 0 \alpha(y) \vdash^* q_f \alpha(x) 0 \alpha(y), \text{ se } x \geq y,$$

$$q_0 \alpha(x) 0 \alpha(y) \vdash^* q_n \alpha(x) 0 \alpha(y), \text{ se } x < y.$$

Para resolver este problema, podemos usar a idéia do exercício 8.1.10, com ligeiras modificações. Ao invés de casar a's e b's, casaremos o 1 mais à esquerda do 0 com o 1 mais à direita. Em vez de apagar os 1's casados os transformaremos em X's, pois depois precisaremos recompor o valor original. Isto se realizará até encontrarmos o 0. No fim do casamento, teremos na fita ou

$$X \dots X 1 \dots 10 X \dots X \quad \text{ou} \quad X \dots X 0 X \dots X \quad \text{ou} \quad X \dots X 01 \dots 1 X \dots X,$$

dependendo se  $x > y$ ,  $x = y$  ou  $y > x$ , respectivamente. No primeiro caso, acharemos  $\square$  quando tentarmos casar o último 1 à esquerda de 0 com o 1 mais à direita de 0. Isto pode ser usado para entrar no estado  $q_f$ . No segundo e terceiro casos, acharemos um X à esquerda de 0 quando tentarmos realizar o casamento. Se ainda encontrarmos um 1 na direita, então  $y > x$  e nesse caso usamos isso para entrar em outro estado  $q_n$ . Se o casamento for perfeito, isto é, não acharmos nenhum 1 na direita do 0, significará que  $x = y$  e, portanto, iremos ao estado  $q_f$ . A máquina de Turing que efetua essas operações é:

	1	0	X	$\square$
$q_0$	$(q_1, X, D)$	$(q_4, 0, D)$		
$q_1$	$(q_1, 1, D)$	$(q_1, 0, D)$	$(q_2, X, E)$	$(q_2, \square, E)$
$q_2$	$(q_3, X, E)$	$(q_7, 0, D)$		
$q_3$	$(q_3, 1, E)$	$(q_3, 0, E)$	$(q_0, X, D)$	
$q_4$	$(q_5, 1, D)$		$(q_7, 1, D)$	
$q_5$	$(q_5, 1, D)$		$(q_5, 1, D)$	$(q_6, \square, E)$
$q_6$	$(q_6, 1, E)$	$(q_6, 0, E)$	$(q_6, 1, E)$	$(q_n, \square, D)$
$q_n$				
$q_7$			$(q_7, 1, D)$	$(q_8, \square, E)$
$q_8$	$(q_8, 1, E)$	$(q_8, 0, E)$	$(q_8, 1, E)$	$(q_f, \square, D)$
$q_f$				

Este exemplo mostra máquinas de Turing que podem ser programadas para tomar decisões baseadas em comparações aritméticas. Esta espécie de decisão simples é comum em linguagens de máquina de computadores reais, onde entram cadeias de instruções alternadas, dependendo da saída de uma operação aritmética.

## 8.2 Combinação de Máquinas de Turing para Realizar Tarefas Complicadas

Mostramos, explicitamente, como algumas operações importantes, encontradas em todo computador, podem ser simuladas numa máquina de Turing. Desde que, nos computadores digitais, tais operações primitivas são blocos para construir instruções mais complexas, vamos ver como essas operações básicas podem ser combinadas numa máquina de Turing. Para mostrar como as máquinas de Turing podem ser combinadas, seguimos uma prática comum em programação. Começamos com uma descrição de alto nível, então refinamo-la sucessivamente até o programa estar na linguagem atual com a qual estamos trabalhando. Podemos descrever máquinas de

## 8.2. Combinação de Máquinas de Turing para Realizar Tarefas Complicadas

---

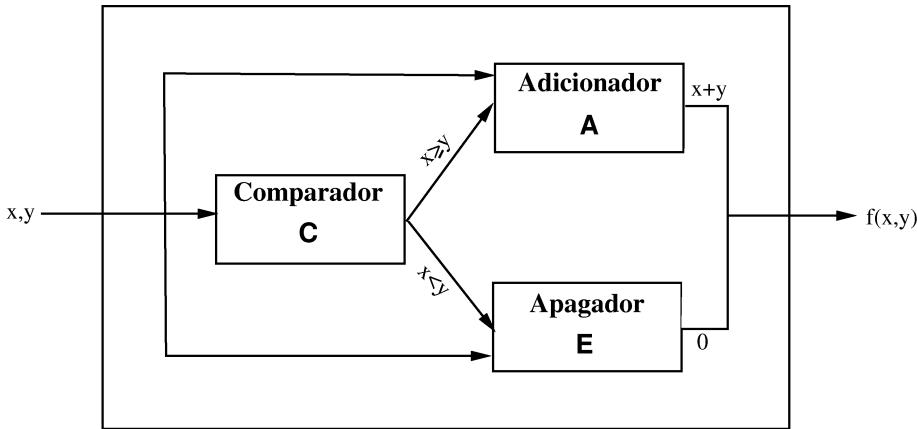


Figura 8.5: Exemplo de combinação de máquinas de Turing

Turing em alto nível de várias maneiras. Diagramas de bloco ou pseudo-códigos são as duas abordagens que freqüentemente usaremos nas discussões que seguem.

### 8.2.1 Diagramas de Blocos

Nos diagramas de bloco encapsulamos informações em caixas pretas cuja função é descrita, mas cujos detalhes interiores não são conhecidos. Ao usar tais caixas, argumentamos, implicitamente, que elas podem ser construídas. Como um primeiro exemplo, combinaremos as máquinas dos exemplo 8.1.13 e 8.1.15.

**Exemplo 8.2.1** Projetar uma máquina de Turing que computa a função

$$f(x, y) = \begin{cases} x + y & , \text{ se } x \geq y \\ 0 & , \text{ se } x < y \end{cases}$$

Suponhamos que  $x$  e  $y$  são números naturais em representação unária separados por um 0. O valor zero será representado pela cadeia vazia, por tanto se ambos ( $x$  e  $y$ ) são zeros, então a fita só terá o símbolo 0 com o resto da fita em branco.

A computação de  $f(x, y)$  pode ser visualizada em alto nível por meio do diagrama na figura 8.5

Este diagrama mostra que primeiro usamos uma máquina de comparação, tal como aquela do exemplo 8.1.15, para determinar se  $x \geq y$  ou não. Se for o caso, o comparador envia um sinal de partida para o adicionador do exemplo 8.1.13, que então computa  $x + y$ . Se não, isto é se  $x < y$ , um programa apagador é acionado para trocar todo 1 por branco.

Em discussões subsequentes, usaremos, freqüentemente, diagramas de blocos de alto nível representando máquinas de Turing. Ele é certamente mais rápido e claro do que o conjunto extensivo correspondente de δ's. Antes de aceitar esta visão de alto nível devemos justificá-la. O que significa, por exemplo, dizer que o comparador envia um sinal de partida para o

adicionador? Não existe nada na definição 8.1.1 que ofereça essa possibilidade. Entretanto ela pode ser dada de forma direta.

O programa  $C$  que compara está escrito de forma sugestivo no exemplo 8.1.15, usando uma máquina de Turing que tem estados indexados com  $C$ . Para o adicionador, usamos a idéia do exemplo 8.1.13, com estados indexados por  $A$ . Para o apagador  $E$ , construímos uma máquina de Turing tendo os estados indexados com  $E$ . As computações efetuadas por  $C$  são:

$$q_{C,0}\alpha(x)0\alpha(y) \vdash^* q_{A,0}, \text{ se } x \geq y,$$

e

$$q_{C,0}\alpha(x)0\alpha(y) \vdash^* q_{E,0}, \text{ se } x < y,$$

Se tomarmos  $q_{A,0}$  e  $q_{E,0}$  como os estados iniciais de  $A$  e  $E$ , respectivamente, vemos que  $C$  começa ou com  $A$  ou com  $E$ . A computação efetuada pelo adicionador  $A$  será

$$q_{A,0}\alpha(x)0\alpha(y) \vdash^* q_{A,f}\alpha(x+y)0$$

e aquela efetuada pelo apagador  $E$  será

$$q_{E,0}\alpha(x)0\alpha(y) \vdash^* q_{E,f}0.$$

O resultado é uma única máquina de Turing que combina as ações de  $C$ ,  $A$  e  $E$  como está indicado na figura 8.5.

Observe que nos exemplos 8.1.13 e 8.1.15, usados aqui, ambas máquinas param a computação deixando o cabeçote de leitura-escrita apontando para o símbolo mais à esquerda da saída. Esta convenção é necessária, quando se deseja combinar máquinas de Turing, pois, por definição, qualquer execução de qualquer máquina de Turing, começa com o cabeçote apontando o símbolo mais à esquerda. Assim, quando conclui a computação do *comparador*, ele deixa o cabeçote na posição correta para começar a execução do *adicionador* ou do *apagador*, segundo seja o caso.

### 8.2.2 Macroinstrução

Uma outra visão de alto nível das máquinas de Turing é aquela envolvendo pseudo-código. Em linguagem de programação, pseudo-código é uma maneira de esboçar uma computação usando frases descritivas cujos significados supostamente entendemos. Embora esta descrição não seja usada no computador, assumimos que podemos traduzi-la na linguagem apropriada quando necessário. Uma espécie simples de pseudo-código é exemplificado pela idéia de uma macroinstrução, que é uma única assertiva para uma seqüência de assertivas de baixo nível. Antes definimos macroinstruções em termos da linguagem de baixo nível. Então usamos as macro instruções num programa com a hipótese de que o código de baixo nível relevante está substituído para cada ocorrência da macroinstrução. Esta idéia é muito útil na programação em máquinas de Turing.

## 8.2. Combinação de Máquinas de Turing para Realizar Tarefas Complicadas

---

**Exemplo 8.2.2** Considere a macroinstrução

se a então  $q_j$  senão  $q_k$ ,

com a seguinte interpretação. Se a máquina lê “a”, então ela entra no estado  $q_j$  sem trocar o conteúdo da fita ou mover o cabeçote. Se o símbolo lido não é “a”, a máquina entra no estado  $q_k$  sem trocar nada.

Para implementar essa macroinstrução no estado  $q$  requeriremos várias etapas de uma máquina de Turing

$$\begin{aligned}\delta(q, a) &= (q_{j_0}, a, D), \\ \delta(q, b) &= (q_{k_0}, b, D) \text{ para todo } b \in \Gamma - \{a\}, \\ \delta(q_{j_0}, c) &= (q_j, c, E) \text{ para todo } c \in \Gamma, \\ \delta(q_{k_0}, c) &= (q_k, c, E) \text{ para todo } c \in \Gamma.\end{aligned}$$

Os estados  $q_{j_0}$  e  $q_{k_0}$  são novos, e foram introduzidos para levar em conta as complicações que aparecem do fato de na máquina de Turing padrão o cabeçote mudar de posição em cada movimento. Na macroinstrução queremos trocar o estado mas não o cabeçote, isto é, deixamo-lo onde estava. Deixamos o cabeçote mover-se para a direita, mas colocamos a máquina no estado  $q_{j_0}$  ou  $q_{k_0}$ . Isto indica que um movimento para a esquerda deve ser feito antes de entrar no estado desejado  $q_j$  ou  $q_k$ .

### 8.2.3 Subprogramas

Dando um passo a frente, podemos trocar as macroinstruções por subprogramas. Normalmente, uma macroinstrução é trocado pelo código atual em cada ocorrência, enquanto um subprograma é um único pedaço de código que é solicitado repetidamente quando necessário. Os subprogramas são fundamentais em linguagens de programação de alto nível, mas eles também podem ser usados com máquinas de Turing. Para tornar isso plausível, vamos esboçar rapidamente como uma máquina de Turing pode ser usada como um subprograma que pode ser solicitada repetidamente por outra máquina de Turing. Isto requer uma nova característica: a habilidade para armazenar informação na chamada configuração de programas, de modo que a configuração possa ser recriada no retorno do subprograma. Por exemplo, digamos que a máquina  $A$  no estado  $q_i$  solicita a máquina  $B$ . Quando  $B$  terminou, deveríamos reasumir o programa  $A$  no estado  $q_i$ , com o cabeçote (que pode ter sido movido durante as operações de  $B$ ) na sua posição original. Em outras vezes,  $A$  pode chamar  $B$  do estado  $q_j$ , no caso em que o controle deve retornar para esse estado. Para resolver o problema da transferência de controle, devemos ser capazes de passar informação de  $A$  para  $B$  e vice-versa, sermos capazes de recriar configuração de  $A$  quando ele recupera o controle de  $B$ , e assegurar que temporariamente as computações suspensas de  $A$  não sejam afetadas pela execução de  $B$ . Para isto, devemos dividir a fita em várias regiões como mostrado na figura 8.6.

Quando  $A$  solicita  $B$ , ele primeiro escreverá as informações necessárias (por exemplo, estado corrente de  $A$ ) na região  $T$ . Após essa transferência  $B$  usará  $T$  para encontrar sua entrada.

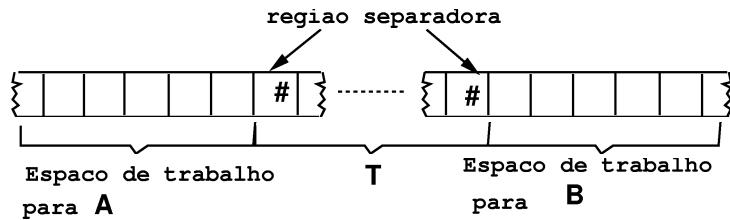


Figura 8.6: Divisão da fita para simulação de subprogramas em máquinas de Turing.

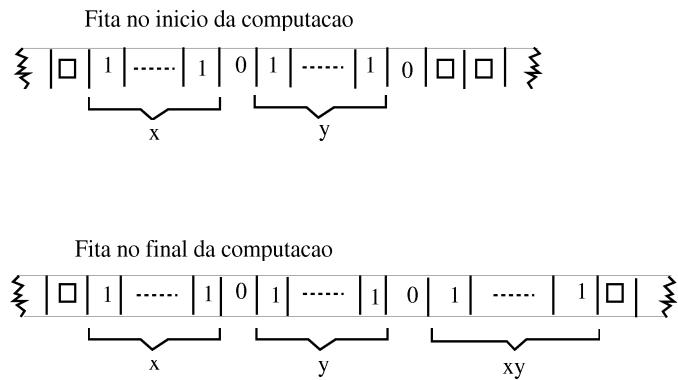


Figura 8.7: Conteúdo da fita para a multiplicação usando a idéia de subprograma.

O espaço de trabalho para  $B$  está separado de  $T$  e do espaço de trabalho de  $A$ , de modo que nenhuma interferência possa ocorrer. Quando  $B$  termina, ele retornará seu resultado para a região  $T$ , onde  $A$  o encontrará. Desta maneira, os dois programas podem interagir como desejado. Observe que isso é muito parecido com o que acontece no computador real quando um subprograma é solicitado.

Agora podemos programar máquinas de Turing em pseudo-códigos, uma vez que conhecemos (ao menos em tese) como traduzir este pseudo-código em um programa de máquina Turing atual.

**Exemplo 8.2.3** *Projetar uma máquina de Turing que multiplique dois inteiros positivos em notação unária.*

*Uma máquina de multiplicação pode ser construída combinando as idéias encontradas na adição e cópia. Vamos assumir que os conteúdos no inicio e final da fita são como indicados na figura 8.7. O processo de multiplicação pode ser visualizado como cópias repetidas do multiplicando  $y$  para cada 1 no multiplicador  $x$ . O pseudo-código seguinte mostra as principais etapas do processo*

1. Repita as seguintes etapas até não conter mais 1's.
  - (a) achar um 1 em  $x$  e trocá-lo por um outro símbolo, digamos “a”,
  - (b) troque o  $\square$  mais à direita por  $y$ .

### 8.3. Procedimentos efetivos e Algoritmos

---

2. Troque todos os  $a$ 's com 1's.

Portanto a máquina de Turing é a seguinte

	1	0	$a$	$\square$
$q_0$	$(q_1, a, D)$	$(q_7, 0, E)$		
$q_1$	$(q_1, 1, D)$	$(q_2, 0, D)$		
$q_2$	$(q_3, a, D)$	$(q_5, 0, E)$		
$q_3$	$(q_3, 1, D)$	$(q_3, 0, D)$		$(q_4, 1, E)$
$q_4$	$(q_4, 1, E)$	$(q_4, 0, E)$	$(q_2, a, D)$	
$q_5$		$(q_6, 0, E)$	$(q_5, 1, E)$	
$q_6$	$(q_6, 1, E)$		$(q_0, a, D)$	
$q_7$			$(q_7, 1, E)$	$(q_8, \square, D)$
$q_8$				

onde  $F = \{q_8\}$ .

Apesar da natureza descritiva desses exemplos, eles já nos possibilita conjecturar que máquinas de Turing, embora em princípio bem rudimentares, podem ser combinadas de muitas maneiras de modo a torná-las bastante poderosas. Nossos exemplos não foram bastante gerais e detalhados para nos permitir argumentar que provamos alguma coisa. Mas, é plausível nesse ponto conjecturar que as máquinas de Turing podem fazer tarefas muito complicadas.

## 8.3 Procedimentos efetivos e Algoritmos

A noção de procedimentos efetivo é uma noção intuitiva que tenta refletir processos estritamente mecânicos, i.e. processos que se seguidos a risca sempre proporcionam o mesmo resultado. O interesse por este tipo de processos é antigo, de fato já os gregos antigos tentaram encontrar processos efetivos para algumas operações matemáticas, por exemplo o algoritmo da divisão de Euclides é um tal processo. A forma como aprendemos a somar, subtrair e multiplicar números na escola são também exemplos de procedimentos efetivos.

**Definição 8.3.1** Um **procedimento efetivo** é uma descrição finita e não ambígua de um conjunto finito de operações as quais devem ser efetivas, no sentido que exista um procedimento estritamente mecânico para realizar estas.

Observe que uma destas operações efetivas num procedimento efetivo pode ser do tipo “va para a operação n”. Assim, o conjunto finito de operações pode descrever um conjunto infinito de passos computacionais (execução das operações). Um conceito estreitamente ligado ao de procedimento efetivo é o de algoritmo.

**Definição 8.3.2** Um **algoritmo** é um procedimento efetivo que especifica uma seqüência de operações as quais sempre param.

Observe que alguns autores como Cutland [Cut80] não fazem distinção entre estes dois conceitos, e portanto algoritmo seria sinônimo de procedimento efetivo. Já outros autores como Brainerd e Landweber [BL74], Lucchesi et al [LSS79], Santiago e Bedregal [SB04],

## 8.4 Tese de Turing

A discussão anterior não somente mostrou como uma máquina de Turing pode ser construída de partes mais simples, mas destaca um aspecto negativo ao se trabalhar com tais autômatos de baixo nível. Enquanto necessitemos de pouca imaginação ou engenhosidade para traduzir um diagrama de bloco ou pseudo código no programa da máquina de Turing correspondente, fazê-lo pode tomar muito tempo ou facilmente levar a erro e não acrescentar nada ao nosso entendimento. O conjunto de instruções de uma máquina de Turing é tão restrito que qualquer argumento, solução ou prova para um problema não trivial é muito tedioso.

Agora enfrentamos um dilema: queremos argumentar que as máquinas de Turing podem efetuar não somente operações simples para as quais oferecemos programas explícitos, porém processos mais complexos, descritíveis por diagramas de blocos ou pseudo-códigos. Para defender tais argumentos devemos mostrar explicitamente programas relevantes, mas isso é bastante trabalhoso. Desse modo, gostaríamos de achar uma maneira de efetuar uma discussão razoavelmente rigorosa de máquinas de Turing sem ter que escrever o código de baixo nível. Não existe infelizmente maneira completamente satisfatória de fazer isso. O melhor que podemos fazer é manter um compromisso razoável. Para ver como devemos alcançar tal compromisso, retomaremos alguns fundamentos filosóficos.

Podemos tirar algumas conclusões simples dos exemplos vistos anteriormente. A primeira é que as máquinas de Turing parecem ser mais poderosas que os autômatos com pilha. No exemplo 8.1.11, esboçamos a construção de uma máquina de Turing para uma linguagem que não é livre do contexto e para a qual, consequentemente, não existe um autômato com pilha. Os exemplos 8.1.13, 8.1.14 e 8.1.15 mostram que as máquinas de Turing podem fazer algumas operações aritméticas simples, manipular cadeias e fazer algumas comparações. A discussão também ilustra como as operações podem ser combinadas para resolver problemas mais complexos, como várias máquinas de Turing podem ser compostas, e como um programa pode atuar como subprograma de um outro. Como operações muito complexas podem ser construídas dessa maneira, podemos suspeitar que as máquinas de Turing podem modelar em poder um computador típico.

Suponha que fôssemos fazer a conjectura de que, em algum sentido, as máquinas de Turing são iguais em poder a um computador típico. Como podemos defender ou refutar tal hipótese? Para defendê-la deveríamos tomar uma seqüência de problemas crescentemente mais complicados e mostrar como eles são resolvidos por máquinas de Turing. Devemos também tomar o conjunto de instruções da linguagem de máquina de algum computador específico e projetar uma máquina de Turing que possa efetuar todas as instruções no conjunto. Embora isto poderia torrar nossa paciência, deve ser possível, em princípio, se nossa hipótese for correta. Além disso, enquanto todo sucesso nessa direção fortaleceria nossa convicção na verdade da hipótese, ela não levaria a uma prova. A dificuldade está no fato de que não sabemos exatamente o que queremos dizer com “um computador típico” e não temos meios de tornar essa definição precisa.

Podemos também abordar o problema de outra perspectiva. Podemos tentar encontrar algum procedimento para o qual podemos escrever um programa de computador, mas para o qual possamos mostrar que não pode existir nenhuma máquina de Turing. Se isto fosse possível teríamos uma base para rejeitar a hipótese. Mas ninguém ainda foi capaz de produzir um contra-exemplo. O fato de que todas as tentativas de contradizer a hipótese foram infrutíferas deve ser tomada como uma evidência circunstancial de que tal contra-exemplo não pode ser dado. Tudo indica que as máquinas de Turing são, em princípio, tão poderosas quanto qualquer computador.

## 8.4. Tese de Turing

---

Uma outra possibilidade seria tentar definir um modelo formal ou máquina que seja mais poderosa que as máquinas de Turing. Porém, todos os modelos de máquinas ou formalismos que surgiram se mostraram serem equivalentes ou inferiores em poder computacional que as máquinas de Turing. Isto levou a Alan Turing e outros, em torno de 1936, à tão conhecida **Tese de Turing**<sup>1</sup>. Esta hipótese estabelece que qualquer computação que pode ser efetuada por meios mecânicos pode ser efetuada por uma máquina de Turing.

é importante ter em mente o que é a tese de Turing. Ela não é algo que possa ser provado. Para fazer isso teríamos de definir precisamente o termo “meios mecânicos”. Isto requereria algum outro modelo abstrato e não nos levaria muito mais longe do que anteriormente. A tese de Turing é vista mais apropriadamente como uma definição do que constitui uma **computação mecânica**: uma computação é mecânica se e somente se ela pode ser efetuada por alguma máquina de Turing.

Se tomarmos essa atitude e encararmos a tese de Turing, simplesmente, como uma definição, podemos colocar a questão de o quanto essa definição é suficientemente abrangente. Ela é bastante para cobrir tudo que fizemos até hoje (e concebivelmente factível no futuro) com computadores? Um “sim” inequívoco não é possível, mas as evidências em seu favor são muito forte. Alguns argumentos para aceitar a tese de Turing como a definição de uma computação mecânica são:

1. Qualquer coisa que pode ser efetuada em qualquer computador existente também pode ser efetuada por uma máquina de Turing.
2. Ninguém foi capaz ainda de sugerir um problema, solúvel, pelo o que intuitivamente entendemos por um algoritmo, para o qual um programa de máquina de Turing não possa ser escrito para ele.
3. Tem sido propostos modelos alternativos de computação mecânica, mas nenhum deles são mais poderosos do que o modelo da máquina de Turing.

Esses argumentos são circunstanciais, e a tese de Turing não pode ser provada usando eles. Apesar de sua plausibilidade, a tese de Turing é ainda uma hipótese. Mas encarar a tese de Turing como, simplesmente, uma definição arbitrária deixa de fora um importante ponto. Em algum sentido, a tese de Turing desempenha o mesmo papel em ciência da computação como fazem as leis básicas da física e da química. A física clássica, por exemplo, é baseada fortemente nas leis do movimento de Newton. Embora as chamemos de leis, elas não são logicamente necessárias. Em contrapartida elas são modelos plausíveis que explicam o mundo físico. Aceitamo-las porque as conclusões que tiramos delas concordam com nossa experiência e nossa observação. Tais leis não podem ser provadas verdadeiras, embora elas poderiam ser invalidadas. Se um experimento resulta na contradição de uma conclusão baseadas nestas leis, começariam a questionar a validade de tais leis. Por outro lado, repetidos insucessos de invalidar uma lei fortalece nossa confiança nela. Esta é a situação para a tese de Turing. Portanto, temos razão em considerá-la uma lei básica da ciência da computação. As conclusões que tiramos dela concordam com o

---

<sup>1</sup>Esta tese também é conhecida como “Tese de Church” ou “Tese de Church-Turing”, em homenagem a Alonzo Church quem em 1936, usando um outro formalismo (as funções  $\lambda$ -definíveis), foi o primeiro a identificar o conceito intuitivo de função efetivamente calculável ou computável com um conceito formal.

que sabemos acerca dos computadores reais, e até aqui, todas as tentativas para invalidá-la resultaram em falhas. Existe sempre a possibilidade de que alguém apareça com outra definição que explique alguma situação sutil não coberta por máquinas de Turing, mas que ainda caia no escopo de nossa noção intuitiva de computação mecânica. Em tal eventualidade, algumas de nossas discussões subseqüentes teriam que ser consideravelmente modificadas. Entretanto, a possibilidade disso acontecer é muito remota. Algumas discussões mais aprofundadas sobre a validade da tese de Church-Turing, considerando argumentos a favor e em contra, podem ser encontradas em [Tho73, Sob87, Bir96].

Tendo aceito a tese de Turing, estamos numa posição para dar uma definição precisa de algoritmo.

**Definição 8.4.1** *Um algoritmo para uma função  $f : D \rightarrow R$  é uma máquina de Turing  $M$ , para a qual dado qualquer entrada  $d \in D$  na sua fita, ela pára com a resposta correta  $f(d) \in R$  na fita. Especificamente podemos exigir que*

$$q_0 d \vdash^* M q_f f(d), \quad q_f \in F, \text{ para todo } d \in D.$$

Identificar um algoritmo com uma máquina de Turing nos permite provar rigorosamente argumentos como “existe um algoritmo...” ou “não existe qualquer algoritmo...”. Entretanto, para construir explicitamente um algoritmo, mesmo para problemas relativamente simples, pode ser bastante trabalhoso. Para evitar tal trabalho podemos apelar para a tese de Turing e alegar que qualquer coisa que podemos fazer com qualquer computador pode também ser feito por uma máquina de Turing. Conseqüentemente, podemos substituir as máquinas de Turing por um programa em JAVA, por exemplo, na definição 8.4.1. Isto facilita a exibição do algoritmo consideravelmente. Realmente, como já fizemos anteriormente, daremos um passo à frente e aceitaremos as descrições verbais ou diagramas de bloco como algoritmos na suposição de que escreveríamos um programa de máquina de Turing para eles se para isso fôssemos desafiados. Isto simplifica imensamente a discussão, mas é óbvio que algo crítico fica em aberto. Enquanto um “programa em JAVA” é bem definido, a “descrição verbal clara” não o é, e estamos em perigo de alegar a existência de um algoritmo não existente. Mas esse perigo é compensado pelo fato de podermos manter a discussão simples e intuitivamente clara, e podermos dar descrições concisas para processos complexos. Quem tiver qualquer dúvida da validade desse argumento pode esclarecê-lo escrevendo um programa adequado em alguma linguagem de programação.

## 8.5 Ligeiras Variações da Máquina de Turing Padrão

Nossa definição de máquina de Turing padrão é algo arbitrária no sentido de que existem definições alternativas que serviriam igualmente bem. As conclusões que podemos tirar sobre o poder de uma máquina de Turing são independentes da estrutura específica escolhida para ela.

Se aceitarmos a tese de Turing, complicações da máquina de Turing padrão dadas por dispositivos de armazenamento mais complexos não terão nenhum efeito no poder do autômato. Qualquer computação que pode ser feita por tais novos arranjos cairá sob a categoria de uma

## 8.5. Ligeiras Variações da Máquina de Turing Padrão

---

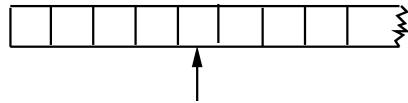


Figura 8.8: Fita ilimitada somente à direita.

computação mecânica e portanto pode ser feita pela máquina de Turing padrão. Existem muitas variações sobre o modelo básico dado na definição 8.1.1. Todas essas variações não aumentam o poder da computação. Se um modelo não padrão reconhece uma linguagem ou computa uma função existe um modelo padrão que também reconhece essa linguagem ou computa essa função. A seguir apresentaremos rapidamente algumas das possíveis variações do modelo padrão.

### 8.5.1 Máquinas de Turing com uma Opção de Permanência

Em nossa definição de máquinas de Turing padrão, o cabeçote deve se mover para a direita ou esquerda. As vezes é conveniente fornecer uma terceira opção, mantendo o cabeçote no mesmo lugar após reescrever o conteúdo da célula. Portanto definimos uma máquina de Turing com opção de permanência trocando o  $\delta$ , da definição 8.1.1, por

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{E, D, P\}$$

com a interpretação de que  $P$  significa que o cabeçote permanece no mesmo lugar. Claramente, toda máquina de Turing padrão é uma máquina de Turing com opção de permanência e pela tese de Turing, a reversa também é válida. Assim, esta opção não aumenta o poder do autômato, isto é,

- A classe das máquinas de Turing com opção de permanência é equivalente à classe das máquinas de Turing padrão.

Simulação é uma técnica comum para mostrar a equivalência de autômatos. Dizemos que um autômato **simula** outro se ele efetua as operações efetuadas por esse outro. Ou seja, ele se comporta como se fosse outro. Uma simulação de esta nova instrução na máquina de Turing Padrão foi vista no exemplo 8.2.2.

### 8.5.2 Máquinas de Turing com uma Fita Semi-infinita

Muitos autores não consideram o modelo da definição 8.1.1 como padrão, mas usam um outro com uma fita que é ilimitada somente numa direção. Podemos visualizar isto como uma fita que tem um limite à esquerda (veja figura 8.8). Esta máquina de Turing é por outro lado idêntica ao nosso modelo padrão, exceto que não é permitido o movimento à esquerda quando o cabeçote está na fronteira.

Não é difícil ver que esta restrição não afeta o poder da máquina, pois, se precisarmos de mais algum espaço à esquerda da fita, basta realizar um deslocamento de um quadradinho para a direita a todo o conteúdo da fita.

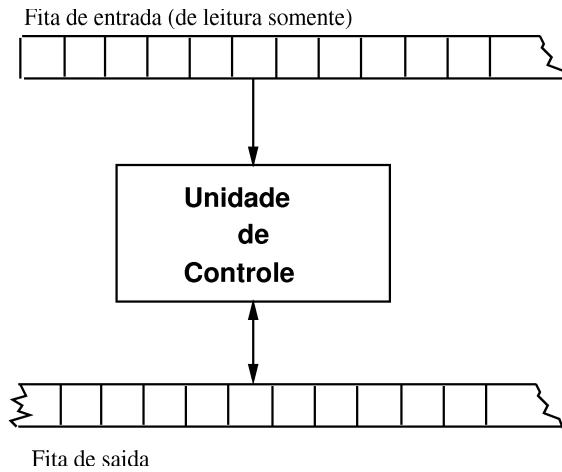


Figura 8.9: Representação esquemática de uma máquina de Turing off-line.

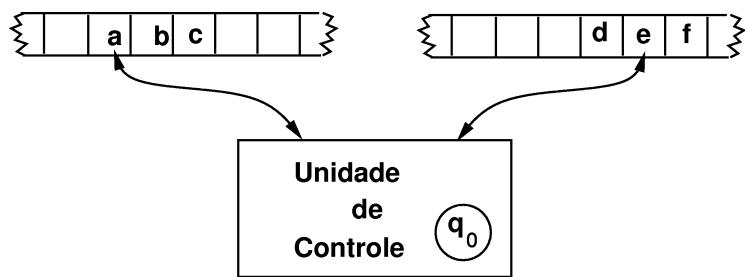


Figura 8.10: Representação esquemática de uma máquina de Turing com duas fitas.

### 8.5.3 Máquinas de Turing Off-line

A definição geral de um autômato contém uma fita de entrada assim como uma memória temporária. Na definição 8.1.1 descartamos a fita de entrada por razões de simplicidade, alegando que isso não faz diferença no conceito de máquinas de Turing.

Se colocarmos a fita de entrada de volta no desenho, obtemos o que é conhecido como **máquinas de Turing off-line**. Em tal máquina, cada movimento é governado pelo estado interno, o que é correntemente lido da fita de entrada, o que é visto pelo cabeçote de leitura-escrita. Uma representação esquemática de uma máquina de Turing off-line é mostrada na figura 8.9.

### 8.5.4 Máquinas de Turing com Múltiplas Fitas

Uma máquina de Turing com múltiplas fitas é, como o nome diz, uma máquina de Turing disposta de várias fitas, cada uma com seu cabeçote de leitura-escrita controlados independentemente, como ilustra a figura 8.10.

## 8.5. Ligeiras Variações da Máquina de Turing Padrão

---

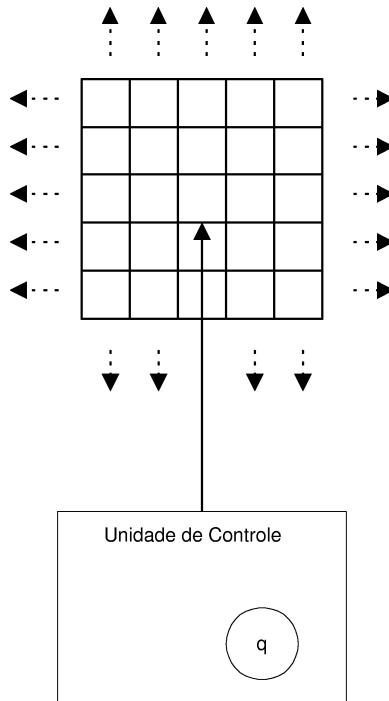


Figura 8.11: Esquema de uma máquina de Turing bi-dimensional.

Tipicamente definimos uma máquina  $n$ -fitas por  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$ , onde  $Q, \Sigma, \Gamma, q_0$ , e  $F$  são como na definição 8.1.1, mas onde

$$\delta : Q \times \Gamma^n \longrightarrow Q \times \Gamma^n \times \{L, R\}^n$$

especifica o que acontece sobre todas as fitas. Por exemplo, se  $n = 2$ , com uma configuração corrente mostrada na figura 8.10 então

$$\delta(q_0, a, e) = (q_1, x, y, L, D)$$

é interpretado como segue. A regra de transição pode ser aplicada somente se a máquina está no estado  $q_0$  e o primeiro cabeçote vê um  $a$  e o segundo um  $e$ . O símbolo na primeira fita será então trocado por  $x$  e seu cabeçote se moverá para a esquerda. Ao mesmo tempo, o símbolo na segunda fita é reescrito como  $y$  e o cabeçote se move para a direita. A unidade de controle muda do estado  $q_0$  para o estado  $q_1$ .

### 8.5.5 Máquinas de Turing Multidimensional

Uma máquina de Turing multidimensional é aquela na qual a fita pode ser estendida infinitamente em mais de uma dimensão. Um diagrama de máquina de Turing bi-dimensional é mostrada na figura 8.11.

A definição formal de uma máquina de Turing bi-dimensional envolve uma função de transição  $\delta$  da forma

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, U, D\},$$

onde  $U$  e  $D$  especificam movimentos do cabeçote para uma célula para acima e uma para abaixo, respectivamente.

### 8.5.6 Máquinas de Turing Não-Determinísticas

Embora a hipótese de Turing seja plausível quando se altera a estrutura da fita, o mesmo não se pode dizer de não-determinismo. Isto porque não-determinismo envolve um elemento de escolha e portanto tem algo de não mecanicista. Assim, o apelo à tese de Turing não é apropriado. Devemos olhar o não-determinismo com mais detalhe se quisermos argumentar que ele não acrescenta nada ao poder de uma máquina de Turing.

**Definição 8.5.1** Uma máquina de Turing não-determinística é um autômato como dado pela definição 8.1.1, exceto que  $\delta$  é agora uma função

$$\delta : Q \times \Gamma \longrightarrow \wp^{fin}(Q \times \Gamma \times \{L, R\}).$$

Como sempre, quando está envolvido não-determinismo o conjunto de valores de  $\delta$  é um conjunto de transições possíveis, cada uma das quais, pode ser escolhida pela máquina.

**Exemplo 8.5.2** Seja uma máquina de Turing não-determinista com a transição

$$\delta(q_0, a) = \{(q_1, b, D), (q_2, c, E)\}$$

Então os movimentos

$$q_0aaa \vdash b q_1aa \quad \text{e} \quad q_0aaa \vdash q_2 \square caa$$

são ambos possíveis.

Como não está claro qual o papel que o não-determinismo desempenha em computação de funções, os autômatos não-determinísticos são usualmente vistos como reconhecedores. Uma máquina de Turing não-determinística diz-se aceitar  $w$  se existe qualquer sequência possível de movimentos tal que

$$q_0 w \vdash^* x_1 q_f x_2,$$

com  $q_f \in F$ . Uma máquina de Turing não-determinística pode ter movimentos disponíveis que levam a um estado não-final ou a um laço infinito. Mas, como sempre acontece com não-determinismo, essas alternativas são irrelevantes.

- A classe das máquinas de Turing (reconhecedoras) determinísticas e a classe das máquinas de Turing (reconhecedoras) não-determinísticas são equivalentes.

## 8.6 Máquina de Turing Universal

Considere o seguinte argumento contra a tese de Turing: “uma máquina de Turing, como apresentado na definição 8.1.1, é um computador de propósito específico ou seja um sistema embarcado. Uma vez que a função  $\delta$  seja definida, a máquina fica restrita a efetuar um tipo particular de computação. Os computadores digitais, por outro lado, são máquinas de propósitos gerais que podem ser programadas para fazer diferentes tarefas em tempos diferentes. Conseqüentemente, as máquinas de Turing não podem ser consideradas equivalentes aos computadores digitais de propósitos gerais.”

Esta objeção pode ser superada projetando-se uma máquina de Turing reprogramável, chamada **máquina de Turing universal**. Uma máquina de Turing universal  $M_U$  é um autômato tal que dado como entrada a descrição de qualquer máquina de Turing  $M$  e uma cadeia  $w$ , ele pode simular a computação de  $M$  sobre  $w$ . Para construir tal  $M_U$  primeiro escolhemos uma maneira padrão de descrever máquinas de Turing. Podemos sem perda de generalidade, assumir que

$$Q = \{q_1, q_2, \dots, q_n\},$$

com  $q_1$  o estado inicial,  $q_2$  o estado final, e

$$\Gamma = \{a_1, a_2, \dots\},$$

onde  $a_1$  representa o branco. Então, escolhemos uma codificação no qual  $q_1$  é representado por 1,  $q_2$  por 11, e assim por diante. Analogamente,  $a_1$  é codificado como 1,  $a_2$  como 11, etc. Com os estados inicial e final, e o símbolo branco definidos por esta convenção, qualquer máquina de Turing pode ser descrita completamente, se usarmos o símbolo 0 como um separador entre os 1's. A função de transição é codificada segundo esse esquema, com argumentos e resultados em alguma seqüência esperada. Por exemplo,  $\delta(q_1, a_2) = (q_2, a_3, E)$  pode aparecer como

$$\dots 1011011011101\dots$$

Disto segue que qualquer máquina de Turing tem uma codificação como uma cadeia sobre  $\{0, 1\}^+$ , e que, dado a codificação de uma máquina de Turing,  $M$ , qualquer, podemos decodificá-la unicamente. Algumas cadeias não representarão qualquer máquina de Turing (por exemplo, a cadeia 00011), mas isso não é relevante. Assim, podemos pensar na existência de uma bijeção  $G$  entre o conjunto das máquinas de Turing  $\mathcal{M}$  e o conjunto dos números naturais  $\mathbb{N}$  ou numa codificação deles (por exemplo representação em números binários), esta bijeção é conhecida como **codificação ou numeração de Gödel**.

Uma máquina de Turing  $M_U$ , então, tem um alfabeto de entrada que inclui  $\{0, 1\}$  e uma estrutura de uma máquina multi-fita, como mostrado na figura 8.12.

Para qualquer entrada  $M$  e  $w$ , a fita 1 manterá a definição codificada de  $M$ , isto é  $G(M)$ . A fita 2 terá o conteúdo da fita de  $M$ , e a fita 3 os estados internos de  $M$ .  $M_U$  olha primeiro os conteúdos das fitas 2 e 3 para determinar a configuração de  $M$ . E então consulta a fita 1 para ver o que  $M$  faria com essa configuração. Finalmente, as fitas 2 e 3 serão modificadas para refletir o resultado do movimento.

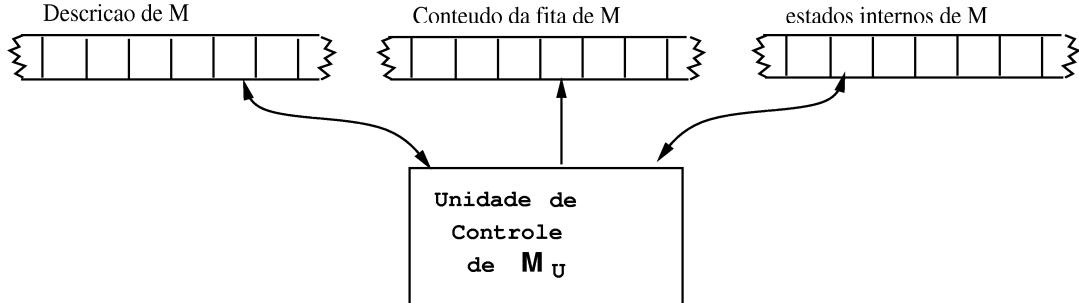


Figura 8.12: Representação esquemática de uma máquina de Turing universal.

É portanto razoável a construção de uma máquina de Turing universal, mas o processo não tem interesse. Preferimos em vez disso apelar para a tese de Turing. A implementação pode ser feito usando-se alguma linguagem de programação.

A observação de que toda máquina de Turing pode ser representada por uma cadeia de 0's e 1's tem implicações importantes. Por exemplo, ela nos permite associar uma ordem ao conjunto de todas as máquinas de Turing, de tal modo que a cada inteiro positivo  $i$ , podemos associar uma única máquina de Turing  $M_i$ . Ao mostrar como isso pode ser feito, introduzimos o conceito de um *procedimento de enumeração*. Um procedimento de enumeração para um conjunto  $S$  é uma máquina de Turing que escreverá na sua fita os elementos de  $S$  em alguma seqüência  $s_1, s_2, \dots$  de tal maneira que qualquer  $s \in S$  será escrito na fita ao cabo de um número finito de etapas.

**Definição 8.6.1** Seja  $S$  um conjunto de cadeias sobre um alfabeto  $\Sigma$ . Um **procedimento de enumeração** para  $S$  é uma máquina de Turing que pode efetuar a seqüência de etapas

$$q_0 \square \vdash^* q_s x_1 \# s_1 \vdash^* q_s x_2 \# s_2 \dots,$$

onde  $x_i \in \Gamma - \{\#\}$ ,  $s_i \in S$ , de tal modo que qualquer  $s \in S$  é produzido num número finito de etapas. O estado  $q_s$  é um estado significando membro de  $S$ , isto é, quando  $q_s$  entra, a cadeia seguindo  $\#$  deve estar em  $S$ .

Estritamente falando, um procedimento de enumeração não pode ser chamado um algoritmo, uma vez que ele não terminará quando  $S$  for infinito. Entretanto, ele pode ser considerado um processo significativo, porque produz resultados previstos e bem-definidos.

Um conjunto infinito diz-se **contável** se seus elementos podem ser postos em uma correspondência um a um com os inteiros positivos. Por isso queremos significar que os elementos do conjunto podem ser escritos numa seqüência  $s_1, s_2, \dots$  de tal maneira que qualquer elemento do conjunto tem um índice finito. Nem todo conjunto é contável, os reais por exemplo. Mas deve ficar claro que qualquer conjunto para o qual existe um procedimento de enumeração é contável, uma vez que o procedimento de enumeração fornece a seqüência requerida. Porém a recíproca não é verdadeira, isto é, existem alguns conjuntos contáveis para os quais não existe nenhum procedimento de enumeração.

## 8.6. Máquina de Turing Universal

---

**Exemplo 8.6.2** Seja  $\Sigma = \{a, b, c\}$ . Podemos mostrar que o conjunto  $\Sigma^+$  é contável se pudermos achar um procedimento de enumeração que produz seus elementos em alguma seqüência. Uma ordem na qual poderiam aparecer as cadeias em  $\Sigma^+$  é a ordem léxica, isto é,

$$a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \dots$$

um procedimento de enumeração para esta seqüência de cadeias pode ser descrito pelo seguinte pseudo-código

1. Gerar  $\#a$
2. Repetir o seguinte fato ad infinitum
  - (a) duplicar o conteúdo da fita desde o  $\#$  mais à direita até o símbolo em  $\Sigma$ , mais à direita.
  - (b) dar o “sucessor” da cadeia (na ordem léxica) à direita do  $\#$ , mais à direita.

O passo mais difícil neste pseudo-código é o 2.(b). Uma máquina de Turing  $S$  que realize esta macro deveria fazer a seguinte computação

$$q_{S,0} \# w \vdash^* S q_{S,f} \# suc(w).$$

Uma tal máquina de Turing deve ir até o símbolo mais à direita de  $w$ , se for “a” ou “b”, substituí-lo por “b” ou “c”, respectivamente. Se for “c”, substitui-lo por “a” e ir para a esquerda, repetindo este comportamento até achar um “a” e substitui-lo por “b” ou um “b” e substitui-lo por um “c”. Mas se  $w = c^k$ , então ele vai substituindo os  $c$ ’s por  $a$ ’s até achar  $\#$ , nesse caso vai para a direita e substitui o “a” por um “b”, logo vai até o branco mais à direita e coloca um “a”. Finalmente, a máquina de Turing deve retornar até o  $\#$  mais à direita.

A seguinte máquina de Turing, com opção de permanência, faz isto. O estado final é  $q_4$ .

	$a$	$b$	$c$	$\#$	$\square$
$q_0$	$(q_0, a, D)$	$(q_0, b, D)$	$(q_0, c, D)$	$(q_0, \#, D)$	$(q_1, \square, E)$
$q_1$	$(q_2, b, E)$	$(q_2, c, E)$	$(q_1, a, E)$	$(q_3, \#, D)$	
$q_2$	$(q_2, a, E)$	$(q_2, b, E)$	$(q_2, c, E)$	$(q_4, \#, S)$	
$q_3$	$(q_3, a, D)$				$(q_2, a, E)$
$q_4$					

Uma consequência importante da discussão acima é o seguinte teorema.

**Teorema 8.6.3** O conjunto de todas as máquinas de Turing, apesar de infinito, é contável.

**DEMONSTRAÇÃO:** Codificamos todas as máquinas de Turing usando 0 e 1, como descrito na construção da máquina de Turing universal. Com esta codificação construímos o seguinte procedimento de enumeração.

1. Gerar 0
2. Verificar se a cadeia gerada é uma codificação de uma máquina de Turing. Se assim for, escreva esta na fita de saí da na forma requerida na definição 8.6.1. Senão o ignore.
3. Gerar a próxima cadeia em  $\{0, 1\}^+$ , na ordem léxica.
4. Retornar ao passo 2.

Como qualquer máquina de Turing tem uma descrição finita, qualquer máquina específica será gerada por este processo em algum tempo finito. ■

## 8.7 Algumas Limitações da Tese de Church-Turing

É claro que a tese de Church-Turing é restrita e facilmente se poderia ir além dela mudando alguns dos preceitos embutidos na tese. Por exemplo, se trabalharmos com computações analógicas que obtém seus dados a partir de sensores também analógicos e que portanto podem ter precisão ilimitada, ou seja não representáveis de maneira fiel pelos computadores digitais, então teriam que ser consideradas computações sobre objetos ideais (números reais). Isto demandaria uma outra teoria da computabilidade, denominada de hipercomputabilidade) a qual tem sido estudada amplamente com diversos enfoques por vários autores, por exemplo [Lof70, BSS89, Bra96, BA99, Wei00].

Uma outra possibilidade é abandonar o preceito que associa computabilidade à computação de funções, ou seja que vê programas como caixas pretas, onde no inicio se lhe é fornecido valores de entrada, os quais o programa os transforma em saídas. Certamente, nem toda a computação digital é dessa forma, por exemplo jogos, prevê interações entre o usuário e o programa, de tal forma que o usuário entre um movimento ao qual o programa reage com um outro movimento e o usuário reage a esse novo movimento, etc. Ou um editor de texto, que na medida que voce vai digitando o editor já vai identificando e sinalizando eventuais erros ortográficos. Isto tem levado a estender a noção de máquinas de Turing para **máquinas de Turing persistente** a qual realiza uma seqüência infinita de computações de uma máquina de Turing não-determinística com 3 fitas, uma das quais é usada para armazenar o conteúdo de uma computação para ser repassada para a seguinte computação [GSAS04]. Outras abordagens para este mesmo paradigma, podem ser encontrados, por exemplo em [Weg97, CD05].

Ainda paradigmas computacionais, como computação quântica, computação molecular ou com DNA e lógica difusa têm associadas novas noções de computabilidade que vão além da tese de Church-Turing. No caso de computação quântica diversos modelos de **máquinas de Turing quânticas** têm sido propostos (por exemplo, [Deu85, Car03, PJ06]). O mais simples dele, é o proposto por [Deu85]. As máquinas de Turing quântica de Deutsch são mais poderosas que as máquinas de Turing, pois por exemplo, podem gerar números verdadeiramente randômicos que não podem ser gerados por máquinas de Turing convencionais e claro podem simular qualquer máquina de Turing (veja <http://www.random.org/randomness/>). No caso de computabilidade difusa, ela foi primeira proposta pelo próprio mentor da lógica difusa, Lotfi Zadeh in [Zad68], quem introduziu o conceito de algoritmo e **máquinas de Turing difusas**, mas sem se preocupar com o poder computacional das mesmas. Jirí Wiedermann provou em [Wie04] que máquinas

## **8.7. Algumas Limitações da Tese de Church-Turing**

---

de Turing difusas podem reconhecer linguagens não reconhecíveis por máquinas de Turing convencionais. Bedregal e Figueira em [BF08] esclareceram alguns pontos errados no trabalho de Wiedermann e provaram a impossibilidade de se construir uma máquina de Turing difusa universal. Computadores moleculares foram introduzidos por Bennett em [Ben82] porém nesse estagio não foi incluída qualquer implementação concreta. Posteriormente, foram propostos diversos modelos para implementações moleculares de máquinas de Turing, por exemplo [Smi95, Rot96]. Computação com moléculas ou DNA tem sido alvo de intensa pesquisa nos últimos anos devido a que elas prometem melhorar substancialmente a complexidade computacional de problemas. Boneh et al. em [BDLS96] declara que **Máquinas de Turing Moleculares** (MTM) conseguem simular qualquer máquina de Turing clássica e que portanto o poder computacional das MTM no mínimo é equivalente à das máquinas de Turing clássicas. Porém em [BS04] é visto que as MTM também conseguem envolver um certo tipo de interações com o médio-ambiente e que portanto conseguem realizar tarefas que não podem ser feitas com máquinas de Turing clássicas.

## 8.8 Exercícios

1. Verificar o que faz a máquina de Turing do exemplo 8.1.10, quando as entradas são  $aba$  e  $aaabbbb$ .
2. Existe alguma entrada para a qual a máquina de Turing do exemplo 8.1.10 entra em laço infinito?
3. Que linguagem é reconhecida pela máquina de Turing

$M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_3\} \rangle$ , com

	$a$	$b$	$\square$
$q_0$	$(q_1, a, D)$	$(q_2, b, D)$	
$q_1$		$(q_1, b, D)$	$(q_3, \square, D)$
$q_2$	$(q_3, a, D)$	$(q_2, b, D)$	
$q_3$			

4. Construir máquinas de Turing que reconheçam as seguintes linguagens sobre  $\{a, b\}$ .
  - (a)  $\mathcal{L} = L(aba^*b)$
  - (b)  $\mathcal{L} = \{w / |w| \text{ é par}\}$
  - (c)  $\mathcal{L} = \{w / w = \overline{w}^R\}$ , onde  $\overline{\lambda} = \lambda$ ,  $\overline{wa} = \overline{w}b$  e  $\overline{wb} = \overline{w}a$ .
  - (d)  $\mathcal{L} = \{a^n b^m / n \geq 1, n \neq m\}$
  - (e)  $\mathcal{L} = \{a^n b^m / n \geq 1, n \leq m \leq 3n\}$
  - (f)  $\mathcal{L} = \{a^n b^m / m - n \text{ é par}\}$
  - (g)  $\mathcal{L} = \{w / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}$
  - (h)  $\mathcal{L} = \{w / \mathcal{N}_a(w) > \mathcal{N}_b(w)\}$
  - (i)  $\mathcal{L} = \{w / \mathcal{N}_a(w) \neq 2\mathcal{N}_b(w)\}$
  - (j)  $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) < 2\mathcal{N}_b(w)\}$
  - (k)  $\mathcal{L} = \{a^n b^m a^{n+m} / n \geq 0, m \geq 1\}$
  - (l)  $\mathcal{L} = \{a^n b^{n+m} a^{2m} / n \geq 0, m \geq 1\}$
  - (m)  $\mathcal{L} = \{a^k b^m a^n / 0 < k < n \text{ e } m = n + 2\}$
  - (n)  $\mathcal{L} = \{a^n b^n a^n b^n / n \geq 1\}$
  - (o)  $\mathcal{L} = \{a^n b^{2n} a^n / n \text{ é ímpar}\}$
  - (p)  $\mathcal{L} = \{a^n b^{2n} / n \geq 1\}$
  - (q)  $\mathcal{L} = \{a^n b^{n+1} a^{2n} / n \geq 1\}$
  - (r)  $\mathcal{L} = \{ww / w \in \{a, b\}^+\}$

Para cada problema, além de escrever o  $\delta$  em detalhes, teste alguns exemplos.

## 8.8. Exercícios

---

5. Construir uma máquina de Turing que aceite o complemento das seguinte linguagem:  
 $\mathcal{L} = \{a^n b^n c^n / n \geq 0\}$
6. Seja  $\Sigma = \{a, b\}$ . Construir uma máquina de Turing que compute as seguintes funções  
 $f : \Sigma^* \rightarrow \Sigma^*$
- (a)  $f(w) = w^R$
  - (b)  $f(w) = \overline{w}^R$
  - (c)  $f(w) = w\overline{w}$  onde  $\overline{\lambda} = \lambda$ ,  $\overline{wa} = \overline{w}b$  e  $\overline{wb} = \overline{w}a$
  - (d)  $f(w) = ww^R$
  - (e)  $f(w) = a^{|\mathcal{N}_a(w)|}b^{|\mathcal{N}_b(w)|}$
  - (f)  $f(\lambda) = \lambda$ ,  $f(wa) = f(w)aa$  e  $f(wb) = f(w)bb$
7. Construir uma máquina de Turing que compute a função sucessor no sistema binário.
8. Construir máquinas de Turing para computar as seguintes funções para  $x$  e  $y$  inteiros positivos representados em unário. Caso a função tenha dois argumentos eles viram separados por um zero.
- (a)  $f(x) = 2x$
  - (b)  $f(x) = \lceil \frac{x}{2} \rceil$
  - (c)  $f(x) = \lfloor \frac{x}{2} \rfloor$
  - (d)  $f(x, y) = 2x + 3y$
  - (e)  $f(x) = x \bmod 5$
  - (f)
- $$f(x) = \begin{cases} 2x + 1 & , \text{ se } x \text{ for par} \\ 0 & , \text{ caso contrário} \end{cases}$$
- (g)
- $$f(x) = \begin{cases} 0 & , \text{ se } x \text{ for par} \\ \frac{x-1}{2} & , \text{ caso contrário} \end{cases}$$
- (h)
- $$f(x, y) = \begin{cases} x - y & , \quad x > y \\ 0 & , \quad x \leq y \end{cases}$$
- (i)
- $$f(x, y) = \begin{cases} y - x & , \text{ se } y \geq x \\ 0 & , \text{ se } x > y \end{cases}$$
- (j)
- $$f(x, y) = \begin{cases} x + y & , \text{ se } x \leq y \\ x - y & , \text{ se } x > y \end{cases}$$
- (k)
- $$f(x, y) = \begin{cases} x \bmod y & , \text{ se } x > y \\ x & , \text{ se } x \leq y \end{cases}$$

(l)

$$f(x, y) = \begin{cases} x & , \text{ se } y \leq x \\ y & , \text{ senão} \end{cases}$$

(m)

$$f(x, y) = \begin{cases} \frac{x}{2} & , \text{ se } x \text{ é par} \\ \frac{x+1}{2} & , \text{ se } x \text{ é ímpar} \end{cases}$$

9. Algumas das funções acima poderia ser computada de forma mais simples caso tivéssemos usado outra representação numérica para os números?
10. Dar uma seqüência de descrições instantâneas da máquina de Turing do exemplo 8.1.13
11. Estabelecer uma convenção para representar inteiros positivos e negativos em notação unária. Com a sua convenção esboce a construção de um substrator para computar  $x - y$ .
12. Projetar uma máquina de Turing com alfabeto de entrada  $\Sigma = \{a, b\}$  que compute a função *cabeça*:  $\Sigma^+ \rightarrow \Sigma^+$ , definida por *cabeça*( $w$ ) é igual ao símbolo mais à esquerda de  $w$ . Por exemplo, *cabeça*(aab) = a.
13. Projetar uma máquina de Turing que compute a função  $f : \mathbb{N} \rightarrow \{0, 1\}$ , definida por

$$f(n) = \begin{cases} 0 & , \text{ se } n \text{ é par} \\ 1 & , \text{ se } n \text{ é ímpar} \end{cases}$$

14. Construir máquinas para computar as seguintes funções sobre os números naturais.
  - (a)  $f(x) = x + 1$
  - (b)  $f(x_1, x_2, x_3) = x_2 + 1$
  - (c)  $Z(x) = 0$
  - (d)  $C_n(x) = n$  para  $n = 3$  e depois generalize.
  - (e)  $P_i(x_1, x_2, \dots, x_i, \dots, x_n)$  para  $i = 3$  e  $n = 5$  e depois generalize.

15. Construir máquinas de Turing para computar as seguintes funções sobre o conjunto dos números naturais. Use a representação unária.
  - (a)  $f_d : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$  definida por  $f_d(x) = (x, x)$
  - (b)  $f_1 : \mathbb{N} \rightarrow \mathbb{N}$  definida por  $f_1(x) = 1$
  - (c)  $f_* : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  definida por  $f_*(x, y) = x * y$
  - (d)  $f_/_\Gamma : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  definida por  $f_/_\Gamma(x, y) = \left\lfloor \frac{x}{y} \right\rfloor$
  - (e)  $f_{mod} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  definida por  $f_{mod}(x, y) = x \bmod y$
  - (f)  $f_p : \mathbb{N} \rightarrow \mathbb{N}$  definida por

$$f_p(x) = \begin{cases} x - 1 & , \text{ se } x \geq 1 \\ 0 & , \text{ senão} \end{cases}$$

## 8.8. Exercícios

---

(g)  $f_{c1} : \mathbb{N} \rightarrow \mathbb{N}$  definida por

$$f_{c1}(x) = \begin{cases} 1 & , \quad x \leq 1 \\ x & , \quad \text{senão} \end{cases}$$

(h)  $f_{c2} : \mathbb{N} \rightarrow \mathbb{N}$  definida por

$$f_{c2}(x, y) = \begin{cases} y & , \text{ se } x = 1 \\ x & , \text{ senão} \end{cases}$$

16. Projetar, usando as máquinas de Turing do exercício anterior, diagramas de blocos para máquinas de Turing que computem as seguintes funções sobre o conjunto dos números naturais. Use a representação unária.

- (a)  $f(n) = n^5$
- (b)  $f(n) = 2^n$
- (c)  $f(n) = n!$
- (d)  $f(n) = n^n!$

17. Use um diagrama de bloco para esboçar a implementação de uma função  $f$  definida para todo  $w_1, w_2, w_3 \in \{1\}^+$  por

$$f(w_1, w_2, w_3) = i$$

onde  $i$  é tal que  $|w_i| = \max(|w_1|, |w_2|, |w_3|)$  se nenhum dos  $w$ 's tem o mesmo comprimento, e  $i = 0$  caso contrário.

18. Sejam  $g_1, \dots, g_m$  funções computáveis de  $n$ -variáveis e  $h$  uma função computável de  $m$ -variáveis. Dar argumentos para mostrar que a função de  $n$ -variáveis  $f = h \circ (g_1, \dots, g_m)$  é computável. Lembre-se que se  $x = (x_1, \dots, x_n)$ ,

$$(h \circ (g_1, \dots, g_m))(x) = h(g_1(x), \dots, g_m(x)).$$

19. Neste problema suponha que  $g$  e  $h$  são funções sobre os naturais de  $n$  e  $n + 2$  variáveis, respectivamente. Então  $g$  e  $h$  podem ser usadas para se definir uma nova função  $f$  de  $n + 1$  variáveis como segue:

$$\begin{cases} f(x, 0) = g(x) \\ f(x, y + 1) = h(x, y, f(x, y)) \end{cases}$$

Esta nova função diz-se definida de  $g$  e  $h$  por **recursão primitiva**. A variável  $y$  é dita **variável de recursão** e  $f$  diz-se obtido por recursão (primitiva) sobre  $y$ . A variável  $x$  é dita os parâmetros da recursão.

Por exemplo, a função  $f(x, y) = x + y$  é obtida por recursão de  $g(x) = x$  e  $h(x, y, z) = z + 1$ , pois

$$f(x, 0) = x + 0 = x = g(x)$$

e

$$f(x, y + 1) = x + (y + 1) = (x + y) + 1 = f(x, y) + 1 = h(x, y, f(x, y))$$

Mostre que (pode ser dado um argumento por diagramas de bloco) se  $g$  e  $h$ , definidas como acima, forem computáveis e  $f$  é definida por recursão primitiva de  $g$  e  $h$ , então  $f$  também é computável.

20. Dar uma definição formal de um autômato com pilha determinístico com duas pilhas. Dê argumentos para mostrar que a classe dos autômatos com pilha determinísticos com duas pilhas é equivalente à classe das máquinas de Turing com fitas semi-infinitas.
21. De um procedimento de enumeração para as seguintes linguagens:
  - (a)  $\{a, b\}^*$
  - (b)  $\mathcal{L} = \{a^n / n \text{ é múltiplo de } 3\}$
  - (c)  $\mathcal{L} = \{a^n b^n / n \geq 1\}$
  - (d)  $\mathcal{L} = \{w \in \{a, b\}^* / |w| = 3\}$

## **8.8. Exercícios**

---

## Capítulo 9

# Linguagens Recursivamente Enumeráveis, Recursivas e Sensíveis ao Contexto

Nosso objetivo imediato aqui será examinar a classe (na verdade as classes, como veremos mais adiante) de linguagens associadas às máquinas de Turing. Como as máquinas de Turing podem efetuar qualquer espécie de computação algorítmica, esperamos que a família das linguagens associadas com elas seja bastante ampla. Ela inclui não somente as linguagens regulares e livres do contexto, mas também várias outras (lineares, determinísticas livres do contexto, etc.) não tratadas profundamente aqui. A questão não trivial é se existe alguma linguagem para a qual, não existe uma máquina de Turing que a aceite. Responderemos essa questão mostrando, primeiro, que existem mais linguagens do que máquinas de Turing. A prova é curta e elegante, mas não construtiva, e nos fornece pouco entendimento no problema. Por isso, estabelecemos a existência de linguagens não reconhecíveis por máquinas de Turing através de exemplos explícitos, que realmente nos permita identificar tais linguagens. Uma outra via de investigação seria examinar a relação entre máquinas de Turing e certos tipos de gramáticas estabelecendo uma conexão entre essas gramáticas e aquelas que são regulares e livres do contexto. Também veremos que existe uma classe intermediária entre as linguagens reconhecidas por máquinas de Turing e aquelas reconhecidas por autômatos com pilha não determinísticos.

### 9.1 Linguagens Recursivas e Recursivamente Enumeráveis

Começaremos dando algumas terminologias para as linguagens associadas às máquinas de Turing. Faremos algumas distinções sutis entre linguagens para as quais existe uma máquina de Turing que a reconhece e linguagens para as quais existe um algoritmo membro a membro. Isto porque uma máquina de Turing não necessariamente pára para uma entrada que não aceita.

**Definição 9.1.1** *A linguagem  $\mathcal{L}$  é recursivamente enumerável se existe uma máquina de Turing,  $M$ , que reconhece  $\mathcal{L}$ , isto é,*

$$\mathcal{L} = L(M).$$

## 9.1. Linguagens Recursivas e Recursivamente Enumeráveis

---

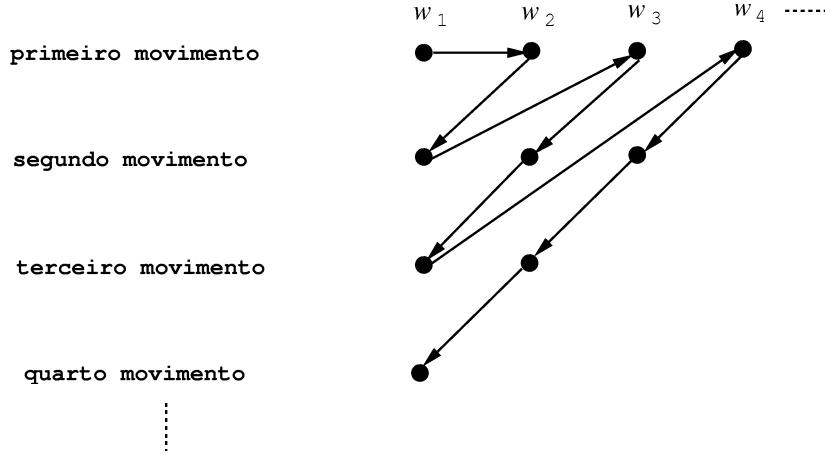


Figura 9.1: procedimento de enumeração das computações de uma máquina de Turing  $M$ .

Esta definição não diz nada com respeito ao que acontece com os  $w \notin \mathcal{L}$ , pois, neste caso  $M$  pode parar num estado não final ou ficar num laço infinito. No entanto, podemos exigir mais e requerer que a máquina diga quando ou não uma dada entrada faz parte da linguagem.

**Definição 9.1.2** Uma linguagem  $\mathcal{L}$  sobre  $\Sigma$  é **recursiva** se existe uma máquina de Turing  $M$  que reconhece  $\mathcal{L}$  e que pára para qualquer  $w \in \Sigma^+$ . Noutras palavras, se existe uma algoritmo membro a membro para  $\mathcal{L}$ .

Uma máquina de Turing que sempre pára é chamada por Sipser [Sip97] de decididor e por Kozen [Koz97] de máquina de Turing total.

Se uma linguagem é recursiva, então existe um procedimento de enumeração facilmente construível. Suponha que  $M$  é uma máquina de Turing que determina os elementos de uma linguagem recursiva  $\mathcal{L}$ . Primeiro construímos uma outra máquina de Turing, digamos  $\widehat{M}$ , que gera todas as cadeias em  $\Sigma^+$  numa ordem própria, digamos  $w_1, w_2, \dots$ . As cadeias assim geradas, se tornam as entradas de  $M$ , a qual é modificada de tal modo que ela escreve sobre sua fita somente se elas são membros de  $\mathcal{L}$ .

Por outro lado, não é fácil de ver que também existe um procedimento de enumeração para toda linguagem recursivamente enumerável. Não podemos usar o argumento acima porque se algum  $w_j \notin \mathcal{L}$ , a máquina  $M$ , quando iniciar com  $w_j$  na fita, pode nunca parar e portanto nunca obter a cadeia em  $\mathcal{L}$  que segue  $w_j$  na enumeração. Para estar seguro de que isto não acontece, a computação é efetuada de um modo diferente. Primeiro fazemos  $\widehat{M}$  gerar  $w_1$  e deixamos  $M$  executar um movimento nele. Então fazemos  $\widehat{M}$  gerar  $w_2$  e deixamos  $M$  executar um movimento sobre  $w_2$ , seguido pelo segundo movimento sobre  $w_1$ . Após isto, geramos  $w_3$  e fazemos uma etapa em  $w_3$ , a segunda etapa em  $w_2$ , a terceira em  $w_1$  e assim por diante. A ordem como isso é feito é mostrado na figura 9.1. Disto, segue que  $M$  nunca entrará num laço infinito. Como  $w \in \mathcal{L}$  é gerado por  $\widehat{M}$  e aceito por  $M$  num número finito de etapas, toda cadeia em  $\mathcal{L}$  será produzida por  $M$ .

É fácil ver que cada linguagem para a qual um procedimento de enumeração existe é recursivamente enumerável. Para ver isso compararemos a cadeia de entrada  $w$  com as cadeias seqüencialmente geradas pelo procedimento de enumeração. Se  $w \in \mathcal{L}$ , em algum momento conseguiremos o casamento e o processo parará. Dessa constatação, junto com a prova do teorema 8.6.3, podemos concluir que o conjunto de todas as máquinas de Turing é recursivamente enumerável.

As definições 9.1.1 e 9.1.2 nos dá uma pequena idéia sobre a natureza das linguagens recursivas e recursivamente enumeráveis. Estas definições dão nomes às duas famílias naturais de linguagens associadas a máquinas de Turing, porém elas não nos dizem nada da natureza da representação dessas linguagens nessas famílias. Assim, gostaríamos de ter respostas para as perguntas: “existem linguagens recursivamente enumeráveis que não são recursivas?” e “Existem linguagens, descrivíveis de alguma forma, que não são recursivamente enumeráveis?”.

## 9.2 Linguagens que Não são Recursivamente Enumeráveis

Podemos determinar a existência de linguagens que não são recursivamente enumeráveis de várias maneiras. Uma delas é sucinta e usa um resultado da matemática fundamental e elegante.

**Lema 9.2.1** *Seja  $S$  um conjunto infinito contável. Então, o conjunto das partes de  $S$ ,  $\wp(S)$ , não é contável.*

**DEMONSTRAÇÃO:** Seja  $S = \{s_1, s_2, \dots\}$  uma enumeração qualquer de  $S$ . Então, cada  $X \in \wp(S)$  (i.e.  $X \subseteq S$ ) pode ser representado por uma seqüência (infinita) de 0's e 1's, com a posição  $i$  tendo um 1 se e somente se  $s_i \in X$ . Assim, por exemplo, o conjunto  $\{s_2, s_3, s_5, s_8\}$  é representado pela seqüência 0110100100..., enquanto  $\{s_1, s_3, s_5, s_7, \dots\}$  é representado pela seqüência 10101010.... Claramente, qualquer elemento de  $\wp(S)$  pode ser representado por uma tal seqüência e qualquer seqüência representa unicamente um elemento de  $\wp(S)$ . Suponhamos que  $\wp(S)$  é contável, então, seus elementos podem ser escritos numa ordem, digamos  $X_1, X_2, X_3, \dots$ . Portanto podemos colocar estes numa tabela como a seguinte

$X_1$	$x_{11}x_{12}x_{13}x_{14}\dots$
$X_2$	$x_{21}x_{22}x_{23}x_{24}\dots$
$X_3$	$x_{31}x_{32}x_{33}x_{34}\dots$
$X_4$	$x_{41}x_{42}x_{43}x_{44}\dots$
$\vdots$	$\vdots$

onde cada  $x_{ij} \in \{0, 1\}$ . Como essa tabela contém todas as seqüências infinitas sobre  $\{0, 1\}$ , ela deveria conter a seguinte seqüência

$$\overline{X} = \overline{x}_{11}\overline{x}_{22}\overline{x}_{33}\overline{x}_{44}\dots$$

onde

## 9.2. Linguagens que Não são Recursivamente Enumeráveis

---

$$\bar{x}_{i_i} = \begin{cases} 1 & \text{se } x_{i_i} = 0 \\ 0 & \text{se } x_{i_i} = 1 \end{cases}$$

Logo,  $\bar{X} = X_k$ , para algum  $X_k$  na tabela. No entanto  $\bar{X}_{k_k} \neq X_{k_k}$ , para todo  $k$ . Portanto,  $\bar{X}$  não está na tabela. Esta contradição gera um impasse lógico que a evitaremos retirando a hipótese de que  $\wp(S)$  é contável. ■

Este tipo de argumento, pelo fato de envolver elementos diagonais na tabela, é chamado *diagonalização*. Este método, foi usado, originalmente, pelo matemático G.F. Cantor para demonstrar a não contabilidade dos números reais.

Como consequência, podemos tirar que existem menos máquinas de Turing que linguagens, e portanto deve existir alguma linguagem que não é recursivamente enumerável.

**Teorema 9.2.2** *Para qualquer  $\Sigma \neq \emptyset$ , existe uma linguagem que não é recursivamente enumerável.*

**DEMONSTRAÇÃO:** Pela definição de linguagem, dada no primeiro capítulo, uma linguagem é um subconjunto de  $\Sigma^*$ , e portanto o conjunto de todas as linguagens é exatamente  $\wp(\Sigma^*)$ , o qual pelo teorema anterior não é contável. Mas, pelo teorema 8.6.3, o conjunto de todas as máquinas de Turing é contável, portanto devem existir algumas linguagens que não são recursivamente enumerável. ■

Esta demonstração, embora breve e simples, não é satisfatória, pois é completamente não construtiva. Ela declara a existência da linguagem, mas não nos diz como ela é. A seguir veremos como obter uma tal linguagem de modo mais explícito.

**Teorema 9.2.3** *Existe uma linguagem recursivamente enumerável cujo complemento não é recursivamente enumerável.*

**DEMONSTRAÇÃO:** Seja  $\Sigma = \{a\}$ , e considere o conjunto de todas as máquinas de Turing com este alfabeto. Como esse conjunto é contável, podemos associar a ela uma ordem  $M_1, M_2, \dots$ . Cada máquina de Turing,  $M_i$ , tem uma linguagem recursivamente enumerável,  $L(M_i)$ , associada. Reciprocamente, para cada linguagem recursivamente enumerável sobre  $\Sigma$ , existe alguma máquina de Turing que a reconhece.

Agora, construiremos uma nova linguagem recursiva,  $\mathcal{L}$ , tal que para cada  $i \geq 0$ , a cadeia  $a^i$  está em  $\mathcal{L}$  se e somente se  $a^i \in L(M_i)$ . É claro que  $\mathcal{L}$  está bem definida, uma vez que  $a^i \in L(M_i)$  deve ser verdadeira ou falsa. Em seguida, consideraremos o complemento de  $\mathcal{L}$ , isto é

$$\bar{\mathcal{L}} = \{a^i / a^i \notin L(M_i)\}, \quad (9.1)$$

que também está bem definida mas, como mostraremos, não é recursivamente enumerável. Se for recursivamente enumerável, então deve existir uma máquina de Turing, digamos  $M_k$ , tal que

$$\bar{\mathcal{L}} = L(M_k). \quad (9.2)$$

Considere a cadeia  $a^k$ . Ela está em  $\mathcal{L}$  ou em  $\overline{\mathcal{L}}$ ? Suponha que  $a_k \in \overline{\mathcal{L}}$ . Por 9.2 isso implica que

$$a^k \in L(M_k).$$

Mas por 9.1 isso também implica que

$$a^k \notin \overline{\mathcal{L}}.$$

Inversamente, se assumirmos que  $a^k \in \mathcal{L}$ , então  $a^k \notin \overline{\mathcal{L}}$  e por 9.2 temos

$$a^k \notin L(M_k).$$

Mas por 9.1, obtemos que

$$a^k \in \overline{\mathcal{L}}.$$

Essa contradição não tem saída, e devemos concluir que nossa hipótese de que  $\overline{\mathcal{L}}$  é recursivamente enumerável é falsa.

Para completar a demonstração, devemos, ainda, mostrar que  $\mathcal{L}$  é recursivamente enumerável. Podemos usar para isso o procedimento de enumeração para máquinas de Turing. Dado  $a^i$ , primeiro achamos  $i$ , contando o número de  $a$ 's. Então, usamos o procedimento de enumeração para máquinas de Turing para achar  $M_i$ . Finalmente, damos sua descrição e  $a^i$  para uma máquina de Turing universal,  $M_U$ , que simula a ação de  $M$  sobre  $a^i$ . Se  $a^i \in \mathcal{L}$ , a computação efetuada por  $M_U$  parará, de modo que  $\mathcal{L}$  é recursivamente enumerável. ■

A prova desse teorema exibe explicitamente, através de 9.1, uma linguagem bem-definida que não é recursivamente enumerável. Isso não quer dizer que existe uma interpretação intuitiva fácil de  $\overline{\mathcal{L}}$ . Seria difícil exibir mais do que alguns membros triviais dessa linguagem. Entretanto  $\mathcal{L}$  foi definida propriamente.

**Teorema 9.2.4** *Se uma linguagem  $\mathcal{L}$  e seu complementar  $\overline{\mathcal{L}}$  são ambas recursivamente enumeráveis, então ambas as linguagens são recursivas.*

**DEMONSTRAÇÃO:** Se  $\mathcal{L}$  e  $\overline{\mathcal{L}}$  são ambas recursivamente enumeráveis, então existem máquinas de Turing  $M$  e  $\widehat{M}$  que servem como procedimentos de enumeração para  $\mathcal{L}$  e  $\overline{\mathcal{L}}$ , respectivamente. A primeira produzirá  $w_1, w_2, \dots$  em  $\mathcal{L}$ , e a segunda  $\widehat{w}_1, \widehat{w}_2, \dots$  em  $\overline{\mathcal{L}}$ . Suponha, agora, que é dado qualquer  $w \in \Sigma^+$ . Primeiro deixamos  $M$  gerar  $w_1$  e o comparamos com  $w$ . Se eles não coincidem deixamos  $\widehat{M}$  gerar  $\widehat{w}_1$  e o comparamos com  $w$ . Se preciso continuamos permitindo  $M$  gerar  $w_2$ , e então  $\widehat{M}$  gerar  $\widehat{w}_2$ , e assim por diante até uma delas ( $M$  ou  $\widehat{M}$ ) gerar  $w$ . Como  $L(M) \cup L(\widehat{M}) = L \cup \overline{L} = \Sigma^+$ ,  $w$  terá, mais cedo ou mais tarde, que ser gerado por alguma das máquinas. Se  $w$  foi obtida através de  $M$ , então  $w \in \mathcal{L}$ , caso contrário  $w \in \overline{\mathcal{L}}$ . O processo é um algoritmo de pertinência para  $\mathcal{L}$  e  $\overline{\mathcal{L}}$ . Portanto, ambas são recursivas. ■

Observe que se  $\mathcal{L}$  é recursiva, então  $\overline{\mathcal{L}}$  também é recursiva, e consequentemente ambas são recursivamente enumeráveis. Dos teoremas 9.2.3 e 9.2.4 podemos concluir que a família das

### 9.3. Gramáticas Irrestritas

---

linguagens recursivamente enumeráveis e a família das linguagens recursivas não são idênticas. Assim, a linguagem  $\mathcal{L}$  no teorema 9.2.3 está na primeira família mas não na segunda família. Logo, como toda linguagem recursiva é, trivialmente, recursivamente enumerável, podemos concluir que a família das linguagens recursivas está contida, propriamente, na família das linguagens recursivamente enumeráveis.

**Teorema 9.2.5** *Existe uma linguagem recursivamente enumerável que não é recursiva.*

**DEMONSTRAÇÃO:** Considere a linguagem  $\mathcal{L}$  do teorema 9.2.3. Esta linguagem é, recursivamente enumerável, mas seu complemento não o é. Logo, pelo teorema 9.2.4,  $\mathcal{L}$  não é recursiva. ■

Podemos concluir, então, que realmente existem linguagens, bem definidas, para as quais não se pode construir um algoritmo de pertinência.

**Corolário 9.2.6** *A classe das linguagens recursivas é a maior sub-classe das linguagens recursivamente enumeráveis fechada sobre complemento.*

**DEMONSTRAÇÃO:** Direto das definições de linguagens recursivas e recursivamente enumerável e dos teoremas 9.2.4 e 9.2.5. ■

## 9.3 Gramáticas Irrestritas

Para investigar a conexão entre as linguagens recursivamente enumeráveis e as gramáticas, retornaremos à definição geral de uma gramática. Na definição 1.2.11, as regras de produção poderiam ser de qualquer forma, mas várias restrições foram feitas, depois, para se obter gramáticas específicas. Se tomarmos a forma geral não impõe restrições, obteremos uma **gramática irrestrita**. Assim, numa gramática irrestrita, qualquer número de variáveis ou terminais podem ocorrer em qualquer ordem, dentro de uma produção. Existe somente uma ligeira restrição:  $\lambda$  não é permitido no lado esquerdo da produção.

**Exemplo 9.3.1** *Uma gramática irrestrita para gerar a linguagem*

$$\mathcal{L}_1 = \{a^n b^{2n} a^n / n \geq 1\}$$

*é descrita a seguir.*

$$\begin{array}{lcl} S & \longrightarrow & aAbba \\ aA & \longrightarrow & aabbA \mid a \\ bAb & \longrightarrow & bbA \\ Aa & \longrightarrow & Baa \\ bB & \longrightarrow & Bb \\ aB & \longrightarrow & aA \end{array}$$

*A cadeia aaabbbbbbaaa pode ser derivada dessa gramática como segue*

$$\begin{array}{llll}
 S & \Rightarrow aAbba & \Rightarrow aabbAbba & \Rightarrow aabbbAba \\
 & \Rightarrow aabbbaBaa & \Rightarrow aabbBbaa & \Rightarrow aabBbbbaa \\
 & \Rightarrow aaBbbbaa & \Rightarrow aaAbbbbaa & \Rightarrow aaabbAbbbbaa \\
 & \Rightarrow aaabbbAbbaa & \Rightarrow aaabbbbAbaa & \Rightarrow aaabbbbBaaa \\
 & \Rightarrow aaabbbbBbaaa & \Rightarrow aaabbbbBbbaaa & \Rightarrow aaabbBbbbaaa \\
 & \Rightarrow aaabBbbbbbbaaa & \Rightarrow aaaBbbbbbbaaa & \Rightarrow aaabbbbbaaa
 \end{array}$$

**Exemplo 9.3.2** Uma gramática irrestrita para gerar a linguagem

$$\mathcal{L}_2 = \{a^n b^n a^n b^n / n \geq 1\}$$

é descrita a seguir.

$$\begin{array}{l}
 S \longrightarrow aAbab \\
 aA \longrightarrow aabA \mid a \\
 bAb \longrightarrow bbA \\
 Aa \longrightarrow aB \\
 Ba \longrightarrow aB \\
 Bb \longrightarrow Cabb \\
 aCa \longrightarrow Caa \\
 bC \longrightarrow Cb \\
 aCb \longrightarrow aAb
 \end{array}$$

**Exemplo 9.3.3** Uma gramática irrestrita para gerar a linguagem

$$\mathcal{L}_3 = \{a^n / n \text{ é uma potência de } 2\}$$

é descrita a seguir.

$$\begin{array}{l}
 S \longrightarrow AaXA \mid aa \\
 Xa \longrightarrow aaX \\
 XA \longrightarrow YaA \\
 aY \longrightarrow Ya \\
 AY \longrightarrow AaX \mid a \\
 A \longrightarrow a
 \end{array}$$

**Exemplo 9.3.4** Uma gramática irrestrita para gerar a linguagem

$$\mathcal{L}_4 = \{a^n / n \text{ é uma potência de } 2\} \cup \{b^{3n} / n \text{ é uma potência de } 2\}$$

poderia ser feita simplesmente fazendo as duas gramáticas por separado (usando variáveis diferentes) e depois unindo elas. Porém a segunda gramática não seria tão simples de ser feita. Uma maneira mais fácil é fazer ela entrelaçada, ou seja que primeiro gere  $a^2$ , depois  $b^3$ , depois  $a^4$ , depois  $b^6$ , e assim por diante, mas usando o resultado prévio para obter o seguinte.

### 9.3. Gramáticas Irrestritas

---

$$\begin{array}{l}
 S \longrightarrow IAaaI \\
 Aaa \longrightarrow bbA \\
 AI \longrightarrow BI \mid \lambda \\
 bbbB \longrightarrow Baaaa \\
 IB \longrightarrow IA \mid \lambda \\
 I \longrightarrow \lambda
 \end{array}$$

As linguagens dos exemplos acima, são todas linguagens recursivas que não são livres do contexto. Exemplos de linguagens recursivamente enumerável que não são recursivas são difíceis de serem descritos (veja por exemplo o teorema 9.2.3) e no entanto, existem gramáticas irrestritas para tais linguagens, elas são altamente complexas. Como teremos oportunidade de ver, as gramáticas irrestritas são muito mais poderosas do que outras formas mais restritas, como gramáticas regulares e livres do contexto. Realmente, as gramáticas irrestritas correspondem à maior família de linguagens que esperamos reconhecer por meios mecânicos. Ou seja, as gramáticas irrestritas geram exatamente a família das linguagens recursivamente enumeráveis.

**Teorema 9.3.5** *Qualquer linguagem gerada por uma gramática irrestrita é recursivamente enumerável.*

**DEMONSTRAÇÃO:** A gramática define um procedimento de enumeração para toda cadeia na linguagem. Por exemplo, podemos listar toda  $w \in \mathcal{L}$  tal que

$$S \Rightarrow w,$$

isto é,  $w$  é derivado em uma etapa. Como o conjunto de produções da gramática é finito, haverá um número finito de tais cadeias. Em seguida, listamos todas  $w \in \mathcal{L}$  que podem ser derivadas em duas etapas

$$S \Rightarrow x \Rightarrow w,$$

e assim por diante. Podemos simular essas derivações numa máquina de Turing, e portanto temos um procedimento de enumeração para a linguagem. Portanto, ela é recursivamente enumerável.

■

Esta parte da correspondência entre linguagens recursivamente enumeráveis e gramáticas irrestritas não é surpreendente. A gramática gera cadeias por um processo algorítmico bem definido. Portanto as derivações podem ser dadas por uma máquina de Turing. Para mostrar a recíproca descrevemos como qualquer máquina de Turing pode ser imitada por uma gramática irrestrita. Esta parte não é fácil. A prova é longa e muito técnica, embora a idéia subjacente seja bastante simples. Por isso, daremos somente um esboço da prova.

**Teorema 9.3.6** *Para toda linguagem recursivamente enumerável,  $\mathcal{L}$ , existe uma gramática irrestrita  $G$ , tal que*

$$\mathcal{L} = L(G).$$

**DEMONSTRAÇÃO:** (Esboço) Se  $\mathcal{L}$  é recursivamente enumerável, então, por definição, existe uma máquina de Turing,  $M$ , que a reconhece. Pretendemos construir uma gramática,  $G$ , tal que  $L(G) = L(M)$ . Isto significa que queremos ter

$$S \xrightarrow{*} w_G,$$

para exatamente aqueles  $w$  para os quais

$$q_0 w \vdash^* M x q_f y.$$

Obteremos isso construindo  $G$  de tal modo que exista uma correspondência entre a seqüência de descrições instantâneas de  $M$  e as formas sentenciais de  $G$ . Como  $M$  começa com  $w$ , as derivações de  $G$  devem terminar nesta cadeia. De fato, a derivação será a reversa da computação de  $M$ . Além disso, como a derivação começa com  $S$ , devemos primeiro derivar dela uma forma sentencial que represente a configuração de parada  $x q_f y$ , da qual, por seu turno, devemos derivar  $q_0 w$ . O que queremos, então, é construir  $G$ , tal que

$$S \xrightarrow{*} x q_f y \xrightarrow{*} q_0 w$$

se e somente se  $w \in L(M)$ . Existem algumas complicações aqui. Uma delas é que, realmente, queremos  $S \xrightarrow{*} w$ , e devemos nos livrar de  $q_0$ . Para isto introduzimos um símbolo especial  $\#$ , o qual é associado com a variável de estado na descrição instantânea, de tal modo que ela possa ser eliminada na última etapa. Uma outra complicação pode provir dos possíveis brancos nas descrições instantâneas intermediárias. Esses também devem ser eliminados pela gramática. Quando colocamos todas essas condições juntas, descobrimos que a gramática  $G$  terá de ter vários conjuntos de produções:

1. Um conjunto que permite todas as derivações da forma

$$S \xrightarrow{*} \# x q_f y,$$

para todos os  $x$  e  $y$  possíveis e todo estado final  $q_f$ , com brancos possíveis à direita e à esquerda de  $\# x q_f y$ ,

2. Um conjunto que permite a derivação

$$\# x q_f y \xrightarrow{*} \# q_0 w,$$

sempre que  $q_0 w \vdash^* M x q_f y$ ,

3. Um conjunto para se livrar de  $\# q_0$  e quaisquer possíveis brancos.

■

**Exemplo 9.3.7** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  uma máquina de Turing com

- $Q = \{q_0, q_1\}$ ,

### 9.3. Gramáticas Irrestritas

---

- $\Sigma = \{0, 1\}$ ,
- $\Gamma = \{0, 1, \square\}$ ,
- $F = \{q_1\}$  e
- $\delta(q_0, 0) = (q_0, 0, D)$  e  $\delta(q_0, \square) = (q_1, \square, E)$ .

Esta máquina aceita a linguagem  $L(00^*)$ . Seguindo as sugestões poderemos constatar que as exigências serão satisfeitas pela gramática  $G = \langle V, T, S, P \rangle$ , com

- $V = \{S, A, \square, \#, q_0, q_1\}$ ,
- $T = \{0, 1\}$ .

O primeiro subconjunto das produções de  $P$  é

$$\begin{aligned} S &\longrightarrow \square S \mid S \square \mid \#A, \\ A &\longrightarrow 0A \mid 1A \mid A0 \mid A1 \mid q_1. \end{aligned}$$

é fácil ver que podemos usar essas produções para gerar qualquer cadeia  $\#xq_1y$ , com  $x, y \in \{0, 1\}^*$ , precedida e seguida por um número arbitrário de brancos. O segundo conjunto de produções é

$$\begin{aligned} 0q_0 &\longrightarrow q_00, \\ q_10\square &\longrightarrow 0q_0\square, \\ q_11\square &\longrightarrow 1q_0\square, \\ q_1\square\square &\longrightarrow \square q_0\square. \end{aligned}$$

Essas produções refletem mudanças feitas na descrição instantânea efetuadas por  $\delta$ . Por exemplo, a subcadeia  $q_00$  toma o lugar de  $0q_0$  quando a transição  $\delta(q_0, 0) = (q_0, 0, D)$  é usada. Poucos testes devem nos convencer de que essas regras tornam possível a derivação  $x \xrightarrow{*} y$ , sempre que  $y \vdash^* x$ .

Finalmente, o último conjunto

$$\begin{aligned} \#q_0 &\longrightarrow \lambda, \\ \square &\longrightarrow \lambda, \end{aligned}$$

é suficiente para se livrar de  $\#q_0$  e quaisquer brancos.

Olhemos, agora, a computação, em  $M$ , que aceita a cadeia 00.

$$q_000 \vdash 0q_00 \vdash 00q_0\square \vdash 0q_10\square.$$

Uma derivação correspondente, em  $G$ , dessa cadeia é

$$\begin{aligned} S &\Longrightarrow S\square \Longrightarrow \#A\square \Longrightarrow \#0A\square \Longrightarrow \#0A0\square \Longrightarrow \#0q_10\square \Longrightarrow \#00q_0\square \Longrightarrow \\ &\#0q_00\square \Longrightarrow \#q_000\square \Longrightarrow \#q_000 \Longrightarrow 00. \end{aligned}$$

Ignorando a marca especial  $\#$  e brancos, vemos que parte dessa derivação é, como alegamos, a seqüência de descrições instantâneas da máquina de Turing em ordem reversa.

## 9.4 Linguagens Sensíveis ao Contexto

Podem serem definidas diversas famílias de linguagens formais mais abrangentes que a família das linguagens livres do contexto, porém menos geral que a família das linguagens recursivas. Cada uma destas famílias podem ser vistas como uma classe de linguagens geradas por gramáticas com tipos de produções mais gerais que as produções das gramáticas livres do contexto, mas com algum tipo de restrição ou, do ponto de vista de autômatos, como certo tipo de máquinas de Turing que seja mais poderosa que os autômatos com pilha não determinísticos. A mais conhecida e interessante destas famílias de linguagens são as **linguagens sensíveis ao contexto**, as quais são geradas pelas gramáticas sensíveis ao contexto ou, equivalentemente, reconhecidas por autômatos limitados linearmente.

### 9.4.1 Gramáticas Sensíveis ao Contexto

No capítulo 1, demos a noção geral de gramática, e nos capítulos posteriores fomos restringindo a forma das produções na gramática até não se ter nenhuma restrição (gramáticas irrestritas). Uma outra possibilidade de se restringir produções numa gramática, de forma que elas sejam menos permissiva que as produções das gramáticas livres do contexto, é por um contexto.

**Definição 9.4.1** *Seja  $G = \langle V, T, S, P \rangle$  uma gramática irrestrita.  $G$  é uma **gramática sensível ao contexto** se toda produção  $v \rightarrow w$  em  $P$ , tem a seguinte propriedade:  $|v| \leq |w|$ , isto é, a forma sentencial  $w$  tem comprimento maior ou igual a cadeia  $v$ .*

Observe que se eliminamos  $\lambda$ -produções de uma gramática livre do contexto, a gramática resultante sempre será sensível ao contexto. Assim, podemos dizer que toda linguagem livre do contexto (que não contenha  $\lambda$ ) é sensível ao contexto. Por outro lado existem gramáticas que são sensíveis ao contexto mas que não são livres do contexto. A seguir, um exemplo de gramática sensível ao contexto que gera uma linguagem que não é livre do contexto.

**Exemplo 9.4.2** *Uma gramática sensível ao contexto  $G = \langle V, T, S, P \rangle$ , para gerar a linguagem*

$$\mathcal{L} = \{a^n b^n a^n / n \geq 1\}$$

*é descrita a seguir.*

1.  $V = \{S, A, B, C\}$
2.  $T = \{a, b\}$
3.  $P$  é o seguinte conjunto de produções:

$$\begin{array}{l} S \longrightarrow aAba \mid aba \\ aA \longrightarrow aabA \\ bAb \longrightarrow bbA \\ Aa \longrightarrow Baa \mid aa \\ bB \longrightarrow Bb \\ aB \longrightarrow aA \end{array}$$

## 9.4. Linguagens Sensíveis ao Contexto

---

Note que as gramáticas irrestritas dos exemplos 9.3.1, 9.3.2, 9.3.3 e 9.3.4 não são sensíveis ao contexto, pois elas contém uma produção com o seu lado direito menor que o seu lado esquerdo. Por exemplo, na gramática 9.3.3 tal produção é  $AY \rightarrow a$ . Porém, isso não significa que as respectivas linguagens geradas por essas gramáticas não sejam sensíveis ao contexto. De fato, as respectivas linguagens são sensíveis ao contexto.

**Exemplo 9.4.3** Uma gramática sensível ao contexto que gera a linguagem  $\mathcal{L}_1$  é a seguinte:

$$\begin{array}{l} S \longrightarrow aAba \\ aA \longrightarrow aabbA \mid ab \\ bAb \longrightarrow bbA \\ Aa \longrightarrow Baa \\ bB \longrightarrow Bb \\ aB \longrightarrow aA \end{array}$$

**Exemplo 9.4.4** Uma gramática sensível ao contexto que gera a linguagem  $\mathcal{L}_2$  é a seguinte:

$$\begin{array}{l} S \longrightarrow aAbbaabb \mid abab \\ aA \longrightarrow aabA \mid aa \\ bAb \longrightarrow bbA \\ Aa \longrightarrow aB \\ Ba \longrightarrow aB \\ Bb \longrightarrow Cabb \\ aCa \longrightarrow Caa \\ bC \longrightarrow Cb \\ aCb \longrightarrow aAb \end{array}$$

**Exemplo 9.4.5** Uma gramática sensível ao contexto que gera a linguagem  $\mathcal{L}_3$  é a seguinte:

$$\begin{array}{l} S \longrightarrow AYA \mid aa \\ Xa \longrightarrow aaX \\ XA \longrightarrow YaaA \\ aY \longrightarrow Ya \\ AY \longrightarrow AaaX \mid aa \\ A \longrightarrow aa \end{array}$$

Note que todas elas são muito similares às gramáticas irrestritas originais. O que nos poderia levar a acreditar na existência de um algoritmo que transforma gramáticas irrestritas em sensíveis ao contexto, mas isto não é possível, pois como será visto na seção 9.4.4, há gramáticas irrestritas para as quais não existem correspondentes sensíveis ao contexto. Mas mesmo com as gramáticas irrestritas que geram linguagens sensíveis ao contexto, nem sempre é possível obter, de forma direta, uma gramática sensível ao contexto equivalente a ela. Por exemplo, mesmo  $\mathcal{L}_4$  sendo sensível ao contexto não resulta imediato achar uma gramática sensível ao contexto a partir da gramática do exemplo 9.3.4.

**Exemplo 9.4.6** Uma gramática sensível ao contexto que gera a linguagem  $\mathcal{L}_4$  é a seguinte:

$$\begin{array}{l}
 S \longrightarrow aa \mid bbb \mid IAaI \\
 IA \longrightarrow FbbB \mid aa \\
 I \longrightarrow a \\
 Baa \longrightarrow bbbB \\
 BaI \longrightarrow bbb \mid bbAI \\
 F \longrightarrow b \\
 bbbA \longrightarrow Aaaaa \\
 FbA \longrightarrow IAa
 \end{array}$$

### 9.4.2 Forma Normal Sensível ao Contexto

Observe que nas gramáticas livres do contexto as produções têm a seguinte forma:  $A \longrightarrow w$ , onde  $A \in V$  e  $w \in (V \cup T)^*$ , a qual significava que uma ocorrência, numa forma sentencial, da variável  $A$  pode ser substituída pela cadeia  $w$ , independente do resto das componentes da forma sentencial. As gramáticas sensíveis ao contexto têm uma forma normal, daí seu nome, nas quais esta troca só será possível de realizar quando parte do que estiver de junto (na direita e na esquerda) da variável  $A$  na forma sentencial for uma determinada cadeia (seu contexto).

**Definição 9.4.7** Seja  $G = \langle V, T, S, P \rangle$  uma gramática sensível ao contexto.  $G$  está na **forma normal SC** se todas as produções em  $P$  têm a seguinte forma:

$$x_1Ax_2 \longrightarrow x_1wx_2,$$

onde  $A \in V$ ,  $x_1, x_2 \in (V \cup T)^*$  e  $w \in (V \cup T)^+$ , ou seja  $w$  é uma forma sentencial não vazia.

Toda gramática sensível ao contexto tem uma gramática sensível ao contexto na forma normal SC.

**Teorema 9.4.8** Se  $G = \langle V, T, S, P \rangle$  é uma gramática sensível ao contexto, então existe uma gramática sensível ao contexto  $\widehat{G} = \langle \widehat{V}, T, S, \widehat{P} \rangle$  na forma normal SC tal que  $L(G) = L(\widehat{G}_1)$ .

**DEMONSTRAÇÃO:** A construção de  $\widehat{G}_1$  será dada em duas etapas.

**Etapa 1:** Esta etapa é similar à etapa 1 da forma normal de Chomsky. Construir a partir de  $G$  uma gramática,  $G_1 = \langle V_1, T, S, P_1 \rangle$ , introduzindo, para cada  $a \in T$ , uma nova variável  $B_a$  e a produção  $B_a \longrightarrow a$ . Para cada produção, cujo lado direito tiver um comprimento maior do que um, substituir, tanto no lado esquerdo quanto direito, cada ocorrência de um símbolo terminal, digamos  $a$ , por sua respectiva variável associada (no caso,  $B_a$ ). Faça o mesmo com as eventuais produções cujo lado esquerdo for um símbolo terminal. No fim dessa etapa, teremos uma gramática  $G_1$ , cujas produções têm a forma

$$X \longrightarrow Y, \tag{9.3}$$

com  $X \in V$  e  $Y \in V \cup T$ , ou a forma

## 9.4. Linguagens Sensíveis ao Contexto

---

$$B_1 \cdots B_m \longrightarrow C_1 \cdots C_n, \quad (9.4)$$

com  $m \leq n$ , e  $B_i, C_j \in V_1$ , para cada  $i = 1, \dots, m$  e  $j = 1, \dots, n$ . É fácil ver que  $L(G_1) = L(G)$  e  $G_1$  continua sendo sensível ao contexto.

**Etapa 2:** Construir a partir de  $G$  uma nova gramática,  $\hat{G}$ , substituindo cada produção  $B_1 \cdots B_m \longrightarrow C_1 \cdots C_n$  em  $P_1$ , com  $m \leq n$  que não esteja na forma normal SC, pode ser substituída pelo seguinte conjunto de produções:

$$\begin{aligned} B_1 \dots B_m &\longrightarrow Z_1 B_2 \dots B_m \\ Z_1 B_2 \dots B_m &\longrightarrow Z_1 Z_2 B_3 \dots B_m \\ &\vdots \\ Z_1 \dots Z_{m-1} B_m &\longrightarrow Z_1 \dots Z_{m-1} C_m \dots C_n \\ Z_1 \dots Z_{m-1} C_m \dots C_n &\longrightarrow C_1 Z_2 \dots Z_{m-1} C_m \dots C_n \\ C_1 Z_2 \dots Z_{m-1} C_m \dots C_n &\longrightarrow C_1 C_2 Z_3 \dots Z_{m-1} C_m \dots C_n \\ &\vdots \\ C_1 \dots C_{m-2} Z_{m-1} C_m \dots C_n &\longrightarrow C_1 \dots C_n \end{aligned}$$

onde  $Z_1, \dots, Z_{m-1}$  são variáveis novas. Assim, claramente, estas novas produções satisfazem a forma normal SC. Logo, a gramática  $\hat{G}$ , resultante, está na forma normal SC e, claramente,  $L(\hat{G}) = L(G_1) = L(G)$ . ■

Observe, que uma gramática sensível ao contexto, obtida usando o algoritmo acima, tem como propriedade extra que o lado esquerdo das produções são sempre cadeias de variáveis.

**Exemplo 9.4.9** Seja a gramática sensível ao contexto do exemplo 9.4.2.

**Etapa 1:** Construção de  $G_1$

$$1. \quad V_1 = \{S, A, B, B_a, B_b\}$$

$$2. \quad T = \{a, b\}$$

3.  $P$  é o seguinte conjunto de produções:

$$\begin{aligned}
 S &\longrightarrow B_a A B_b B_a \mid B_a B_b B_a \\
 B_a A &\longrightarrow B_a B_a B_b A \\
 B_b A B_b &\longrightarrow B_b B_b A \\
 A B_a &\longrightarrow B B_a B_a \mid B_a B_a \\
 B_b B &\longrightarrow C B_b B_a \\
 B_b B &\longrightarrow B B_b \\
 B_a B &\longrightarrow B_a A \\
 B_a &\longrightarrow a \\
 B_b &\longrightarrow b
 \end{aligned}$$

**Etapa 2:** *Construção de  $\widehat{G}$*

1.  $T = \{a, b\}$
2.  $P$  é constituído pelos seguintes conjuntos de produções:

**Produções que já está na forma normal SC:**

$$\begin{aligned}
 S &\longrightarrow B_a A B_b B_a \mid B_a B_b B_a \\
 B_a &\longrightarrow a \\
 B_b &\longrightarrow b
 \end{aligned}$$

**Transformando a produção:**  $B_a A \longrightarrow B_a B_a B_b A$

$$\begin{aligned}
 B_a A &\longrightarrow Z_1 A \\
 Z_1 A &\longrightarrow Z_1 B_a B_b A \\
 Z_1 B_a B_b A &\longrightarrow B_a B_a B_b A
 \end{aligned}$$

**Transformando a produção:**  $B_b A B_b \longrightarrow B_b B_b A$

$$\begin{aligned}
 B_b A B_b &\longrightarrow Z_2 A B_b \\
 Z_2 A B_b &\longrightarrow Z_2 Z_3 B_b \\
 Z_2 Z_3 B_b &\longrightarrow Z_2 Z_3 A \\
 Z_2 Z_3 A &\longrightarrow B_b Z_3 A \\
 B_b Z_3 A &\longrightarrow B_b B_b A
 \end{aligned}$$

**Transformando a produção:**  $A B_a \longrightarrow B B_a B_a$

$$\begin{aligned}
 A B_a &\longrightarrow Z_3 B_a \\
 Z_3 B_a &\longrightarrow Z_3 B_a B_a \\
 Z_3 B_a B_a &\longrightarrow B B_a B_a
 \end{aligned}$$

**Transformando a produção:**  $A B_a \longrightarrow B_a B_a$

$$\begin{aligned}
 A B_a &\longrightarrow Z_4 B_a \\
 Z_4 B_a &\longrightarrow Z_4 B_a \\
 Z_4 B_a &\longrightarrow B_a B_a
 \end{aligned}$$

## 9.4. Linguagens Sensíveis ao Contexto

---

**Transformando a produção:**  $B_bB \longrightarrow BB_b$

$$B_bB \longrightarrow Z_5B$$

$$Z_5B \longrightarrow Z_5B_b$$

$$Z_5B_b \longrightarrow BB_b$$

**Transformando a produção:**  $B_bB \longrightarrow B_aA$

$$B_bB \longrightarrow Z_6B$$

$$Z_6B \longrightarrow Z_6A$$

$$Z_6A \longrightarrow B_aA$$

3. Logo,  $\widehat{V} = \{S, A, B, B_a, B_b, Z_1, \dots, Z_5\}$ .

**Exemplo 9.4.10** Seja a gramática sensível ao contexto do exemplo 9.4.6.

**Etapa 1:** Construção de  $G_1$

$$1. V_1 = \{S, A, B, C, B_a, B_b\}$$

$$2. T = \{a, b\}$$

3.  $P$  é o seguinte conjunto de produções:

$$\begin{aligned} S &\longrightarrow B_aB_a \mid B_bB_bB_b \mid IAB_aI \\ IA &\longrightarrow FB_bB_bB \mid B_aB_a \\ I &\longrightarrow a \\ BB_aB_a &\longrightarrow B_bB_bB_bB \\ BB_aI &\longrightarrow B_bB_bB_b \mid B_bB_bAI \\ F &\longrightarrow b \\ B_bB_bB_bA &\longrightarrow AB_aB_aB_aB_a \\ FB_bA &\longrightarrow IAB_a \\ B_a &\longrightarrow a \\ B_b &\longrightarrow b \end{aligned}$$

**Etapa 2:** Construção de  $\widehat{G}$

$$1. T = \{a, b\}$$

2.  $P$  é constituído pelos seguintes conjuntos de produções:

**Produções que já está na forma normal SC**

$$S \longrightarrow B_aB_a \mid B_bB_bB_b \mid IAB_aI$$

$$I \longrightarrow a$$

$$F \longrightarrow b$$

$$B_a \longrightarrow a$$

$$B_b \longrightarrow b$$

**Transformando a produção:**  $IA \longrightarrow FB_bB_bB$

$$\begin{array}{l} IA \longrightarrow Z_1A \\ Z_1A \longrightarrow Z_1B_bB_bB \\ Z_1B_bB_bB \longrightarrow FB_bB_bB \end{array}$$

**Transformando a produção:**  $IA \longrightarrow B_aB_a$

$$\begin{array}{l} IA \longrightarrow Z_2A \\ Z_2A \longrightarrow Z_2B_a \\ Z_2B_a \longrightarrow B_aB_a \end{array}$$

**Transformando a produção:**  $BB_aB_a \longrightarrow B_bB_bB_bB$

$$\begin{array}{l} BB_aB_a \longrightarrow Z_3B_aB_a \\ Z_3B_aB_a \longrightarrow Z_3Z_4B_a \\ Z_3Z_4B_a \longrightarrow Z_3Z_4B_bB \\ Z_3Z_4B_bB \longrightarrow B_bZ_4B_bB \\ B_bZ_4B_bB \longrightarrow B_bB_bB_bB \end{array}$$

**Transformando a produção:**  $BB_aI \longrightarrow B_bB_bB_b$

$$\begin{array}{l} BB_aI \longrightarrow Z_5B_aI \\ Z_5B_aI \longrightarrow Z_5Z_6I \\ Z_5Z_6I \longrightarrow Z_5Z_6B_b \\ Z_5Z_6B_b \longrightarrow B_bB_bB_b \end{array}$$

**Transformando a produção:**  $BB_aI \longrightarrow B_bB_bAI$

$$\begin{array}{l} BB_aI \longrightarrow Z_6B_aI \\ Z_6B_aI \longrightarrow Z_6Z_7I \\ Z_6Z_7I \longrightarrow Z_6Z_7AI \\ Z_6Z_7AI \longrightarrow B_bZ_7AI \\ B_bZ_7AI \longrightarrow B_bB_bAI \end{array}$$

**Transformando a produção:**  $B_bB_bB_bA \longrightarrow AB_aB_aB_aB_a$

$$\begin{array}{l} B_bB_bB_bA \longrightarrow Z_8B_bB_bA \\ Z_8B_bB_bA \longrightarrow Z_8Z_9B_bA \\ Z_8Z_9B_bA \longrightarrow Z_8Z_9Z_{10}A \\ Z_8Z_9Z_{10}A \longrightarrow Z_8Z_9Z_{10}B_aB_a \\ Z_8Z_9Z_{10}B_aB_a \longrightarrow AZ_9Z_{10}B_aB_a \\ AZ_9Z_{10}B_aB_a \longrightarrow AB_aZ_{10}B_aB_a \\ AB_aZ_{10}B_aB_a \longrightarrow AB_aB_aB_aB_a \end{array}$$

**Transformando a produção:**  $FB_bA \longrightarrow IAB_a$

## 9.4. Linguagens Sensíveis ao Contexto

---

$$\begin{aligned}
 FB_bA &\longrightarrow Z_{11}B_bA \\
 Z_{11}B_bA &\longrightarrow Z_{11}Z_{12}A \\
 Z_{11}Z_{12}A &\longrightarrow Z_{11}Z_{12}B_a \\
 Z_{11}Z_{12}B_a &\longrightarrow IZ_{12}B_a \\
 IZ_{12}B_a &\longrightarrow IAB_a
 \end{aligned}$$

3. Logo,  $\widehat{V} = \{S, A, B, I, F, B_a, B_b, Z_1, \dots, Z_{12}\}$ .

### 9.4.3 Autômatos Limitados Linearmente

Um autômato limitado linearmente é, simplesmente, uma máquina de Turing não-determinística a qual no lugar de ter uma fita potencialmente infinita sob a qual computa, ela é restrita à porção da fita que contém a entrada mais dois quadrados extras, um em cada extremo, contendo marcadores de início e fim de fita, respectivamente. Assim, um **autômato limitado linearmente (ALL)** é uma máquina de Turing satisfazendo as seguintes condições:

1. O alfabeto de entrada contém dois símbolos especiais \* e \$, usados como marcadores do início e fim da fita, respectivamente.
2. Não existem movimentos à esquerda de \* nem à direita de \$
3. Nada pode ser escrito sobre estes símbolos.

**Definição 9.4.11** Um autômato limitado linearmente (ALL), é uma oito-tupla  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, *, \$, F \rangle$ , tal que

- $\langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  é uma máquina de Turing não-determinística (definição 8.5.1),
- \* e \$ são símbolos especiais de  $\Sigma$ , chamados **marcador do início da fita** e **marcador do final da fita**, respectivamente, e
- A função de transição  $\delta : Q \times \Sigma \longrightarrow \wp(Q \times \Sigma \times \{D, E\})$  tem as seguintes restrições para todo  $q \in Q$  e  $a \in \Sigma - \{*, \$\}$ :
  - $\delta(q, *) \subseteq Q \times \{*\} \times \{D\}$
  - $\delta(q, \$) \subseteq Q \times \{\$\} \times \{E\}$
  - $\delta(q, a) \subseteq Q \times (\Sigma - \{*\$}) \times \{D, E\}$

Como todo autômato, um ALL reconhece uma linguagem, mas neste caso a linguagem reconhecida será um subconjunto de  $\Sigma - \{*, \$\}$  em vez de um subconjunto de  $\Sigma$ , pois os símbolos \* e \$ são meros marcadores. Assim, a linguagem reconhecida por um ALL  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, *, \$, F \rangle$  é o conjunto

$$L(M) = \{w \in (\Sigma - \{*, \$\})^* / q_0 * w \$ \vdash^* u q_f v \text{ para algum } q_f \in F\}$$

**Exemplo 9.4.12** Um ALL que reconhece a linguagem  $\mathcal{L} = \{a^n b^n a^n / n \geq 1\}$ , é a oito-tupla  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, *, \$, F \rangle$ , tal que

- $Q = \{q_0, \dots, q_8\}$
- $\Sigma = \{a, b, *, \$\}$
- $\Gamma = \{a, b, \$, \square, x, y, z\}$
- $F = \{q_8\}$
- $\delta$  é definido através da seguinte tabela:

	$a$	$b$	*	$\$$	$x$	$y$	$z$	$\square$
$q_0$								
$q_1$	$(q_2, x, D)$	$(q_5, x, D)$	$(q_1, *, D)$					
$q_2$	$(q_2, a, D)$	$(q_2, b, D)$		$(q_3, \$, E)$	$(q_1, x, D)$	$(q_3, y, E)$	$(q_8, z, E)$	
$q_3$	$(q_4, y, E)$							
$q_4$	$(q_4, a, E)$	$(q_4, b, E)$			$(q_1, x, D)$			
$q_5$		$(q_5, b, D)$		$(q_6, \$, E)$		$(q_5, y, D)$	$(q_6, z, E)$	
$q_6$						$(q_7, z, E)$		
$q_7$		$(q_7, b, E)$			$(q_1, x, D)$	$(q_7, y, E)$		
$q_8$								

Observe que os marcadores estão no início da fita de entrada, mas não são considerados como partes das palavras aceitas ou rejeitadas pelo autômato. Por outro lado, ALL's não podem acessar células à esquerda de \* nem à direita de \$, tornando inúteis os símbolos em brancos à esquerda e à direita do conteúdo inicial da fita de entrada, usuais em máquinas de Turing padrão. Por esse motivo,  $\square$  não foi considerado explicitamente como componente da oito-tupla na definição de ALL (embora implicitamente seja considerado ao se requerer que uma certa sete-tupla seja uma máquina de Turing) e, portanto, não temos necessidade de supor que existe uma fita em branco à direita de \$ nem à esquerda de \*. Logo, podemos considerar que a fita de entrada começa em \* e termina em \$ e eliminar completamente da definição de ALL a ocorrência do símbolo branco ( $\square$ ).

**Exemplo 9.4.13** Um ALL que reconhece a linguagem  $\mathcal{L}_1 = \{a^m b^n a^n b^m / m, n \geq 1\}$  é descrito na seguinte tabela:

#### 9.4. Linguagens Sensíveis ao Contexto

---

	<i>a</i>	<i>b</i>	*	\$	<i>x</i>
$q_0$			$(q_1, *, D)$		
$q_1$	$(q_2, x, D)$	$(q_5, x, D)$			$(q_1, x, D)$
$q_2$	$(q_2, a, D)$	$(q_3, x, D)$		$(q_3, \$, E)$	$(q_3, x, E)$
$q_3$		$(q_4, x, E)$			
$q_4$	$(q_4, a, E)$	$(q_4, b, E)$			$(q_1, x, D)$
$q_5$	$(q_5, a, D)$	$(q_5, b, D)$			$(q_6, x, E)$
$q_6$	$(q_7, x, E)$				
$q_7$	$(q_7, a, E)$	$(q_7, b, E)$			$(q_8, x, D)$
$q_8$		$(q_5, x, D)$			$(q_9, x, D)$
$q_9$					

O estado final deste ALL é o estado  $q_9$ .

**Proposição 9.4.14** Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, *, \$, F \rangle$  um ALL. Então existe um ALL  $M' = \langle Q', \Sigma, \Gamma', F' \rangle$  tal que  $L(M) = L(M')$  e

$$w \in L(M') \text{ se, e somente se, } q_0 * w \$ \vdash^*_{M'} q_f * w \$$$

**DEMONSTRAÇÃO:** (Argumento) Como a fita é limitada ao tamanho da entrada, nunca adicionaremos símbolos extras à fita, o máximo que pode acontecer é substituir os símbolos da entrada por outros. Se ao fizermos isto, tivermos o cuidado de substituir cada símbolo  $a \in \Sigma - \{*, \$\}$  por um símbolo especial associado a ele, então ao final da computação, poderemos re-escrever a entrada original na fita e nos posicionar ao início da mesma. ■

Na proposição anterior, podemos pensar  $M'$  como uma forma normal de  $M$ .

**Exemplo 9.4.15** O ALL do exemplo anterior, pode ser re-escrito nesta forma normal, da seguinte maneira:

	<i>a</i>	<i>b</i>	*	\$	<i>x</i>	<i>y</i>
$q_0$			$(q_1, *, D)$			
$q_1$	$(q_2, x, D)$	$(q_5, y, D)$			$(q_1, x, D)$	
$q_2$	$(q_2, a, D)$	$(q_2, b, D)$		$(q_3, \$, E)$		$(q_3, y, E)$
$q_3$		$(q_4, y, E)$				
$q_4$	$(q_4, a, E)$	$(q_4, b, E)$			$(q_1, x, D)$	
$q_5$	$(q_5, a, D)$	$(q_5, b, D)$			$(q_6, x, E)$	$(q_6, y, E)$
$q_6$	$(q_7, x, E)$					
$q_7$	$(q_7, a, E)$	$(q_7, b, E)$				$(q_8, y, D)$
$q_8$		$(q_5, y, D)$			$(q_9, x, D)$	
$q_9$				$(q_{10}, \$, E)$	$(q_9, x, D)$	$(q_9, y, D)$
$q_{10}$			$(q_{11}, *, D)$		$(q_{10}, a, E)$	
$q_{11}$	$(q_{12}, a, E)$	$(q_{12}, b, E)$				$(q_{10}, b, E)$
$q_{12}$						

O estado final deste ALL é o estado  $q_{12}$ .

#### **9.4.4 Relação entre Linguagens Recursivas e Linguagens Sensíveis ao Contexto**

Como toda gramática sensível ao contexto é uma gramática irrestrita, então toda linguagem sensível ao contexto é uma linguagem recursiva enumerável. Menos evidente é mostrar que toda linguagem sensível ao contexto é recursiva, pois ALL's podem não parar para toda entrada.

**Teorema 9.4.16** *Toda linguagem sensível ao contexto é recursiva.*

**DEMONSTRAÇÃO:** (Esboço) Seja  $\mathcal{L}$  uma linguagem sensível ao contexto e  $G$  uma gramática sensível ao contexto, tal que  $L(G) = \mathcal{L}$ . Analogamente ao teorema 9.3.5, podemos simular as derivações de  $G$  numa máquina de Turing, definindo um procedimento de enumeração para toda cadeia na linguagem. Por exemplo, podemos listar toda  $w \in \mathcal{L}$  tal que

$$S \Rightarrow w,$$

isto é,  $w$  é derivado em uma etapa. Como o conjunto de produções da gramática é finito, haverá um número finito de tais cadeias. Em seguida, listamos todas  $w \in \mathcal{L}$  que podem ser derivadas em duas etapas

$$S \Rightarrow x \Rightarrow w,$$

e assim por diante. Como as gramáticas sensíveis ao contexto têm a propriedade que a cada passo o tamanho da cadeia gerada aumenta, então para qualquer  $u \in T^*$ , este processo sempre pára, ou gerando  $u$ , neste caso  $u \in \mathcal{L}$ , ou gerando uma forma sentencial maior que  $|u|$ , o qual significa que a gramática nunca gerará  $u$  e, portanto,  $u \notin \mathcal{L}$ . Logo, como, temos uma máquina de Turing que sempre pára e reconhece  $\mathcal{L}$ , então  $\mathcal{L}$  é recursiva. ■

Por outro lado, nem toda linguagem recursiva é sensível ao contexto, para demonstrar isto precisaríamos descrever uma linguagem e mostrar que ela é, efetivamente, recursiva mas não é sensível ao contexto. Linguagens recursivas que não sejam sensíveis ao contexto, e que sejam intuitivas são difíceis de se encontrarem, embora existam uma infinidade de tais linguagens. Uma demonstração elegante da existência de uma tal linguagem, pode ser encontrada em [Sal73, HU79, DW83]. No entanto, nestas demonstrações se constrói tal linguagem recursiva a partir de uma enumeração de um conjunto de máquinas de Turing que sempre param. Um exemplo mais tangível de uma tal linguagem, pode ser obtida via uma codificação do gráfico da função de Ackermann  $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ , definida por

$$A(0, y) = y + 1$$

$$A(x + 1, 0) = A(x, 1)$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y)).$$

## 9.4. Linguagens Sensíveis ao Contexto

---

Esta função de dupla recursão, foi introduzida por Wilhelm Ackermann no intuito de demonstrar, construtivamente, a existência de uma função recursiva<sup>1</sup> que não é primitiva recursiva<sup>2</sup>, isto é, uma função que seja computada por uma máquina de Turing que sempre pára, mas que não é computada por uma máquina limitada linearmente. Existem diversas versões da função de Ackermann, a dada aqui é a mais usual e pode ser encontrada em [BL74, ?] entre outros, outras definições podem ser encontradas em [HU79, Bir96].

Uma tal codificação poderia ser a linguagem  $L_{Ack} \subseteq \{0, 1\}^*$  descrita por

$$L_{Ack} = \{w_1 0 w_2 0 w_3 / , w_1, w_2, w_3 \in \{1\}^* \text{ e } A(|w_1|, |w_2|) = |w_3|\}, \quad (9.5)$$

onde  $|w|$  é o tamanho da cadeia  $w$ .

### 9.4.5 Equivalência entre ALL's e Gramáticas Sensíveis ao Contexto

Mostraremos que a não ser pelo fato que um ALL pode aceitar a palavra vazia,  $\lambda$ , enquanto uma gramática sensível ao contexto não, ambos (ALL's e gramáticas sensíveis ao contexto) aceitam, exatamente, a mesma classe de linguagens, ou seja as linguagens sensíveis ao contexto.

**Teorema 9.4.17** *Se  $\mathcal{L}$  é uma linguagem sensível ao contexto, então existe um ALL  $M$  que aceita  $\mathcal{L}$ , isto é  $L(M) = \mathcal{L}$ .*

DEMONSTRAÇÃO: (Veja [HU79, Teorema 9.5]). ■

Observe que, linguagens sensíveis ao contexto não consideram  $\lambda$ , pois para uma gramática gerar a cadeia vazia  $\lambda$ , precisaria da produção  $S \rightarrow \lambda$  a qual fere a propriedade que o lado esquerdo seja menor que lado direito de uma produção, e por tanto não seria sensível ao contexto. Por outro lado linguagens aceitas por ALL's não têm esta restrição. Isto não é um grande problema, pois se quiséssemos considerar linguagens com a cadeia vazia, bastaria flexibilizar a definição de gramática sensível ao contexto, impondo que toda produção  $w \rightarrow v$ , satisfaça  $|w| \leq |v|$  ou que  $w = S$  e  $v = \lambda$ .

**Teorema 9.4.18** *Seja  $M$  um ALL, então  $L(M) - \{\lambda\}$  é uma linguagem sensível ao contexto.*

DEMONSTRAÇÃO: (Veja [HU79, Teorema 9.6]). ■

---

<sup>1</sup> As funções recursivas são a sub-classe das funções parciais recursivas, ou analogamente as funções Turing-computáveis, que são totais, ou noutras palavras para as quais existe uma máquina de Turing que a computa e que sempre pára [BL74, Cut80, Smi94, CO98].

<sup>2</sup> Uma função sobre uma representação dos números naturais, por exemplo a unária, é primitiva recursiva se pode ser construída a partir de funções básicas ( $z(\lambda) = 0$ ,  $s(w) = w1$ ,  $\pi(w) = \lambda$  e  $i(w) = w$ ), e operações entre funções ( $f \circ g(x) = f(g(x))$ ,  $f \times g(x, y) = (f(x), g(y))$  e  $f^\#(x, y) = \underbrace{f \circ \dots \circ f}_{y-\text{vezes}}(x)$ ).

#### 9.4.6 Propriedades de Fecho das Linguagens Sensíveis ao Contexto

As linguagens sensíveis ao contexto são fechadas sobre a maioria das operações usuais sobre linguagens: união, intersecção, concatenação, fecho positivo e imagem homomorfa.

**Proposição 9.4.19** *Sejam  $\mathcal{L}_1, \mathcal{L}_2$  linguagens sensíveis ao contexto. Então as linguagens:*

1.  $\mathcal{L}_1 \cup \mathcal{L}_2$ ,
2.  $\mathcal{L}_1 \cap \mathcal{L}_2$ ,
3.  $\mathcal{L}_1 \circ \mathcal{L}_2$ ,
4.  $\mathcal{L}_1^+ = \bigcup_{i=1}^{\infty} \mathcal{L}_1^i$
5.  $h(\mathcal{L}_1)$ , para qualquer homomorfismo  $h : T_1 \longrightarrow T_2^*$

*são sensíveis ao contexto.*

**DEMONSTRAÇÃO:** Sejam  $G_1 = \langle V_1, T_1, S_1, P_1 \rangle$  e  $G_2 = \langle V_2, T_2, S_2, P_2 \rangle$  gramáticas sensíveis ao contexto tais que  $\mathcal{L}_1 = L(G_1)$  e  $\mathcal{L}_2 = L(G_2)$  e  $V_1 \cap V_2 = \emptyset$ .

1. Análogo ao caso das linguagens regulares e livres do contexto, a união das linguagens  $\mathcal{L}_1$  e  $\mathcal{L}_2$  pode ser obtida simplesmente juntando as duas gramáticas e adicionando uma nova variável de início  $S$  e as produções:

$$S \longrightarrow S_1 \mid S_2.$$

Assim, a nova gramática

$$G_1 \cup G_2 = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \longrightarrow S_1, S \longrightarrow S_2\} \rangle,$$

claramente, é sensível ao contexto e gera a linguagem:

$$L(G_1 \cup G_2) = \mathcal{L}_1 \cup \mathcal{L}_2.$$

2. Pelo teorema 9.4.17 e proposição 9.4.14, é possível construir ALL's  $M_1 = \langle Q_1, \Sigma_1, \Gamma_1, \delta_1, q_0^1, *, \$, F_1 \rangle$  e  $M_2 = \langle Q_2, \Sigma_2, \Gamma_2, \delta_2, q_0^2, *, \$, F_2 \rangle$ , tais que  $L(M_1) = L(G_1)$ ,  $L(M_2) = L(G_2)$ , para cada  $w \in L(M_1)$  se, e somente se,  $q_0 * w\$ \vdash^* M_1 q_f * w\$$ , para algum  $q_f \in F_1$  e, analogamente, para cada  $w \in L(M_2)$  se, e somente se,  $q_0 * w\$ \vdash^* M_2 q_f * w\$$ , para algum  $q_f \in F_2$ . Sem perda alguma, podemos pensar que  $F_1 = \{q_0^2\}$  e  $Q_1 \cap Q_2 = F_1$ . A partir, dessas duas máquinas, podemos construir uma terceira  $M_3 = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2, \delta_3, q_0^1, *, \$, F_2 \rangle$ , onde  $\delta_3 : (Q_1 \cup Q_2) \times (\Gamma_1 \cup \Gamma_2) \longrightarrow (Q_1 \cup Q_2) \times (\Gamma_1 \cup \Gamma_2) \times \{D, E\}$  é definida por

$$\delta_3(q, a) = \begin{cases} \delta_1(q, a) & , \text{ se } q \in Q_1 - F_1 \\ \delta_2(q, a) & , \text{ se } q \in Q_2 \end{cases}$$

## 9.4. Linguagens Sensíveis ao Contexto

---

Claramente  $M_3$  aceitará um  $w$  se, e somente se,  $w$  é aceito por  $M_1$  e por  $M_2$ , assim,

$$\begin{aligned} L(M_3) &= L(M_1') \cap L(M_2') \\ &= L(M_1) \cap L(M_2) \\ &= L(G_1) \cap L(G_2) \\ &= \mathcal{L}_1 \cap \mathcal{L}_2 \end{aligned}$$

3. Sem perda de generalidade podemos pensar que  $G_1$  e  $G_2$  estão na forma normal SC. Construa a gramática sensível ao contexto  $G_3$  como

$$G_3 = \langle V_1 \cup V_2, T_1 \cup T_2, S_1, \widehat{P}_1 \cup P_2 \rangle,$$

onde  $\widehat{P}_1 = \{x \rightarrow y \in P_1 / y \notin T_1^*\} \cup \{x \rightarrow yS_2 / x \rightarrow y \in P_1 \text{ e } y \in T_1^*\}$

Claramente,  $G_3$  é uma gramática sensível ao contexto na forma normal SC tal que  $L(G_3) = L(G_1) \circ L(G_2)$ . Observe, que as restrições colocadas por [HU79] para construir esta gramática não acontecem aqui, pois nos assumimos que as gramáticas  $G_1$  e  $G_2$  estão na forma normal SC via o algoritmo apresentado na demonstração do teorema 9.4.8, no qual sempre o lado esquerdo de uma produção é constituída só de variáveis.

4. Seja gramática  $G = \langle V_1 \cup \{S\}, T_1, S, P_1 \cup \{S \rightarrow S_1S \mid S_1\} \rangle$ , para algum  $S \notin V_1$ . Trivialmente,  $G$  é uma gramática sensível ao contexto tal que  $L(G) = L(G_1)^+$
5. Seja a gramática  $G = \langle V_1, T_2, S_1, P \rangle$ , onde  $P$  é obtido substituindo em cada produção  $w \rightarrow v$  os símbolos terminais por sua imagem homomorfa, isto é, se o símbolo terminal  $a$  ocorre em  $w$  ou em  $v$ , então substituímos cada ocorrência de  $a$ , por  $h(a)$ . Claramente,  $G$  continua sendo sensível ao contexto e  $L(G) = h(L(G_1))$ . ■

## 9.5 Exercícios

1. Dê uma gramática irrestrita que gere as linguagens
  - (a)  $\mathcal{L} = \{ww \mid w \in \{a, b\}^*\}$
  - (b)  $\mathcal{L} = \{www \mid w \in \{a, b\}^*\}$
  - (c)  $\mathcal{L} = \{w\bar{w} \mid w \in \{a, b\}^*\}$ , onde  $\bar{\lambda} = \lambda$ ,  $\bar{wa} = \bar{W}b$  e  $\bar{wb} = \bar{w}a$ .
  - (d)  $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$
  - (e)  $\mathcal{L} = \{a^n b^n c^n \mid n \text{ é ímpar}\}$
  - (f)  $\mathcal{L} = \{w \in \{a, b, c\}^* \mid N_a(w) = N_b(w) = N_c(w)\}$
  - (g)  $\mathcal{L} = \{w \in \{a, b, c\}^* \mid 1 \leq N_a(w) \leq N_b(w) \leq N_c(w)\}$
  - (h)  $\mathcal{L} = \{w \in \{a, b\}^* \mid N_a(w) \neq 2N_b(w)\}$
  - (i)  $\mathcal{L} = \{a^m b^n a^{m-n} \mid n, m \geq 0 \text{ e } m > n\}$
  - (j)  $\mathcal{L} = \{a^{n+2} b^{3n} a^n \mid n \geq 1\}$
  - (k)  $\mathcal{L} = \{a^{2n} b^n a^{3n+2} \mid n \geq 1\}$
  - (l)  $\mathcal{L} = \{a^n b^m c^{n-m} \mid n \geq 0\}$
  - (m)  $\mathcal{L} = \{a^{n+2} b^{m+1} a^{2n} \mid n \geq m \geq 1\}$
  - (n)  $\mathcal{L} = \{a^m b^n a^{2m} b^{2n} \mid n \geq m \geq 1\}$
2. Nas gramáticas irrestritas que voce fez para o exercício anterior, quais delas são sensíveis ao contexto e quais não. Justifique.
3. Dê uma gramática irrestrita que gere a linguagem  $\mathcal{L}_{Ack}$  (seção 9.4.4).
4. Dê uma gramática sensível ao contexto que gere as linguagens do exercício 1 sem o  $\lambda$ .
5. Transforme as gramáticas sensíveis ao contexto do exercício anterior à forma normal SC.
6. Transforme as gramáticas sensíveis ao contexto dos exemplos 9.4.2 e 9.4.3 à forma normal SC.
7. Dê ALL's que reconheçam as linguagens do exercício 1.

## **9.5. Exercícios**

---

## Capítulo 10

# Hierarquia de Chomsky e Classes de Complexidades de Linguagens

Muitas das discussões aqui, estritamente falando, são válidas somente para linguagens que não incluem a cadeia vazia. Esta restrição provém do fato de que as máquinas de Turing, como temos definidos, não podem aceitar a cadeia vazia. Para evitar ter de refazer a definição ou ter de adicionar repetidas observações, assumimos tacitamente que as linguagens discutidas aqui, a menos que explicitamente se diga o contrário, não contém  $\lambda$ . É claro que podemos, trivialmente, restabelecer tudo de modo a incluir a cadeia vazia.

Até agora, temos estudado uma série de tipos de autômatos e gramáticas e suas respectivas linguagens. Temos observado que algumas classes são mais abrangentes que outras classes de linguagens. Isto em princípio da uma hierarquia entre as classes de linguagens formais. O primeiro a formular este tipo de hierarquia foi primeiro formulada por Noam Chomsky, quem fez um análises de inclusão de quatro classes de linguagens. Posteriormente esta hierarquia tem sido estendida para considerar outras classes de linguagens.

As famílias de linguagens vistas aqui foram classificadas em termos dos modelos computacionais que a descrevem, não considerando o fator da complexidade ou eficiência de seu algoritmo de pertinência (se é linear ou não, por exemplo). Em teoria da computação, há uma preocupação maior em introduzir modelos computacionais e em estudar sua potencialidade, isto é, determinar o que podemos realizar no modelo em vez de como podemos resolver um problema nesse modelo. Mais na prática da computação, a eficiência dos algoritmos que resolvem os problemas é fundamental, e portanto este fator também deve ser considerado numa “teoria da computação”. Por esta razão, também vamos analisar a eficiência dos algoritmos, em termos de tempo e de espaço. Assim, uma outra maneira de classificar linguagens é usar uma máquina de Turing como o autômato comum subjacente, mas considerar a complexidade como fator de distinção. Para isso, definimos, na seção 10.2 algumas noções básicas sobre complexidade computacional, o que nos permitirá classificar linguagens pela complexidade de seus algoritmos de pertinência. Na seção 10.3, veremos como se relacionam algumas das classes de linguagens na hierarquia de Chomsky com as classes de complexidade.

### 10.1. A Hierarquia de Chomsky

---

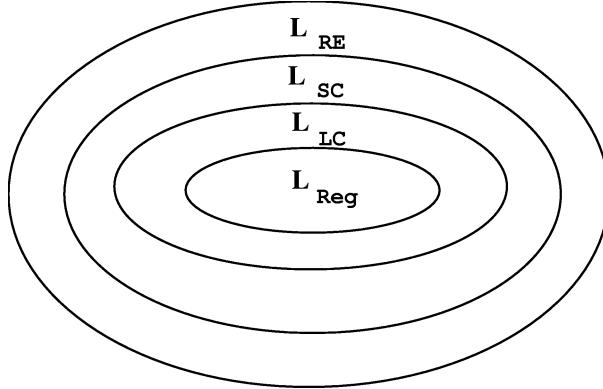


Figura 10.1: Hierarquia de Chomsky original.

## 10.1 A Hierarquia de Chomsky

Vimos a família das linguagens recursivamente enumeráveis ( $\mathcal{L}_{RE}$ ), a família das linguagens recursivas ( $\mathcal{L}_{Rec}$ ), a família das linguagens sensíveis ao contexto ( $\mathcal{L}_{SC}$ ), a família das linguagens livres do contexto ( $\mathcal{L}_{LC}$ ) e a família das linguagens regulares ( $\mathcal{L}_{Reg}$ ). Uma maneira de exibir as relações entre essas famílias é através da **hierarquia de Chomsky**. Noam Chomsky, um dos fundadores da teoria das linguagens formais, forneceu uma classificação inicial em quatro tipos de linguagens do tipo 0 ao tipo 3. Essa terminologia inicial persistiu e se encontra freqüentemente referências a ela. O tipo 0, o mais abrangente, é a família das linguagens geradas pelas gramáticas irrestritas, isto é, recursivamente enumeráveis. O tipo 1 consiste das linguagens sensíveis ao contexto ( $\mathcal{L}_{SC}$ ) que são geradas por gramática sensível ao contexto. O tipo 2 consiste das linguagens livres do contexto e o tipo 3 das linguagens regulares. Cada família de tipo  $i$  é um subconjunto próprio da família de tipo  $i - 1$ . O diagrama da figura 10.1 exibe, claramente, essa relação.

Podemos, também, acrescentar a essa hierarquia outras famílias de linguagens formais, apresentadas aqui de modo superficial, tais como: a família das linguagens recursivas ( $\mathcal{L}_{Rec}$ ), a família das linguagens lineares, ( $\mathcal{L}_{Lin}$ ), isto é as linguagens geradas por gramáticas lineares, e a família das linguagens livres do contexto determinísticas ( $\mathcal{L}_{LCD}$ ), isto é, linguagens livres do contexto reconhecidas por autômatos com pilha determinísticos. Os níveis no diagrama de Venn da figura 10.2 indicam que a respectiva classe está contida propriamente na outra. Assim, a família das linguagens recursivamente enumeráveis contém propriamente a família das linguagens recursivas, um exemplo de uma linguagem recursivamente enumerável não recursiva é a linguagem descrita no teorema 9.2.3. Por sua vez, a família das linguagens recursivas contém propriamente a família das linguagens sensíveis ao contexto, um exemplo de uma linguagem recursiva que não é sensível ao contexto é a linguagem  $L_{Ack}$  descrita em (9.5). Já a família das linguagens sensíveis ao contexto contém propriamente as linguagens livres do contexto, um exemplo de uma linguagem sensível ao contexto que não é livre do contexto é  $\mathcal{L} = \{a^n b^n a^n / n \geq 0\}$ . A família das linguagens livres do contexto contém propriamente a família das linguagens lineares e a família das linguagens livres do contexto determinística, por exemplo a linguagem  $\mathcal{L} = \mathcal{L}_{Pal}^2$ , onde  $\mathcal{L}_{Pal}$  é a linguagem dos palíndromos, é livre do contexto mas não é nem linear nem livre de

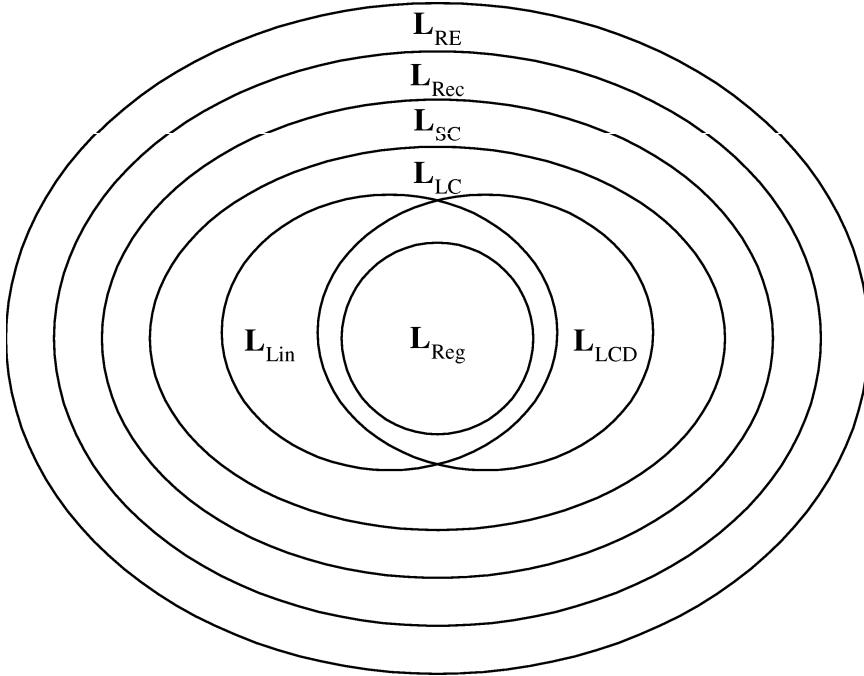


Figura 10.2: Hierarquia de Chomsky Estendida.

contexto determinístico. A família das linguagens lineares não contém a família das linguagens livres do contexto determinísticas e vice-versa, ou seja as linguagens livres do contexto determinísticas não contém a classe das linguagens lineares. Um exemplo de uma linguagem linear que não é livre do contexto determinística é  $\mathcal{L}_{Pal}$  e um exemplo de linguagem livre do contexto determinística mas que não é linear é a linguagem  $\mathcal{L} = \{a^m b^{m+n} a^n / m, n \geq 0\}$ . Finalmente, a família das linguagens que são ao mesmo tempo livres do contexto determinísticas e lineares contém a família das linguagens regulares, um exemplo de uma linguagem livre do contexto determinística e linear, ao mesmo tempo, mas não regular é a linguagem  $\mathcal{L} = \{a^n b^n / n \geq 0\}$ .

Assim, como temos versões determinísticas e não-determinísticas das classes das linguagens livres do contexto, seria razoável pensar em algo análogo para as outras classes da hierarquia. No caso das linguagens regulares dado que AFD's e AFN's são equivalentes, a versão determinística e não-determinística desta classe são a mesma. A classe das linguagens lineares é não-determinística, pois são definidas em termos de gramáticas lineares a quais são inherentemente não-determinísticas. Há diversas versões não-equivalentes de linguagens lineares determinísticas, veja por exemplo [?, Bed08], mas todas elas estritamente contidas na classe das linguagens lineares. A classe das linguagens sensíveis ao contexto também é não-determinística, pois é baseado em gramáticas que são inherentemente não-determinísticas. Como provado por Savitch [Sav73], as classes das linguagens sensíveis ao contexto e um tipo de sensíveis ao contexto determinísticos são exatamente as mesmas. Porém, ainda esta em aberto saber se a classe das linguagens aceitas por ALL's determinísticos coincide com a classe das linguagens sensíveis ao contexto [DW83, Men05]. As classes das linguagens recursivas e recursivas enumeráveis são determinísticas, pois são baseadas em máquinas de Turing determinísticas. Pela tese de

## 10.2. Complexidade Computacional

---

Church-Turing, a capacidade de tais máquinas não pode ser aumentada, por tanto a classe das linguagens recursivamente enumeráveis determinística e não-determinísticas são as mesmas e como corolário a classe das linguagens recursivas é equivalente à classe da linguagens recursivas não-determinísticas.

## 10.2 Complexidade Computacional

Em nosso estudo dos processos algoritmos que consideramos até aqui levamos em conta, somente, a possibilidade ou impossibilidade de computações específicas. Não encontramos diferenças essenciais entre os vários modelos de máquinas de Turing. Poderemos, sempre, usar simulações para mudar de um modelo para outro, de modo que a decibildade e a computabilidade não sejam afetados pelos detalhes da máquina de Turing que usamos. Isso, entretanto, ignora alguns fatos que se tornam relevantes quando são considerados aplicações, tais como exigências de tempo e espaço de computações em diferentes modelos. Na prática, estes elementos são muito importantes e portanto deveriam ser levados em consideração. Isto nos leva a um novo tópico, chamado **teoria da complexidade**. No estudo de complexidade, nosso interesse principal é com eficiência de uma computação como medida para requisitos de recursos.

A teoria da complexidade computacional é um tópico muito extenso, contendo assuntos inteiramente fora do nosso escopo. Entretanto, existem alguns resultados que podem ser estabelecidos de modo simples, trazendo alguns esclarecimentos sobre a natureza de linguagens e computações. Daremos, abaixo, um esboço de alguns resultados de complexidade. Não faremos a maior parte das provas difíceis, recomendando, [HU79, Pap94, Smi94], ao leitor interessado. Nossa intenção, aqui, é apresentar a relevância do assunto e mostrar como ele se relaciona com o que sabemos sobre linguagens e autômatos.

### 10.2.1 Medida e Complexidade

Na prática da computação, a questão fundamental sobre uma função  $f$ , não é se  $f$  é computável, mas se  $f$  é computável em termos úteis, isto é, se existe um programa que compute  $f$  no tempo e espaço que precisamos ou dispomos. A resposta dependerá, essencialmente, da nossa capacidade para escrever programas e da linguagem de programação e do computador usado. Porém, intuitivamente existe um fator adicional que pode ser descrito como a *complexidade intrínseca* da função  $f$ . Duas medidas básicas de complexidade são o tempo de computação e o espaço usado na computação.

Assim, daremos um significado preciso para os conceitos de tempo e espaço e sua conexão com as máquinas de Turing. Para obter isso pensamos da máquina de Turing como efetuando um movimento por unidade de tempo. De modo que o tempo gasto numa computação é o número de movimentos efetuados. Analogamente, o espaço pode significar o número total de células da fita que são usadas ou visitadas durante a computação. Agora, em geral, não há interesse em conhecer o espaço e o tempo exatos exigidos para uma única computação. Com estas noções precisas de medidas de complexidade computacional, podemos responder perguntas do tipo: Quão intrinsecamente complexa é uma função computável  $f$ ? e, é possível desenvolver um programa “melhor” para computar  $f$ ? Geralmente queremos saber como a máquina efetua todas as suas computações no domínio e também caracterizar como o espaço e o tempo requeridos dependem

## Capítulo 10. Hierarquia de Chomsky e Classes de Complexidades de Linguagens

do tamanho da cadeia de entrada, digamos seu comprimento. Dentre todos os problemas de um dado tamanho pode haver alguma variação, mas estaremos interessados somente no pior caso, isto é, o máximo de recursos requeridos de todos os problemas do mesmo tamanho. Usaremos  $n$  para denotar o tamanho do problema. Dizemos, então, que a computação tem **complexidade de tempo**  $T(n)$  para significar que a computação para todo problema de tamanho  $n$  pode ser completada em não mais do que  $T(n)$  movimentos. Se dizemos que a computação tem **complexidade de espaço**  $S(n)$ , queremos dizer que as computações usam no máximo  $S(n)$  células da fita. Exatamente o que significa um movimento ou uma célula da fita depende da estrutura da máquina de Turing específica e será deixado sem solução porque não queremos fazer uma distinção muito refinada. Suponha, por exemplo, que temos uma máquina de Turing que efetua uma computação usando um espaço de  $n$  células. Se agora, tomamos uma outra máquina de Turing que comprime a fita da primeira máquina usando  $\frac{n}{2}$  espaços de células de um tipo mais complicado. Como não dissemos o que é uma célula, a segunda alegação faz tanto sentido quanto a primeira. além disso, estamos geralmente interessados somente em computações em grandes problemas que requerem muito recursos. Conseqüentemente, pensamos de  $n$  como grande, e uma computação que efetua uma computação num tempo  $n^2$  não difere, significativamente, de uma que leva um tempo de  $n^2 + 2n$ . Estamos interessados na **ordem de magnitude** do tempo e do espaço requerido e, em geral, não prestamos atenção àqueles casos envolvendo fatores multiplicativos e termos de mais baixa ordem. No que segue assumiremos que isso está entendido. Quando dizemos que uma computação toma um tempo  $T(n)$  queremos, realmente, dizer que ela toma  $O(T(n))$ . Aqui usamos a  $O$ -notação para dar a ordem de magnitude do comportamento de uma função para  $n$  grande, ignorando possíveis fatores multiplicativos. Mais precisamente, dizemos que uma função  $f$  é da ordem da função  $g$  se, e somente se, existe alguma função  $c$ , tal que para todo natural  $n$

$$\frac{f(n)}{g(n)} \leq c(n), \text{ com } \lim_{n \rightarrow \infty} c(n) < \infty.$$

**Exemplo 10.2.1** No exemplo 8.1.10, construímos uma máquina de Turing com uma única fita para a linguagem

$$\mathcal{L} = \{a^n b^n / n \geq 1\}.$$

Uma olhada nesse algoritmo mostrará que para  $w = a^n b^n$  ele leva  $2n$  passos para casar o primeiro  $a$  com o último  $b$ , na etapa seguinte levará  $2n-2$ , e assim por diante. Portanto a computação total de  $a^n b^n$  deverá levar aproximadamente um tempo  $n^2$ , isto é,  $O(T(n)) = n^2$ .

Mas, com uma máquina de duas fitas podemos usar um algoritmo diferente. Primeiro copiamos todos os  $a$ 's na segunda fita e então comparamo-lo com os  $b$ 's na primeira fita. A situação antes e depois da cópia é mostrado na figura 10.3. Ambas, a cópia e o casamento, podem ser realizados em  $O(n)$  passos e portanto, vemos que com uma máquina de duas fitas, conseguíramos reconhecer a linguagem  $\mathcal{L}$  com uma complexidade de tempo  $n$ .

## 10.2. Complexidade Computacional

---

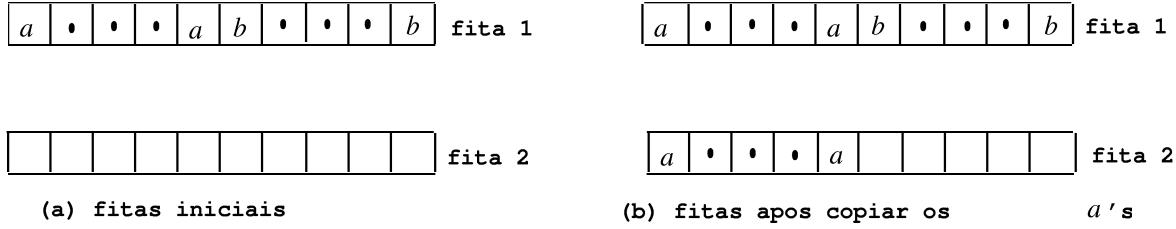


Figura 10.3: Configuração de uma Máquina de Turing de duas fitas que reconhece a linguagem  $a^n b^n$ .

### 10.2.2 Famílias e Classes de Complexidade

Na abordagem da hierarquia de Chomsky para classificação de linguagens, associamos famílias de linguagens com classes de autômatos, onde cada classe de autômatos é definida pela natureza de seu dispositivo temporário de armazenamento. Uma outra maneira de classificar as linguagens é usar uma máquina de Turing como o autômato comum subjacente, mas considerar a complexidade como fator de distinção. Para isso, devemos primeiro definir a complexidade de uma linguagem.

**Definição 10.2.2** Seja  $\mathcal{L}$  qualquer linguagem. Dizemos que uma máquina de Turing,  $M$ , aceita  $\mathcal{L}$ , em tempo  $T(n)$ , se para cada  $w \in \mathcal{L}$ , com  $|w| \leq n$ , é aceito por  $M$  em  $O(T(n))$  ou menos movimentos. Similarmente,  $\mathcal{L}$  é aceito por  $M$  em espaço  $S(n)$ , se  $M$  requerer, no máximo,  $O(S(n))$  células para aceitar  $w$ .

As famílias de linguagens podem ser definidas de acordo com as diversas medidas de complexidade. A seguinte são as maiores classes que consideraremos. Uma linguagem diz-se da classe  $DTempo(T(n))$  se existe alguma máquina de Turing multi-fita determinística que aceita esta em tempo  $T(n)$ . Similarmente,  $NTempo(T(n))$  é o conjunto de todas as linguagens aceitas por alguma máquina de Turing não determinística em tempo  $T(n)$ . Nas máquinas não determinísticas, requeremos que exista ao menos uma possível computação que possa ser completada nesse tempo. Analogamente,  $DEspaço(S(n))$  é o conjunto de todas as linguagens aceitas por alguma máquina de Turing determinística em espaço  $S(n)$ , e  $NEspaço(S(n))$  é o análogo não-determinístico.

A definição dessas classes de complexidade não são suficientes para nos proporcionar uma boa idéia da natureza dessas classes. O restante desta parte é uma exploração dessas classes com o intuito de dar um melhor entendimento delas e mostrar como elas estão relacionadas com as classificações prévias para linguagens. O primeiro grupo de questões exprimem a relação existente entre estas quatro classes de linguagens. Algumas delas saem por considerações elementares. Como, no mínimo, são necessários  $n$  movimentos para ler cada símbolo de uma entrada de tamanho  $n$ ,  $DTempo(T(n))$  não é interessante para  $T(n) < n$ . Este consiste de linguagens para cujos membros dependem somente de uma parte da entrada. No entanto, não podemos dizer o mesmo para  $DEspaço$ , pois a cadeia de entrada não é contada. Por exemplo,  $DEspaço(\log n)$  inclui muitas linguagens não triviais.

## **Capítulo 10. Hierarquia de Chomsky e Classes de Complexidades de Linguagens**

Como uma máquina de Turing determinística é um caso especial de uma máquina de Turing não determinística, podemos imediatamente concluir que

$$DTempo(T(n)) \subseteq NTempo(T(n)) \quad (10.1)$$

e

$$DEspaço(S(n)) \subseteq NEspaço(S(n)) \quad (10.2)$$

Também, como somente podemos ganhar usando recursos adicionais,

$$DTempo(T_1(n)) \subseteq DTempo(T_2(n)) \quad (10.3)$$

e

$$DEspaço(S_1(n)) \subseteq DEspaço(S_2(n)), \quad (10.4)$$

devem ser verdadeiras para todo  $T_1(n) \leq T_2(n)$  e  $S_1(n) \leq S_2(n)$ . Resultados análogos são verdadeiros para outras classes de complexidade. Finalmente, como cada movimento de uma máquina de Turing usa ao menos uma nova célula, temos que

$$DTempo(f(n)) \subseteq DEspaço(f(n)) \quad (10.5)$$

para toda função  $f(n)$ .

Cada uma das relações, acima, embora imediatas, sugerem várias questões. O máximo que poderemos fazer é destacar alguns teoremas, e, então mostrar como eles esclarecem as relações entre as várias classes de complexidade que provém de um certo conjunto restrito. Este conjunto, entretanto, é bastante grande e inclui todas as funções comuns tais como  $n^i$ ,  $i = 1, 2, \dots, 2^n, n!$  e seus produtos. Por este motivo, ignoraremos as limitações técnicas que são necessárias para tornar completos os enunciados dos teoremas abaixo.

**Teorema 10.2.3** *Sejam  $S_1$  e  $S_2$  duas funções tais que*

$$\lim_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$$

*Então,  $DEspaço(S_1) \subset DEspaço(S_2)$*

DEMONSTRAÇÃO: Ver [HU79, Teorema 12.8]. ■

**Exemplo 10.2.4** *Existe uma linguagem  $DEspaço(n)$  que não está em  $DEspaço(\sqrt{n} \log n)$ .*

*Isto segue imediatamente do teorema 10.2.3 e do fato de que  $\lim_{n \rightarrow \infty} \frac{\sqrt{n} \log n}{n} = 0$ .*

### 10.3. Classificação da Complexidade e a Hierarquia de Chomsky

---

**Teorema 10.2.5** Sejam  $T_1$  e  $T_2$  duas funções tais que

$$\lim_{n \rightarrow \infty} \frac{T_1(n) \log T_1(n)}{T_2(n)} = 0, \quad (10.6)$$

então  $DTempo(T_1) \subset DTempo(T_2)$ .

DEMONSTRAÇÃO: Ver [HU79, Teorema 12.9]. ■

**Exemplo 10.2.6** Para todo  $k \geq 1$ , existe uma linguagem em  $DTempo(n^{k+1})$  que não está em  $DTempo(n^k)$ . Isto segue imediatamente do teorema 10.2.5, com

$$\lim_{n \rightarrow \infty} \frac{n^k \log n^k}{n^{k+1}} = \lim_{n \rightarrow \infty} \frac{k \log n}{n} = 0.$$

Esses dois teoremas, junto com os exemplos, mostram que existe uma hierarquia de família de linguagens contendo um número infinito de membros distintos. Um ligeiro crescimento na ordem de magnitude da função de complexidade de tempo ou do espaço define uma nova e maior família de linguagens.

Um conjunto de questões interessantes vem ao perseguir ( 10.1) e ( 10.5). É a inclusão própria. Em geral, como não-determinismo reduz a complexidade? Um resultado bem conhecido relacionado com isto é chamado teorema de Savitch.

**Teorema 10.2.7 (Savitch)** Seja  $S(n) \geq \log n$ . Então  $NEspaço(S(n)) \subseteq DEspaço(S(n)^2)$ .

DEMONSTRAÇÃO: Vêr [HU79, Teorema 12.11]. ■

O teorema de Savitch nos diz que, embora uma máquina determinística possa precisar de mais espaço para uma computação do que a correspondente não-determinística, os recursos extras requeridos são previsíveis e igual a não mais do que o quadrado de  $S(n)$ . Questões similares sobre a relação entre  $DTempo$  e  $NTtempo$  não estão ainda totalmente bem entendidas.

## 10.3 Classificação da Complexidade e a Hierarquia de Chomsky

Os resultados estabelecidos anteriormente nos permite fazer várias alegações, por exemplo, que para todo  $k > 3$  existe uma linguagem em  $DTempo(n^k)$  que não está em  $DTempo(n^3)$ . Embora isto possa ser de interesse teórico, não está claro que esse resultado tenha qualquer significado prático. Neste caso, não temos nenhuma idéia de como devem ser as características de uma linguagem em  $DTempo(n^4)$ . Podemos progredir no entendimento dessa matéria se relacionarmos a classificação de complexidade e a hierarquia de Chomsky. A seguir veremos alguns exemplos simples.

## **Capítulo 10. Hierarquia de Chomsky e Classes de Complexidades de Linguagens**

**Exemplo 10.3.1** Toda linguagem regular pode ser reconhecida por um autômato finito determinístico, num tempo proporcional ao comprimento da entrada. Portanto

$$\mathcal{L}_{Reg} \subseteq DT\text{empo}(n)$$

Mas,  $DT\text{empo}(n)$  inclui muito mais do que  $\mathcal{L}_{Reg}$ . Já estabelecemos, no exemplo 10.2.1, que a linguagem livre do contexto  $\{a^n b^n / n \geq 0\}$  pode ser reconhecida no tempo  $O(n)$ . O argumento dado pode ser usado mesmo para linguagens mais complicadas.

**Exemplo 10.3.2** A linguagem não livre do contexto  $\mathcal{L} = \{ww / w \in \{a, b\}^*\}$  está em  $NT\text{empo}(n)$ . Isto sai diretamente, uma vez que podemos reconhecer cadeias nesta linguagem pelo algoritmo

1. Copie a entrada da fila de entrada para a fita 1. Não-deterministicamente adivinhe o meio da cadeia.
2. Copie a segunda parte para a fita 2.
3. Compare os símbolos da fita 1 e da fita 2 um a um.

Claramente, todos os passos podem ser efetuados num tempo  $O(|w|)$ . Logo,  $\mathcal{L} \in NT\text{empo}(n)$ .

Note que podemos mostrar que  $\mathcal{L} \in DT\text{empo}(n)$ , fornecendo um algoritmo para encontrar o meio de uma cadeia em tempo  $O(n)$ . Este algoritmo é o seguinte: Para cada símbolo da fita 1 lê dois símbolos da fita 2. Quando terminar de ler todos os símbolos na fita o cabeçote da fita 1 estará posicionado no meio da cadeia.

**Exemplo 10.3.3** Desde que toda linguagem livre do contexto pode ser analisada em tempo  $O(n^3)$ , concluímos que toda linguagem livre do contexto está em  $DT\text{empo}(n^3)$ . Mas, como o exemplo acima mostra, existem linguagens em  $DT\text{empo}(n^3)$  que não são livres do contexto. Portanto,

$$\mathcal{L}_{LC} \subset DT\text{empo}(n^3).$$

Se permitirmos não-determinismo, a análise das linguagens livres do contexto é mais simples. Adivinhando a produção correta em cada etapa, podemos analisar em tempo proporcional ao comprimento da derivação. Mas, sempre existirá uma derivação proporcional a  $|w|$ , de modo que

$$\mathcal{L}_{LC} \subset NT\text{empo}(n).$$

**Teorema 10.3.4** Não existe uma função total Turing Computável  $f$  tal que toda linguagem recursiva esteja em  $DT\text{empo}(f(n))$ .

#### 10.4. A Classe de Complexidade $P$ e $NP$

---

**DEMONSTRAÇÃO:** Considere o alfabeto  $\Sigma = \{0, 1\}$  e a **ordem própria**  $\leq_P$  sobre  $\Sigma^+$ , isto é,  $w_1 \leq_P w_2$  se, e somente se,  $||w_1| \leq ||w_2|$  e  $w_1 \leq_L w_2$ , onde  $\leq$  é a ordem usual dos números naturais e  $\leq_L$  é a ordem léxica. Note que também é possível definir uma ordem própria sobre as máquinas de Turing, baseadas na codificação de Gödel.

Suponha, agora, que a função  $f(n)$  do enunciado do teorema existe. Podemos, então definir a linguagem

$$\mathcal{L} = \{w_i / M_i \text{ não aceita } w_i \text{ em } f(|w_i|) \text{ etapas}\} \quad (10.7)$$

Seja  $w \in \mathcal{L}$  e compute  $f(|w|)$ , assumindo que  $f$  é uma função Turing computável total. Em seguida achamos a posição  $i$  de  $w$  na seqüência  $w_1, w_2, \dots, w_n, \dots$ . Isto também é possível porque a seqüência está na ordem própria. Quando tivermos  $i$ , achamos  $M_i$  e deixamo-la operar sobre  $w$  para  $f(|w|)$  etapas. Isso nos dirá se  $w \in \mathcal{L}$  ou não. Portanto  $\mathcal{L}$  é recursiva.

Agora, podemos mostrar que  $\mathcal{L}$  não está em  $DTempo(f(n))$ . Suponha que ela estivesse. Como  $\mathcal{L}$  é recursiva existe alguma máquina de Turing  $M_k$ , tal que  $\mathcal{L} = L(M_k)$ .  $w_k$  está em  $\mathcal{L}$ ? se alegarmos que  $w_k \in \mathcal{L}$ , então  $M_k$  aceita  $w_k$  em  $f(|w_k|)$  etapas. Isto acontece porque  $\mathcal{L} \in DTempo(f(n))$  e todo  $w \in \mathcal{L}$  é aceito por  $M_k$ , em tempo  $f(|w|)$ . Mas isso contradiz (10.7).

Inversamente, se assumirmos que  $w_k \notin \mathcal{L}$  obteremos uma contradição. A impossibilidade para resolver este fato é um resultado típico de diagonalização e leva-nos a concluir que a hipótese original, qual seja a existência de uma  $f(n)$  computável, deve ser falsa. ■

## 10.4 A Classe de Complexidade $P$ e $NP$

Diferenças pequenas como entre  $DTempo(n^k)$  e  $DTempo(n^{k-1})$  são irrelevantes, pois para um  $n$  suficientemente grande elas são muito próximas e portanto de pouco interesse. Mais ainda, a complexidade depende do modelo específico de máquina de Turing considerado, por exemplo, com uma quantidade determinada de fitas. Como não é consensual qual tipo de máquina de Turing é o mais apropriado para modelar um computador real, consideraremos classes de complexidade suficientemente abrangentes como para não depender do modelo de máquina de Turing particular. De fato único que será relevante é se a máquina de Turing é determinística ou não-determinística. Assim, consideraremos a famosa classe de complexidade dos *problemas polinomiais*:

$$P = \bigcup_{i \geq 1} DTempo(n^i),$$

a qual inclui todas as linguagens que são reconhecidas por alguma máquina de Turing determinística em tempo polinomial, sem qualquer preocupação com o grau do polinômio nem com as características específicas da máquina de Turing determinística usada (ver seção 8.5). Como já vimos  $\mathcal{L}_{Rec}$  e  $\mathcal{L}_{LC}$  estão em  $P$ .

Podemos fazer definições análogas para as outras medidas de complexidades introduzindo a classe  $NP$  de todas as linguagens que são reconhecidas por alguma máquina de Turing não-determinística em tempo polinomial, sem qualquer preocupação com o grau do polinômio nem com as características específicas da máquina:

## Capítulo 10. Hierarquia de Chomsky e Classes de Complexidades de Linguagens

$$NP = \bigcup_{i \geq 1} NT\text{empo}(n^i)$$

Analogamente, para o caso de considerar espaço em vez de tempo temos :

$$P-\text{Espaço} = \bigcup_{i \geq 1} D\text{Espaço}(n^i)$$

$$NP-\text{Espaço} = \bigcup_{i \geq 1} N\text{Espaço}(n^i).$$

Existem algumas relações óbvias entre essas classes. Por causa das restrições de tempo e espaço de uma máquina de Turing, temos

$$NP \subseteq NP-\text{Espaço}.$$

Do teorema de Savitch segue que

$$P-\text{Espaço} = NP-\text{Espaço}.$$

Portanto, temos

$$P \subseteq NP \subseteq P-\text{Espaço} = NP-\text{Espaço}.$$

O que não sabemos é se essas continências são próprias ou não.

O objetivo de se estudar estas classes de complexidade, particularmente a classe  $P$ , é tentar de distinguir entre computações realísticas e não realísticas. Certas computações, embora teoricamente possíveis, requerem tanto recursos que na prática elas devem ser consideradas como irrealísticas, mesmo para os computadores existentes e até para supercomputadores ainda a serem projetados. Tais problemas são às vezes chamados *intratáveis*, para indicar que, embora em princípio sejam computáveis, não existe esperança real de um algoritmo prático. Para entender isso melhor, os cientistas da computação têm tentado pôr a idéia da intratabilidade numa base formal. Uma tentativa de definir o termo intratável é feita no que é, geralmente, chamado de *tese de Cook-Karp*. Na tese de Cook-Karp, um problema que está em  $P$  é chamado tratável, um que não está diz-se intratável.

A tese de Cook-Karp, embora em princípio possa parecer uma boa maneira de separar problemas que podemos trabalhar realisticamente daqueles que não podemos, não é um consenso da comunidade científica. Obviamente, qualquer computação que não está em  $P$  tem tempo de complexidade que cresce mais rápido que qualquer polinômio, e seus requerimentos crescerão muito rapidamente com o tamanho do problema. Mesmo para a função como  $2^{\lfloor \frac{n}{10} \rfloor}$ , isto será excessivo para  $n$  grande, digamos  $n \geq 1000$ . Portanto é razoável considerar um problema com essa complexidade como sendo intratável. Mas que dizer de problemas que estão em  $D\text{Tempo}(n^{100})$ ? Enquanto a tese de Cook-Karp chama tais problemas tratáveis, de fato, certamente não se poderá fazer muito com eles, mesmo para  $n$  pequeno, ou seja seriam irrealísticos. A justificativa para a

## 10.4. A Classe de Complexidade $P$ e $NP$

---

tese de Cook-Karp parece se sustentar na observação empírica de que muitos problemas práticos em  $P$  estão em  $DTempo(n)$ ,  $DTempo(n^2)$  ou  $DTempo(n^3)$ , enquanto outros fora dessa classe parecem ter complexidade exponencial. Dentre os problemas práticos, existe uma distinção clara entre problemas  $P$  e aqueles que não estão em  $P$ .

A diferença fundamental entre os problemas  $P$  e problemas  $NP$ , está em que a segunda classe contém a palavra extra “não-determinístico” o qual indica uma diferença, aparentemente, intransponível, no sentido de que um problema em  $P$  pode ser resolvido em tempo polinomial, enquanto os problemas em  $NP$  são resolvidos em tempo polinomial, mas por uma máquina de Turing não-determinística. O não-determinismo traz em si a idéia de computação paralela, pois as diversas possíveis escolhas dos movimentos que uma máquina de Turing não-determinística pode realizar, a cada passo, podem ser simuladas, na computação prática, via o uso de diversos processadores em paralelo. Por outro lado, qualquer tentativa de se transformar as máquinas de Turing não-determinísticas em determinísticas, resultaria numa máquina de Turing, para alguns casos, com uma complexidade exponencial.

O estudo da relação entre as classes de complexidades  $P$  e  $NP$  tem gerado interesse particular entre os cientistas da computação. Na raiz disto está a questão de se

$$P = NP$$

ou não.

A resposta ainda não é conhecida, de fato há um fabuloso prêmio de um milhão de dólares para a pessoa (ou grupo de pessoas) que conseguir responder positiva ou negativamente a questão (veja o site <http://www.claymath.org/millennium/>). Um resposta positiva a esta questão terá devastadoras consequências no campo de segurança de dados, uma vez que a segurança criptográfica de códigos se baseia na intratabilidade de quebrar (descobrir) as chaves de codificação. Felizmente, embora ninguém tenha sido capaz até agora de prová-la, existem boas razões para se acreditar que existem problemas em  $NP$  que não estão em  $P$  e, portanto, não podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística. Muitas das evidências vêm de exemplos negativos. Problemas como do caixeiro viajante está em  $NP$ . Mas, em que pese as tentativas, não se tem encontrado um algoritmo de tempo polinomial para ele. Esta falha, e outras, nos leva a acreditar que  $P \neq NP$ . Mas a evidência para essa crença é estritamente circunstancial. Este problema é discutido amplamente em [Kar72, Pap94].

## 10.5 Exercícios

1. Situe na hierarquia de Chomsky estendida (figura 10.2) as seguintes linguagens:
  - (a)  $\mathcal{L}_1 = \{a^k / a^k \in L(M_k)\}$  onde  $M_k$  é a k-ésima máquina de Turing na enumeração de Gödel.
  - (b)  $\mathcal{L}_2 = \{a^k b^m c^n / 1 \leq k + m \leq n\}$
  - (c)  $\mathcal{L}_3 = \{a^k b^m c^n / 1 \leq k + m + n \leq 8\}$
  - (d)  $\mathcal{L}_4 = \{a^k b^m c^n / 1 \leq k \leq m \leq n\}$
  - (e)  $\mathcal{L}_5 = \{a^m b^m a^n b^n / m, n \geq 1\}$
  - (f)  $\mathcal{L}_6 = \{a^m b^n a^n b^m / m, n \geq 1\}$
  - (g)  $\mathcal{L}_7 = \{a^{k!} / k \geq 1\}$
2. Mostre que  $\mathcal{L} = \{ww^Rw / w \in \{a, b\}^+\}$  está em  $DTempo(n)$ .
3. Encontre um algoritmo que aceite a linguagem  $\mathcal{L} = \{ww^R / w \in \{a, b\}^+\}$  com uma complexidade de tempo  $O(n)$ .

## **10.5. Exercícios**

---

## Capítulo 11

# Limites da Computação Algorítmica: Problemas Indecidíveis

Embora a tese de Turing nos leve a acreditar que essas limitações não são muitas, um argumento análogo ao teorema 9.2.2, nós leva à conclusão que há uma quantidade não-contável de funções que não podem ser computadas por máquinas de Turing e que portanto não existem algoritmos para resolver certos problemas. A equação (9.1) apresenta um exemplo explícito de uma linguagem que não é recursivamente enumerável e que por tanto não pode ser reconhecido por uma máquina de Turing, deixando em evidência que máquinas de Turing são limitadas. O argumento de que o poder das computações mecânicas é limitado não é surpreendente. Intuitivamente, sabemos que muitas questões vagas e especulativas requerem idéias e raciocínios bem além da capacidade de qualquer computador previsível. O que é mais interessante para o cientista da computação é que existem questões que podem ser descritas de forma clara e simples, que aparentemente têm soluções algorítmicas, mas as quais não são solúveis por qualquer computador.

Antes de tudo, definiremos os conceitos de **decibilidade** e **computabilidade** para precisar o que queremos dizer com a afirmação de que algo não pode ser feito por uma máquina de Turing. Veremos, então, vários problemas clássicos desse tipo, dentre eles o bem conhecido problema da parada para máquinas de Turing. Desse problema segue um número de problemas relacionados para máquinas de Turing e linguagens recursivamente enumeráveis. Esses problemas podem parecer artificiais, porém veremos que há diversas as questões de relevância prática, no estudo das linguagens de programação, por exemplo “determinar se uma gramática livre de contexto é ambígua ou não”, para as quais não existem algoritmos.

### 11.1 Computabilidade e Decibilidade

Na definição 8.1.12, estabelecemos que uma função  $f$ , num certo domínio, diz-se computável se existe uma máquina de Turing que computa o valor de  $f$ , para todos os argumentos no seu domínio. Como funções mapeiam elementos de um conjunto (domínio) em outro conjunto (contradomínio) e máquinas de Turing, por outro lado, transformam cadeias de um alfabeto ( $\Sigma$ ) em cadeias de um alfabeto ( $\Gamma$ ), devemos encarar  $\Sigma^*$  e  $\Gamma^*$ , como codificações ou representações do domínio e contradomínio, respectivamente. Uma função é não computável se tal máquina

## 11.2. O Problema da Parada para Máquinas de Turing

---

de Turing não existe. Pode existir uma máquina de Turing que computa  $f$  em parte de seu domínio, mas, para nós, uma função é computável somente se existe uma máquina de Turing que computa a função sobre todo seu domínio. Disso podemos inferir que quando classificamos uma função como computável ou não computável, devemos ser claros sobre o que é seu domínio.

Por simplicidade, estudaremos uma classe de problemas específica: a dos **problemas de decisão**, que são problemas de determinar se um elemento de algum universo pertence ou não a um determinado conjunto. O problema de decidir se um elemento qualquer do universo  $U$  pertence ou não a um conjunto  $A$  é denotado por  $P_A^U$ , quando o universo  $U$  for claro do contexto simplesmente denotaremos este problema por  $P_A$ . Se existir um algoritmo que receba um elemento  $a \in U$  e dê como resultado um simples “sim” caso  $a \in A$  ou “não”, caso  $a \notin A$ , dizemos que  $P_A^U$  é **decidível**. Se tal algoritmo não existir, então dizemos que  $P_A^U$  é **indecidível**. Observe que todo problema decidível é computável, no sentido de que existe um algoritmo que computa uma solução para o problema, mas o contrário não é correto, uma vez que a decibildade está atrelada à classe de problemas de decisão. Um problema de decisão pode ser entendido como um conjunto de afirmações que são verdadeiras ou falsas, dependendo do objeto sobre o qual predram. Por exemplo, consideremos a afirmação “Para uma gramática livre de contexto,  $G$ , a linguagem  $L(G)$  é inherentemente ambígua”. Para algumas gramáticas livres de contexto,  $G$ , isso é verdadeiro, mas para outras é falso. Porém, para qualquer gramática livre de contexto,  $G$ , devemos ter uma ou outra situação. Como o problema é decidir se a afirmação é verdadeira para alguma gramática livre de contexto, o domínio ou universo subjacente, é o conjunto de todas as gramáticas livres de contexto. Assim, um problema é decidível se existir uma máquina de Turing que dá a resposta correta para cada afirmação no domínio do problema.

Quando estabelecemos resultados de decibildade ou indecibildade, deveremos, sempre, saber qual é o domínio, porque pode afetar a conclusão. Um problema pode ser decidível sobre algum domínio mas não sobre outro. Especificamente uma única instância do problema é sempre decidível, desde que a resposta é ou verdadeira ou falsa. No primeiro caso, a máquina de Turing que sempre responde “verdadeiro” da a resposta correta, enquanto no segundo caso uma que responde sempre “falso” seria a correta. Isso pode parecer uma resposta falaciosa, mas enfatiza um ponto importante. O fato de que não sabemos qual a resposta correta não faz nenhuma diferença, o que importa é que existe alguma máquina de Turing que dê a resposta correta.

## 11.2 O Problema da Parada para Máquinas de Turing

O mais conhecido dos problemas indecidíveis é o **problema da parada**, para as máquinas de Turing. Este problema além de ter um significado histórico, nos dá um ponto de partida para desenvolver outros resultados. O problema da parada pode ser enunciado de modo simples como: *dada a descrição de uma máquina de Turing  $M$  e uma entrada  $w$ , quando iniciado na configuração inicial  $q_0w$ , ela efetua uma computação que pára?* usando uma maneira abreviada de falar do problema, perguntamos se  $M$  aplicada a  $w$ , ou simplesmente  $(M, w)$ , pára ou não pára. O domínio desse problema é o conjunto de todas as máquinas de Turing e todo  $w$ , isto é, estamos procurando uma única máquina de Turing que, dada a descrição de uma máquina de Turing arbitrária  $M$  e  $w$ , prevê se a computação de  $M$ , quando aplicada a  $w$ , parará ou não. Seja  $\mathcal{M}$  o conjunto das máquinas de Turing,  $G$  a codificação de Gödel e  $H = \{M \in \mathcal{M} / M \text{ aplicado a } G(M) \text{ pára}\}$ , então o problema da parada seria  $P_H^{\mathcal{M}}$ .

## Capítulo 11. Limites da Computação Algorítmica: Problemas Indecidíveis

Observe que não poderemos achar a resposta a este problema simplesmente simulando a ação de  $M$  sobre  $w$  através da máquina de Turing universal, pois não existe limite no comprimento da computação. Assim, caso  $M$  entre num laço infinito, não importa o tempo que esperemos, jamais poderemos estar seguros de que  $M$ , realmente, está num laço. Pode acontecer, simplesmente, o caso de uma computação muito longa, por exemplo algoritmos com uma complexidade de tempo da ordem  $O(2^n)$ , para  $n$  suficientemente grande, podem demorar séculos em terminar, mesmo sendo executada no computador mais potente da atualidade. O que precisamos é um algoritmo que possa determinar a resposta correta para qualquer  $M$  e  $w$ , ao efetuar uma análise na descrição da máquina e a entrada. Mas, como veremos, tal algoritmo não existe.

Para discussões posteriores, é conveniente ter uma idéia precisa do que seria uma solução para o problema da parada. Por isso, faremos a seguinte definição.

**Definição 11.2.1** Uma solução para o problema da parada é uma máquina de Turing  $M_H$ , a qual para qualquer máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  e  $w \in \Sigma^+$ , fornece a computação

$$q_0 G(M) w \vdash^* x_1 q_f x_2,$$

se  $M$  aplicado a  $w$  pára, e

$$q_0 G(M) w \vdash^* y_1 q_n y_2,$$

se  $M$  aplicado a  $w$  não pára. Aqui  $q_f$  e  $q_n$  são ambos estados finais de  $M_H$  e  $G(M)$  é a codificação de Gödel da máquina de Turing  $M$ .

**Teorema 11.2.2** Não existe qualquer máquina de Turing,  $M_H$ , que resolva o problema da parada, isto é se comporta como as exigências da definição 11.2.1. Portanto, o problema da parada é indecidível.

**DEMONSTRAÇÃO:** Assumiremos o contrário, isto é, que existe um algoritmo, e consequentemente uma máquina de Turing  $M_H$ , que resolve o problema da parada. A entrada para  $M_H$  será a descrição codificada de  $M$ , isto é  $G(M)$ , assim como a entrada  $w$ . A exigência é, então, que dado qualquer  $(G(M), w)$ , a máquina de Turing  $M_H$  parará com um sim ou um não. Conseguiremos isso pedindo que  $M_H$  pare num dos dois estados finais correspondentes, digamos,  $q_f$  e  $q_n$ . A situação pode ser visualizada por um diagrama de bloco como na figura 11.1.

Esse diagrama tem a intenção de indicar que, se  $M_H$  começar no estado  $q_0$ , com entrada  $(G(M), w)$  ela parará no estado  $q_f$  se  $M$  parar com a entrada  $w$  ou  $q_n$  caso contrário. Pela exigência da definição 11.2.1 devemos pedir que  $M_H$  opere segundo as seguintes regras:

$$q_0 G(M) w \vdash^* x_1 q_f x_2,$$

se  $M$  aplicado  $w$  pára, e

$$q_0 G(M) w \vdash^* y_1 q_n y_2,$$

## 11.2. O Problema da Parada para Máquinas de Turing

---

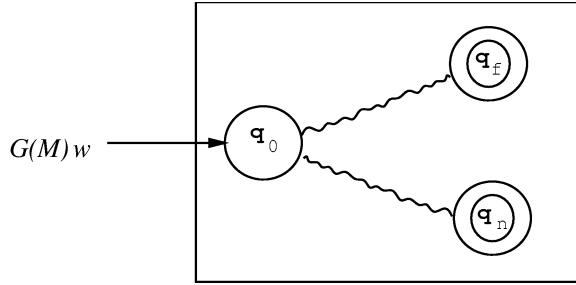


Figura 11.1: Diagrama de blocos para uma suposta máquina de Turing que resolve o problema da parada.

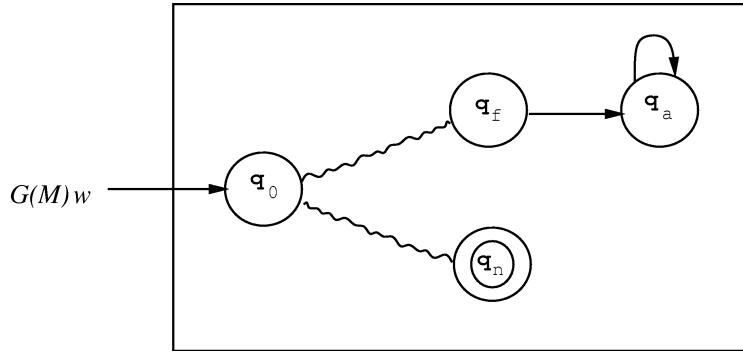


Figura 11.2: Diagrama de blocos para uma variante da suposta máquina de Turing que resolve o problema da parada

se  $M$  aplicado a  $w$  não pára.

Em seguida modificamos  $M_H$  para produzir uma máquina de Turing  $M_{H'}$ , como da figura 11.2.

Com o novo estado acrescentado a  $M_H$  queremos assegurar que existem transições entre o estado  $q_f$  e o novo estado  $q_a$ , não importando o símbolo que está sendo lido na fita, de tal modo que a fita se mantém inalterada. A maneira como é feita é direta. Comparando  $M_H$  e  $M_{H'}$  observamos que, na situação onde  $M_H$  atinge  $q_f$  e pára, a máquina modificada  $M_{H'}$  entra num laço infinito. Formalmente, a ação de  $M_{H'}$  é descrita por

$$q_0 G(M)w \vdash^*_{M_{H'}} \infty,$$

se  $M$  aplicada a  $w$  pára, e

$$q_0 G(M)w \vdash^*_{M_{H'}} y_1 q_n y_2,$$

se  $M$  aplicada a  $w$  não pára.

## Capítulo 11. Limites da Computação Algorítmica: Problemas Indecidíveis

---

De  $M_{H'}$  construímos uma outra máquina de Turing  $M_{\hat{H}}$ . Esta nova máquina toma a entrada  $G(M)$ , a cópia, e então se comporta exatamente com  $M_{H'}$ . Assim, a ação de  $M_{\hat{H}}$  é tal que

$$q_0 G(M) \vdash^*_{M_{\hat{H}}} q_0 G(M) G(M) \vdash^*_{M_{\hat{H}}} \infty,$$

se  $M$  aplicada a  $G(M)$  pára, e

$$q_0 G(M) \vdash^*_{M_{\hat{H}}} q_0 G(M) G(M) \vdash^*_{M_{\hat{H}}} y_1 q_n y_2,$$

se  $M$  aplicado a  $G(M)$  não pára.

Agora,  $M_{\hat{H}}$  é uma máquina de Turing, que terá a cadeia  $G(M_{\hat{H}})$  como descrição em  $\Sigma^+$ . Essa cadeia além de ser a descrição de  $M_{\hat{H}}$  pode, também, ser usada como entrada. Podemos, então perguntar o que aconteceria se  $M_{\hat{H}}$  fosse aplicada a  $G(M_{\hat{H}})$ . Assim,

$$q_0 G(M_{\hat{H}}) \vdash^*_{M_{\hat{H}}} \infty,$$

se  $M_{\hat{H}}$  aplicada a  $G(M_{\hat{H}})$  pára, e

$$q_0 G(M_{\hat{H}}) \vdash^*_{M_{\hat{H}}} y_1 q_n y_2,$$

se  $M_{\hat{H}}$  aplicada a  $G(M_{\hat{H}})$  não pára. Isto é claramente um absurdo. Essa contradição resultou da suposição de que  $M_H$  existe e portanto a decibildade do problema da parada, deve ser falsa.

■

Observe que o teorema 11.2.2 não proíbe a solução do problema da parada para casos específicos. De fato, para algumas máquinas de Turing  $M$  particulares podemos dizer, via uma análise de  $M$  e  $w$ , se ela parará ou não para a entrada  $w$ . O que o teorema diz é que isso não pode ser feito sempre, ou seja que não existe um algoritmo que pode tomar a decisão correta para todo  $M$  e  $w$ .

O argumento para provar o teorema 11.2.2 foi dado porque ele é clássico e de interesse histórico. A conclusão do teorema, como mostra o seguinte argumento, pode ser derivada de resultados anteriores.

**Teorema 11.2.3** *Se o problema da parada fosse decidível, então toda linguagem recursivamente enumerável seria recursiva. Por tanto, o problema da parada é indecidível.*

**DEMONSTRAÇÃO:** Para ver isto, seja  $\mathcal{L}$  uma linguagem recursivamente enumerável sobre  $\Sigma$ , e seja  $M$  uma máquina de Turing que reconhece  $\mathcal{L}$ . Seja  $M_H$  a máquina de Turing que resolve o problema da parada. Construímos o seguinte procedimento:

1. Aplique  $M_H$  a  $G(M)$ . Se  $M_H$  responder “não” (isto é, se parar no estado  $q_n$ ), então por definição,  $w \notin \mathcal{L}$ .

### 11.3. Redução de um Problema Indecidível ao Problema da Parada

---

2. Se  $M_H$  disser “sim” (isto é, se parar no estado  $q_f$ ), então aplique  $M$  a  $w$ . Mas,  $M$  deve parar. Portanto ele dirá mais cedo ou mais tarde se  $w \in \mathcal{L}$  ou não.

Isto constitui um algoritmo de pertinência, tornando  $\mathcal{L}$  recursiva. Mas já sabemos que existem linguagens recursivamente enumeráveis que não são recursivas. A contradição nos diz que o problema da parada é indecidível. ■

**Corolário 11.2.4 (Problema da pertinência para a classe das linguagens recursivamente enumeráveis)**  
*Não existe qualquer algoritmo capaz de decidir se dada uma cadeia e uma máquina de Turing qualquer, essa máquina de Turing aceita ou não tal cadeia.*

Ou seja, este corolário nos diz que não existe um algoritmo de pertinência para a classe das linguagens recursivamente enumeráveis.

A simplicidade com que a indecibilidade do problema da parada pode ser obtida no teorema 9.2.5 se deve a que ambos problemas são muito semelhantes. A principal diferença é que enquanto no problema de pertinência distinguimos entre parar num estado final e parar num estado não-final, já no problema da parada não fazemos tal distinção. As provas dos teoremas 9.2.5 e 11.2.2 estão estreitamente relacionados, ambos são uma versão da aplicação do método da diagonalização de Cantor.

## 11.3 Redução de um Problema Indecidível ao Problema da Parada

O argumento usado no teorema 11.2.3 para conectar o problema da parada com o problema da pertinência para linguagens recursivamente enumeráveis, ilustra uma técnica de redução muito importante, que pode ser utilizada para demonstrar a indecibilidade de outros problemas de decisão. Dizemos que um problema  $A$  é **reduzido** a um problema  $B$  se a decidibilidade de  $A$  acarreta a decidibilidade de  $B$ . A idéia é expressar o problema  $B$  em termos do problema  $A$  e de problemas conhecidos como decidíveis. Assim, se soubermos que  $A$  é decidível, podemos concluir que  $B$  também é decidível. Logo, por contraposição, se soubermos que  $B$  é indecidível então necessariamente  $A$  é indecidível. Vamos ver alguns exemplos para ilustrar essa idéia.

**Exemplo 11.3.1 O problema da entrada em um estado** é o seguinte. Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  uma máquina de Turing qualquer,  $q \in Q$  e  $w \in \Sigma^+$ . Decidir se  $M$  entrará ou não no estado  $q$  quando  $M$  é aplicado a  $w$ . Este problema é indecidível.

Para reduzir o problema da entrada em um estado ao problema da parada, suponha que temos um algoritmo  $A$  que resolve o problema da entrada de um estado. Poderíamos, então, usá-lo para resolver o problema da parada. Por exemplo, dado qualquer  $M$  e  $w$ , primeiro modificamos  $M$  para obter  $\widehat{M}$  de tal maneira que  $\widehat{F} = \{q\}$  e  $\widehat{M}$  páre no estado  $q$  quando  $M$  é aplicado a  $w$  se e somente se  $M$  pára (em qualquer estado) quando é aplicado a  $w$ . Podemos fazer isso, simplesmente, olhando a função de transição  $\delta$ . Note que  $M$  parará somente se  $\delta(q_i, a)$  é indefinido. Para obter  $\widehat{M}$  trocamos este tipo de produção por

$$\delta(q_i, a) = (q, a, D).$$

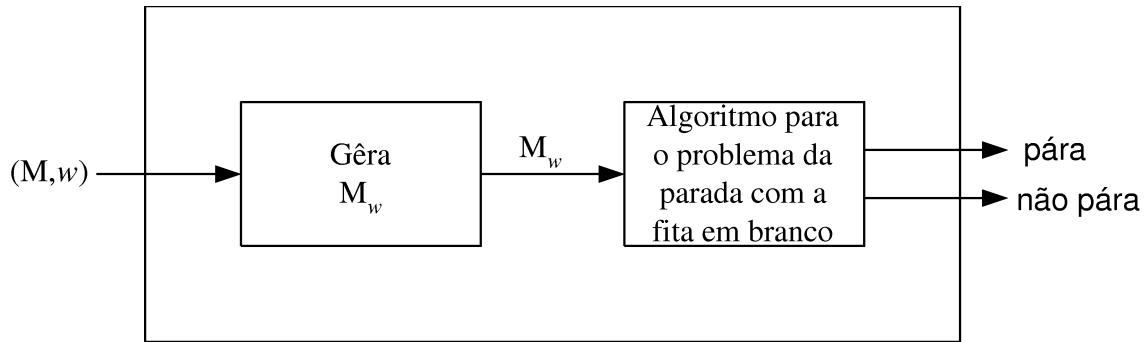


Figura 11.3: Algoritmo para o problema da parada usando um eventual algoritmo para o problema da parada com a fita em branco.

Aplicando o algoritmo da entrada de um estado  $A$  a  $(\widehat{M}, q, w)$ , teremos que  $A$  responderá sim se  $\widehat{M}$  entrar no estado  $q$ , e nesse caso  $(M, w)$  pararia. Se  $A$  responder não, então  $(M, w)$  não pararia, o qual constituiria uma solução para o problema da parada.

Portanto a hipótese de que o problema da entrada de um estado é decidível levaria a um algoritmo para o problema da parada. Como o problema da parada é indecidível, o problema da entrada de um estado deve ser indecidível.

**Exemplo 11.3.2 O problema da parada com a fita em branco** é um outro problema que pode ser reduzido ao problema da parada. Dado uma máquina de Turing  $M$ , determine se  $M$  pára ou não quando iniciado com a fita em branco. Este problema é indecidível.

Para provar isso, tome qualquer máquina de Turing  $M$  e qualquer  $w$ . Primeiro, construiremos a partir de  $M$  uma nova máquina,  $M_w$ , que inicia com a fita em branco, escreve  $w$  nela, e então se posiciona na configuração  $q_0w$ . Após isso,  $M_w$  se comporta exatamente como  $M$ . É claro que  $M_w$  parará sobre uma fita em branco se e somente se  $M$  pára quando iniciado com  $w$ .

Suponha, agora, que o problema da parada da fita em branco seja decidível. Dado qualquer  $(M, w)$ , primeiro construímos  $M_w$ , então aplicamos o algoritmo da parada da fita em branco a ele. Como isso pode ser feito para qualquer  $M$  e  $w$ , então o algoritmo para o problema da fita em branco pode ser usado para obter um algoritmo para o problema da parada. Como este último é indecidível, então por contraposição, o problema da parada para a fita em branco também é indecidível.

Esses dois exemplos apresentam a mesma construção para estabelecer resultados de indecidibilidade de um problema de decisão, que consiste em escrever uma eventual solução para o problema da parada em função da solução do problema abordado. Um diagrama de bloco pode ajudar a visualizar este processo. A construção no exemplo 11.3.2 é resumida na figura 11.3. Naquele diagrama, primeiro usamos um algoritmo que transforma  $(M, w)$  em  $M_w$ . Tal algoritmo claramente existe. Em seguida usamos o algoritmo para resolver o problema da parada para a fita em branco,  $A$ , que assumimos existir. Pondo os dois juntos obteremos um algoritmo para o problema da parada, mas isso é impossível, concluindo que  $A$  não existe.

## 11.4. Problemas Indecidíveis para Linguagens Recursivamente Enumeráveis

---

Um problema de decisão admite como possíveis soluções funções com valores de saída em  $\{0, 1\}$ , isto é, verdadeiro ou falso. Podemos olhar também para funções mais gerais para ver se são computáveis. Redução ao problema da parada é, também, adequado aqui. Devido à tese de Turing, esperamos que funções encontradas em circunstâncias práticas sejam computáveis. Assim, para achar exemplos de funções não computáveis deveremos ir um pouco além. Muitos exemplos de funções não computáveis estão associadas à tentativa de prever o comportamento de uma máquina de Turing.

**Exemplo 11.3.3** Seja  $\Gamma = \{0, 1, \square\}$ . Seja  $\mathcal{M}_\Gamma^n$  o conjunto das máquinas de Turing com  $n$  estados, tendo  $\Gamma$  como alfabeto da fita e que pára quando inicia com uma fita em branco. Considere a função  $f(n)$  cujo valor é o número máximo de movimentos que pode ser feito por qualquer máquina de Turing em  $\mathcal{M}_\Gamma^n$  quando iniciada com a fita em branco. Esta função, como não poderia deixar de ser, não é computável.

Antes de provarmos a afirmação, observe que por  $Q$  e  $\Gamma$  serem finitos, a menos de nomes dos estados, somente há uma quantidade finita de possíveis  $\delta$ 's em  $\mathcal{M}_\Gamma^{|Q|}$  e consequentemente esse conjunto é finito. Logo, por definição, toda máquina de Turing em  $\mathcal{M}_\Gamma^{|Q|}$  executará um certo número de movimentos quando iniciada com a fita em branco. Desses tomamos o maior para fornecer  $f(n)$ . Logo  $f$  está bem definido.

Tome qualquer máquina de Turing  $M$  e um inteiro positivo  $m$ . É fácil modificar  $M$  para produzir  $\widehat{M}$  de tal maneira que esta última sempre parará com uma das duas respostas:  $M$  aplicada a uma fita em branco pára em não mais do que  $m$  movimentos, ou  $M$  aplicada a uma fita em branco faz mais do que  $m$  movimentos. Tudo que temos de fazer para isso é tomar  $M$ , contar seus movimentos e terminar quando este número exceder  $m$ . Assuma, agora, que  $f(n)$  é computável por alguma máquina de Turing  $F$ . Podemos, então, juntar  $\widehat{M}$  e  $F$  como é mostrado na figura 11.4. Primeiro computamos  $f(|Q|)$ , onde  $Q$  é o conjunto de estados de  $M$ . Isto nos diz o número máximo de movimentos que  $M$  pode fazer quando iniciada com a fita em branco se parar. O valor obtido é, então, usado como  $m$  para construir  $\widehat{M}$ , como esboçado acima. Uma descrição de  $\widehat{M}$  é dada a uma máquina de Turing universal para execução. Isto nos diz se  $M$  aplicado a uma fita em branco pára ou não em menos de  $f(|Q|)$  etapas. Se acharmos que  $M$  aplicada a uma fita em branco faz mais do que  $f(|Q|)$  movimentos, então, devido à definição de  $f$ , a implicação é que  $M$  nunca pára. Portanto, teríamos uma solução para o problema da parada para a fita em branco, o qual é impossível. Logo,  $f$  não é computável.

## 11.4 Problemas Indecidíveis para Linguagens Recursivamente Enumeráveis

O corolário 11.2.4 mostra a não existência de um algoritmo de pertinência para a classe das linguagens recursivamente enumeráveis. Esta classe de linguagem é tão geral que, praticamente, qualquer problema de decisão sobre a classe como um todo é indecidível. A demonstração de resultados específicos são feitas aplicando uma redução ao problema da parada ou a uma de suas consequências imediatas. Apresentaremos aqui alguns exemplos que ilustram como isso pode ser feito e a partir daí esboçar uma indicação para uma situação geral.

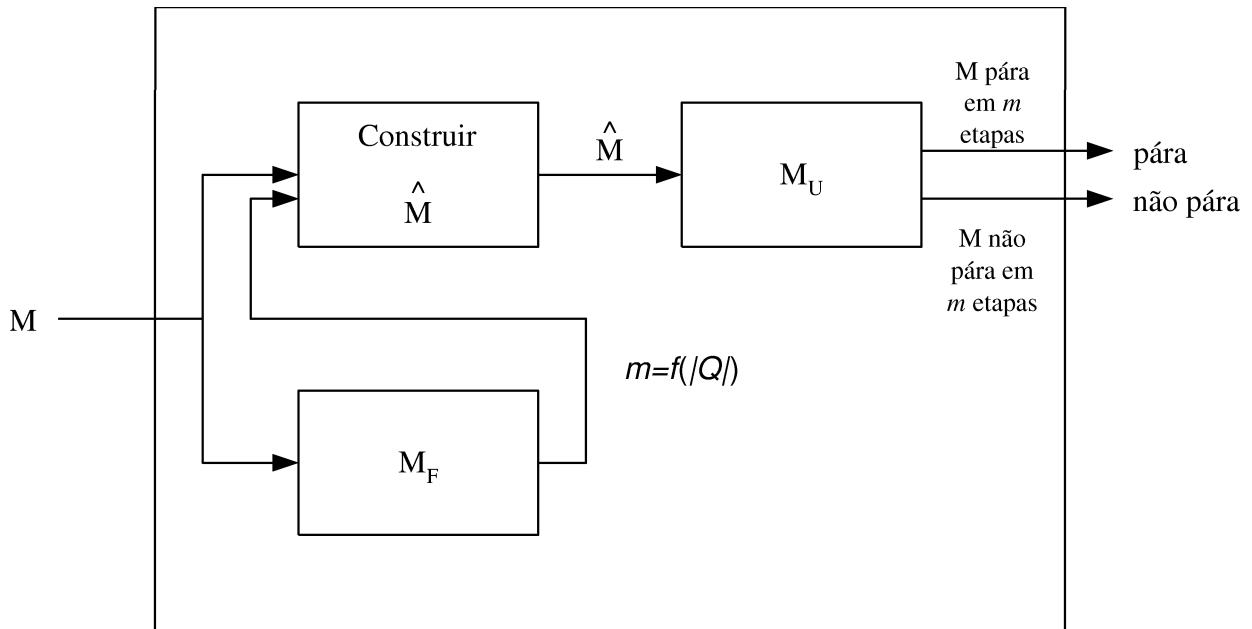


Figura 11.4: Algoritmo para o problema da parada com a fita em branco baseado num algoritmo para computar  $f(n)$ .

**Teorema 11.4.1** Seja  $M$  uma máquina de Turing. Então o problema de determinar se

$$L(M) = \emptyset$$

ou não é indecidível.

**DEMONSTRAÇÃO:** Reduziremos esse problema ao problema da pertinência para linguagens recursivamente enumeráveis. Suponha que é dado uma máquina de Turing,  $M$ , e uma cadeia  $w$ . Podemos construir uma nova máquina de Turing  $M_w$  a partir de  $M$  e  $w$  como segue. Primeiro verifica se a entrada é igual com  $w$ , se não for então pára em um estado não final se for então deixa  $w$  na fita e se posiciona no símbolo mais à esquerda de  $w$  para então se comportar como  $M$ . Assim, claramente,

$$L(M_w) = L(M) \cap \{w\},$$

e portanto  $L(M_w)$  é não vazio se, e somente se,  $w \in L(M)$ .

Suponha, agora, que existe um algoritmo  $A$  para decidir se  $L(M) = \emptyset$  ou não. Assim, compondo um algoritmo por meio do qual geramos  $M_w$  ao algoritmo  $A$ , como mostrado na figura 11.5, teríamos uma máquina de Turing que para qualquer máquina de Turing  $M$  e entrada  $w$  nos diria se  $w \in L(M)$  ou não. Se tal máquina de Turing existisse teríamos um algoritmo de pertinência para qualquer linguagem recursivamente enumerável, resultando em uma contradição com o resultado estabelecido anteriormente. Concluímos, portanto, que o problema de se “ $L(M) = \emptyset$  ou não” não é decidível. ■

## 11.4. Problemas Indecidíveis para Linguagens Recursivamente Enumeráveis

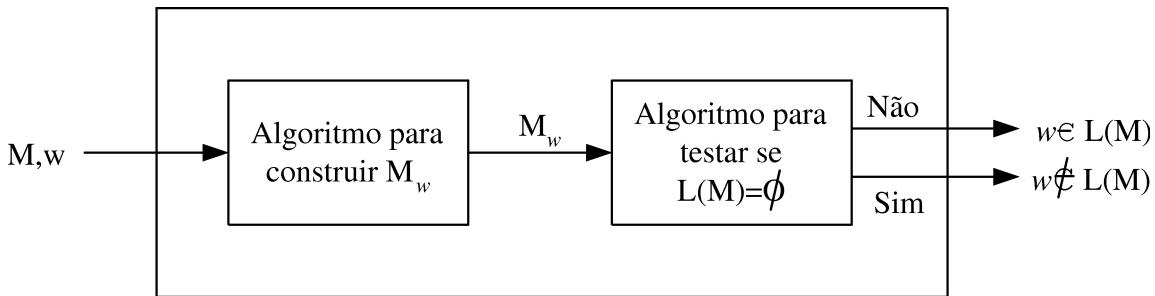


Figura 11.5: Algoritmo de pertinência para linguagens recursivamente enumeráveis.

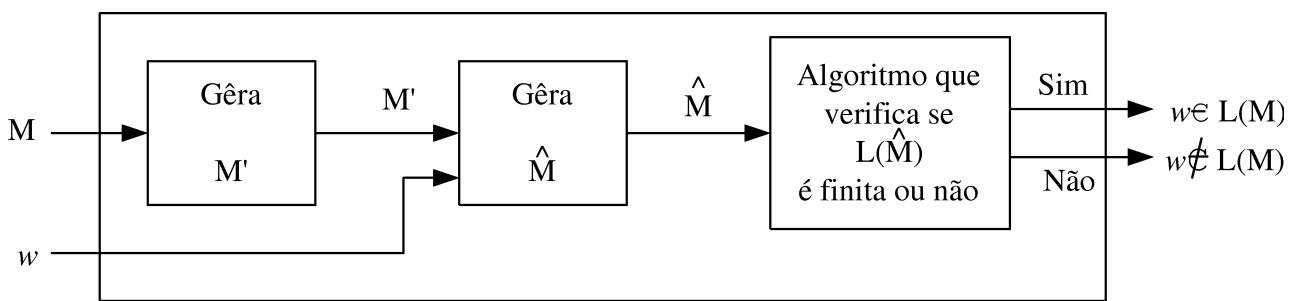


Figura 11.6: Uma solução para o problema da parada baseada num algoritmo que diz se uma linguagem é finita ou não.

**Teorema 11.4.2** Seja  $M$  uma máquina de Turing qualquer. A questão de se  $L(M)$  é finita ou não é indecidível.

**DEMONSTRAÇÃO:** Seja  $M$  uma máquina de Turing qualquer e  $w$  uma cadeia de entrada. De  $M$  construiremos uma outra máquina de Turing,  $M'$ , trocando os estados de parada de  $M$  de tal modo que se qualquer um é atingido, vai para um estado final. Claramente  $M'$  aceita toda cadeia para a qual  $M$  pára em algum estado (final ou não). A partir de  $M'$  construímos uma outra máquina de Turing  $\widehat{M}$ , a qual primeiro apaga o que tiver na fita, gera  $w$  e se posiciona no símbolo mais à esquerda de  $w$  para depois se comportar como  $M'$ . Em outras palavras, os movimentos feitos por  $\widehat{M}$  após ela escrever  $w$  na fita são os mesmos que teriam sido feitos por  $M'$  se ela tivesse começado na configuração original  $q_0w$ . Logo, se  $M$  com  $w$  na fita de entrada parar em algum estado, então  $\widehat{M}$  atingirá um estado final para todas as entradas. Se  $M$  com  $w$  na fita de entrada não parar, então  $\widehat{M}$ , também, não parará e portanto não aceitará qualquer cadeia. Em outras palavras,  $\widehat{M}$  ou aceita a linguagem infinita  $\Sigma^+$  ou a linguagem finita  $\emptyset$ .

Se, agora, assumimos a existência de um algoritmo,  $A$ , que nos diz se  $L(M)$  é finita ou não, podemos construir uma solução para o problema da parada como é mostrado na figura 11.6. Logo, não existe algoritmo para decidir se  $L(M)$  é finito ou não. ■

Note que nas figuras 11.5 e 11.6 a pergunta, no caso de “ $L(M) = \emptyset?$ ” e “ $L(M)$  é finito?”, não é fundamental, ou seja se substituirmos por outra pergunta do mesmo tipo, não mudaremos significativamente o argumento de prova. Logo podemos colocar outras questões tais como “ $L(M)$

contém qualquer cadeia de comprimento 5?” ou “ $L(M)$  é regular?” sem afetar essencialmente o argumento. Essas questões, assim como questões similares, são todas indecidíveis. Um resultado geral formalizando este fato é conhecido como **teorema de Rice**.

**Teorema 11.4.3 (de Rice)** *Qualquer propriedade não trivial sobre  $\mathcal{L}_{RE}$  é indecidível.*

DEMONSTRAÇÃO: Vêr [HU79, Teorema 8.6]. ■

## 11.5 O Problema da Correspondência de Post

As consequências de interesse prático que resultam da indecibilidade do problema da parada são das mais variadas, em particular, na classe das linguagens livres de contexto. No entanto, em diversas situações não é possível usar de forma natural e direta o problema da parada. Assim, é conveniente estabelecer alguns resultados intermediários que preencha a distância que existe entre o problema da parada e outros problemas. Embora esses resultados intermediários seguem da indecibilidade do problema da parada, estão mais próximos dos problemas que pretendemos abordar, tornando assim os argumentos mais simples. Um desses resultados intermediários é o **problema da correspondência de Post**.

O problema da correspondência de Post pode ser enunciado como segue. Sejam

$$A = w_1, w_2, \dots, w_n \quad \text{e} \quad B = v_1, v_2, \dots, v_n,$$

duas seqüências de  $n$  cadeias sobre algum alfabeto  $\Sigma$ . Existe uma **solução da correspondência de Post (solução-CP)** para o par  $(A, B)$  se houver uma seqüência não vazia de inteiros  $i, j, \dots, k$  tal que

$$w_i w_j \dots w_k = v_i v_j \dots v_k.$$

O problema da correspondência de Post é fornecer um algoritmo que nos diga se existe ou não uma solução-CP para qualquer  $(A, B)$ .

**Exemplo 11.5.1** Seja  $\Sigma = \{0, 1\}$  e tome  $A$  e  $B$  como

$$w_1 = 11, \quad w_2 = 100, \quad w_3 = 111$$

$$v_1 = 111, \quad v_2 = 001, \quad v_3 = 11.$$

A figura 11.7 apresenta uma solução-CP para esse par  $(A, B)$ .

Já, se  $A$  for a seqüência

$$w_1 = 00, \quad w_2 = 001, \quad w_3 = 1000$$

## 11.5. O Problema da Correspondência de Post

---

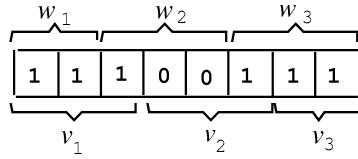


Figura 11.7: Exemplo de solução-CP

e  $B$  a seqüência

$$v_1 = 0, \quad v_2 = 11, \quad v_3 = 001,$$

então não existirá qualquer solução-CP para  $(A, B)$ , pois qualquer cadeia composta de elementos de  $A$  sempre será maior do que a cadeia correspondente de  $B$ .

Para instâncias específicas de pares  $(A, B)$  podemos ser capazes, caso  $(A, B)$  tenha solução-CP, de fornecer uma tal seqüência de cadeia, ou, caso  $(A, B)$  não tenha solução-CP, de argumentar, como fizemos acima, que uma tal seqüência não existe. No entanto em geral, não existe um algoritmo capaz de decidir essa questão para todas as circunstâncias possíveis. Portanto o problema da correspondência de Post é indecidível.

Uma demonstração mais formal da indecibilidade do problema da correspondência de Post é longa, pelo que quebraremos ela em partes. Primeiro, introduzimos o **problema da correspondência de Post modificado**. Depois mostramos que o problema da correspondência de Post modificado para depois reduzir o problema da correspondência de Post ao problema da correspondência de Post modificado.

Dizemos que o par  $(A, B)$  admite uma solução do problema da correspondência de Post modificado (solução-CPM) se existir uma seqüência de inteiros  $i, j, \dots, k$  tal que

$$w_1 w_i w_j \dots w_k = v_1 v_i v_j \dots v_k.$$

No problema da correspondência de Post modificado, os primeiros elementos da seqüência  $A$  e  $B$  desempenham um papel especial, uma vez que qualquer solução deve começar com eles. Observe que toda solução-CPM para um par  $(A, B)$  é também uma solução-CP, mas a recíproca não é verdadeira.

O problema da correspondência de Post modificado consiste em fornecer um algoritmo para decidir se um par arbitrário  $(A, B)$  admite ou não uma solução-CPM. Este problema também é indecidível e demonstraremos sua indecibilidade reduzindo-o ao já conhecido problema da pertinência de linguagens recursivamente enumeráveis, o qual já foi provado ser um problema indecidível.

Antes porém, introduzimos a seguinte construção a partir de uma gramática irrestrita,  $G = \langle V, T, S, P \rangle$ , e uma cadeia alvo  $w$ . Seja o par  $(A, B)$  apresentado na tabela 11.1.

Nesta tabela, as cadeias  $FS \implies$  e  $F$  devem ser atribuídas a  $w_1$  e  $v_1$ , respectivamente. A ordem das outras cadeias é irrelevantes.

$A$	$B$	
$FS \Rightarrow$	$F$	$F$ é um símbolo não em $V \cup T$
$a$	$a$	para todo $a \in T$
$V_i$	$V_i$	para todo $V_i \in V$
$E$	$\Rightarrow wE$	$E$ é um símbolo não em $V \cup T$
$y_i$	$x_i$	para todo $x_i \rightarrow y_i$ em $P$
$\Rightarrow$	$\Rightarrow$	

Tabela 11.1: Uma solução-CPM genérica para gramáticas irrestritas.

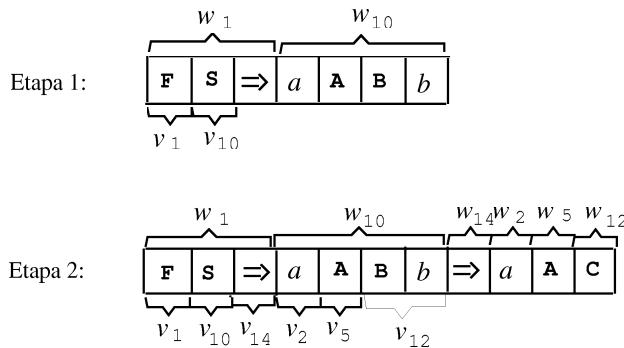


Figura 11.8: Exemplo de solução-CPM (parcial)

Mais adiante alegaremos que  $w \in L(G)$  se, e somente se, os conjuntos  $A$  e  $B$  construídos dessa maneira têm uma solução-CPM. Como isso não é imediatamente óbvio, vamos ilustrá-lo com um exemplo simples.

**Exemplo 11.5.2** Seja  $G = \langle \{S, A, B, C\}, \{a, b, c\}, S, P \rangle$  uma gramática irrestrita com produções

$$S \rightarrow aABb \mid Bbb,$$

$$Bb \rightarrow C,$$

$$AC \rightarrow aac.$$

Considere como cadeia alvo  $w = aaac$ . As seqüências  $A$  e  $B$  obtidas da construção sugerida na tabela 11.1 são apresentadas na tabela 11.2. A cadeia  $w = aaac$  está em  $L(G)$  e pode ser derivada como segue:

$$S \Rightarrow aABb \Rightarrow aAC \Leftrightarrow aaac.$$

A figura 11.8 ilustra as duas primeiras etapas de uma possível solução-CPM baseada na derivação acima. As chaves acima e abaixo das cadeias de derivação mostram os  $w_i$ 's e  $v_i$ 's, respectivamente, usados para criar a cadeia.

### 11.5. O Problema da Correspondência de Post

$i$	$w_i$	$v_i$
1	$FS \implies$	$F$
2	$a$	$a$
3	$b$	$b$
4	$c$	$c$
5	$A$	$A$
6	$B$	$B$
7	$C$	$C$
8	$S$	$S$
9	$E$	$\implies aaacE$
10	$aABb$	$S$
11	$Bbb$	$S$
12	$C$	$Bb$
13	$aac$	$AC$
14	$\implies$	$\implies$

Tabela 11.2: Exemplo de solução-CPM para uma gramática irrestrita.

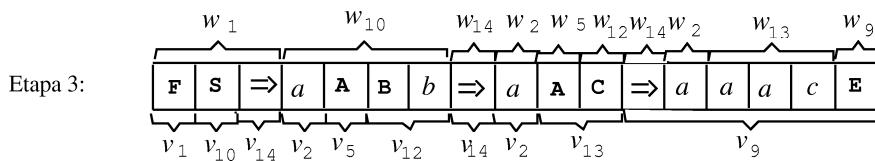


Figura 11.9: Exemplo de solução-CPM (continuação)

Examine cuidadosamente a figura 11.8 para ver o que acontece. Queremos construir uma solução-CPM, portanto devemos começar com  $w_1$ , isto é,  $FS \Rightarrow$ . Esta cadeia contém  $S$ . Portanto para casá-la temos de usar  $v_{10}$  ou  $v_{11}$ . Nesta instância, usamos  $v_{10}$ , pois ela foi obtida a partir da produção usada na derivação de  $aaac$  usada como referência. A cadeia em  $w_{10}$  leva-nos a uma segunda cadeia na derivação parcial. Olhando várias etapas na frente, vemos que a cadeia  $w_1w_iw_j\dots$  é sempre mais comprida do que a cadeia correspondente  $v_1v_iv_j\dots$  e a primeira está exatamente uma etapa à frente na derivação. A figura 11.9 apresenta a solução-CPM completa para este exemplo. Esta construção indica qual o caminho que devemos seguir para se chegar ao próximo resultado.

**Teorema 11.5.3** Seja  $G = \langle V, T, S, P \rangle$  uma gramática irrestrita e  $w \in T^*$ . Seja  $(A, B)$  o par correspondente construído de  $G$  e  $w$  pelo processo exibido na tabela 11.1. Então o par  $(A, B)$  admite uma solução-CPM se, e somente se,  $w \in L(G)$ .

DEMONSTRAÇÃO: (omitida) O exemplo acima dá uma idéia da demonstração.

**Teorema 11.5.4** O problema da correspondência de Post modificado é indecidível.

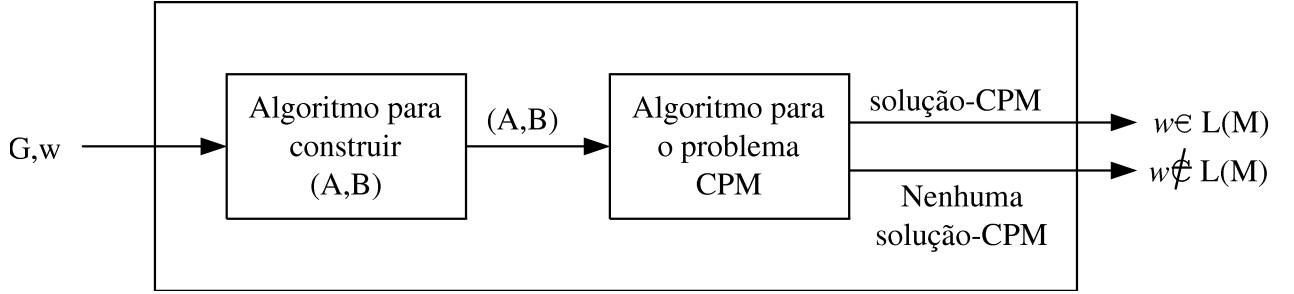


Figura 11.10: Algoritmo de pertinência para uma gramática irrestrita.

**DEMONSTRAÇÃO:** Dada uma gramática irrestrita  $G = \langle V, T, S, P \rangle$  e  $w \in T^*$ , construímos os conjuntos  $A$  e  $B$  como sugerido acima. Pelo teorema 11.5.3, o par  $(A, B)$  tem uma solução-CPM se, e somente se,  $w \in L(G)$ .

Suponha, agora, que o problema da correspondência de Post modificado é decidível. Podemos, então, construir um algoritmo para o problema da pertinência para  $G$  como esboçado no diagrama de blocos da figura 11.10. Um algoritmo para construir  $A$  e  $B$  a partir de  $G$  e  $w$  claramente existe, mas um algoritmo de pertinência para  $G$  e  $w$  não existe. Podemos, portanto, concluir que não existe um algoritmo para decidir o problema da correspondência de Post modificado. ■

**Teorema 11.5.5** *O problema da correspondência de Post é indecidível.*

**DEMONSTRAÇÃO:** Mostraremos que se o problema da correspondência de Post fosse decidível, então o problema da correspondência de Post modificado também seria decidível.

Sejam as seqüências  $A = w_1, w_2, \dots, w_n$  e  $B = v_1, v_2, \dots, v_n$ , sobre algum alfabeto  $\Sigma$  e dois novos símbolos  $\flat$  e  $\#$ . Defina as duas novas seqüências

$$C = y_0, y_1, \dots, y_{n+1}, \quad D = z_0, z_1, \dots, z_{n+1},$$

onde, para cada  $i = 1, 2, \dots, n$

$$y_i = w_{i_1} \flat w_{i_2} \flat \cdots \flat w_{i_{m_i}} \flat, \quad z_i = \flat v_{i_1} \flat v_{i_2} \flat \cdots \flat v_{i_{r_i}},$$

com  $w_{i_j}$  e  $v_{i_j}$  denotando a  $j$ -ésima letra de  $w_i$  e  $v_i$ , respectivamente, e  $m_i = |w_i|$ ,  $r_i = |v_i|$ . Ou seja,  $y_i$  é criado de  $w_i$  anexando  $\flat$  após cada caracter de  $w_i$ , enquanto  $z_i$  é obtido prefixando  $\flat$  a cada caracter de  $v_i$ . Para completar a definição de  $C$  e  $D$ , tomamos

$$\begin{aligned} y_0 &= \flat y_1 \\ y_{n+1} &= \# \\ z_0 &= z_1 \\ z_{n+1} &= \flat \# \end{aligned}$$

## 11.6. Problemas Indecidíveis para Linguagens Livres do Contexto

---

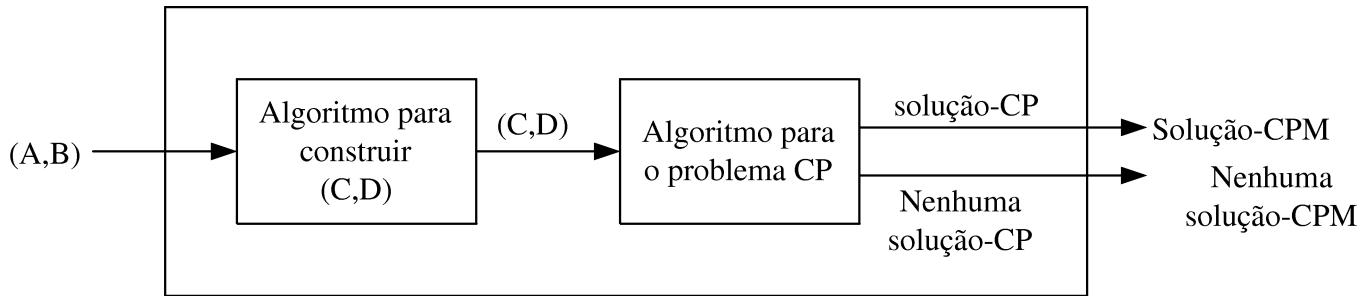


Figura 11.11: Algoritmo CPM.

Considere, agora, o par  $(C, D)$ , e suponha que ele tem uma solução-CP. Por causa da troca de  $\flat$  e  $\#$ , tal solução deve ter  $y_0$  no lado esquerdo e  $y_{n+1}$  no lado direito. Portanto essa solução deveria ser do tipo:

$$\flat w_{1_1} \flat w_{1_2} \flat \dots \flat w_{j_1} \flat \dots \flat w_{k_1} \flat \dots \flat \# = \flat v_{1_1} \flat v_{1_2} \flat \dots \flat v_{j_1} \flat \dots \flat v_{k_1} \flat \dots \flat \#$$

Ignorando os caracteres  $\flat$  e  $\#$ , vemos que isto implicaria que

$$w_1 w_j \dots w_k = v_1 v_j \dots v_k.$$

Portanto o par  $(A, B)$  nesse caso admitiria uma solução-CPM.

Assim, podemos modificar esse argumento para mostrar que se existe uma solução-CP para  $(C, D)$  então existe uma solução-CPM para o par  $(A, B)$ .

Logo, se o problema da correspondência de Post fosse decidível poderíamos, então, construir a máquina mostrada na figura 11.11. Esta máquina, claramente, decide o problema da correspondência de Post modificado. Mas este problema é indecidível, consequentemente não podemos ter um algoritmo para decidir o problema da correspondência de Post. ■

## 11.6 Problemas Indecidíveis para Linguagens Livres do Contexto

O problema da correspondência de Post pode ser utilizado para demonstrar a indecibilidade de alguns problemas para a classe das linguagens livres do contexto. Nesta seção mostraremos alguns desses problemas.

**Teorema 11.6.1** *Não existe um algoritmo para decidir se uma dada gramática livre do contexto é ambígua ou não.*

**DEMONSTRAÇÃO:** Considere duas seqüências arbitrárias  $A = (w_1, w_2, \dots, w_n)$  e  $B = (v_1, v_2, \dots, v_n)$ , sobre algum alfabeto  $\Sigma$ . Sejam  $n$  novos símbolos,  $a_1, a_2, \dots, a_n$ , isto é

$$\{a_1, a_2, \dots, a_n\} \cap \Sigma = \emptyset,$$

e as linguagens

$$\mathcal{L}_A = \{w_{i_1} \dots w_{i_k} a_{i_k} \dots a_{i_1} / \text{ para todo } j = 1, \dots, k, \text{ com } k \in \mathbb{N}, i_j \in \{1, \dots, n\}\}$$

e

$$\mathcal{L}_B = \{v_{i_1} \dots v_{i_k} a_{i_k} \dots a_{i_1} / \text{ para todo } j = 1, \dots, k, \text{ com } k \in \mathbb{N}, i_j \in \{1, \dots, n\}\}.$$

Defina  $P_A$  como sendo composto pelas produções:

$$S \longrightarrow S_A,$$

$$S_A \longrightarrow w_i S_A a_i \mid w_i a_i, i = 1, 2, \dots, n,$$

e  $P_B$  pelas produções

$$S \longrightarrow S_B,$$

$$S_B \longrightarrow v_i S_B a_i \mid v_i a_i, i = 1, 2, \dots, n.$$

Assim, as gramáticas livre do contextos

$$G_A = \langle \{S, S_A\}, \Sigma \cup \{a_1, a_2, \dots, a_n\}, S, P_A \rangle,$$

$$G_B = \langle \{S, S_B\}, \Sigma \cup \{a_1, a_2, \dots, a_n\}, S, P_B \rangle,$$

e

$$G = \langle \{S, S_A, S_B\}, \Sigma \cup \{a_1, a_2, \dots, a_n\}, S, P_A \cup P_B \rangle,$$

são tais que:  $\mathcal{L}_A = L(G_A)$ ,  $\mathcal{L}_B = L(G_B)$  e  $L(G) = \mathcal{L}_A \cup \mathcal{L}_B$ .

Se uma dada cadeia  $w \in L(G)$  então ou  $w \in L(G_A)$ , ou  $w \in L(G_B)$  ou ambos. Se  $w$  termina com  $a_i$  e  $w \in L(G_A)$ , então sua derivação com a gramática  $G_A$  deve ter início com  $S \implies S_A \implies w_i S_A a_i$ , já se  $w \in L(G_B)$ , então sua derivação com a gramática  $G_B$  deve ter início com  $S \implies S_B \implies v_i S_B a_i$ . Assim em qualquer estágio posterior, podemos dizer de forma análoga quais regras devem ser aplicadas. Portanto, se  $G$  for ambígua, então necessariamente existirá um  $w$  para o qual existem duas derivações.

$$S \implies S_A \implies w_i S_A \xrightarrow{*} w_i w_j \dots w_k a_k \dots a_j a_i = w$$

e

## 11.6. Problemas Indecidíveis para Linguagens Livres do Contexto

---

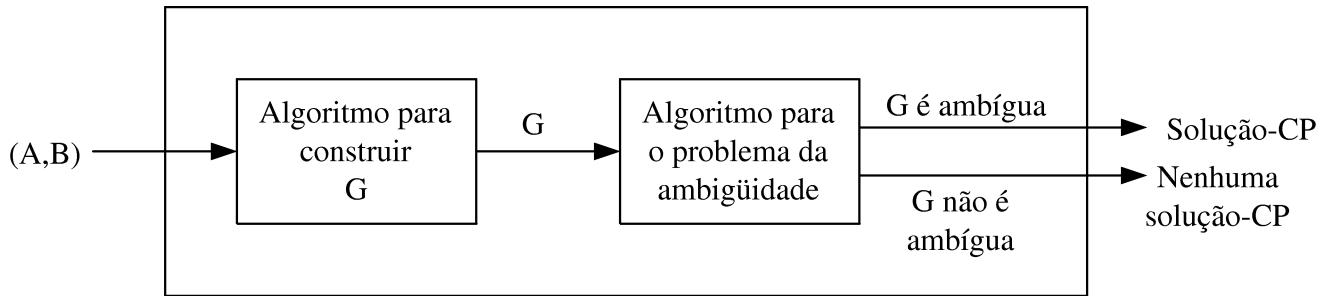


Figura 11.12: Algoritmo CP.

$$S \Rightarrow S_B \Rightarrow v_i S_B \xrightarrow{*} v_i v_j \dots v_k a_k \dots a_j a_i = w.$$

Conseqüentemente, se  $G$  for ambígua, o problema da correspondência de Post com o par  $(A, B)$  teria uma solução. Inversamente, se  $G$  não for ambígua, então o problema da correspondência de Post não teria solução.

Assim, se existisse um algoritmo para resolver a Ambigüidade, poderíamos adaptá-lo, como mostra a figura 11.12, para resolver o problema da correspondência de Post. Mas, como não existe um algoritmo para resolver o problema da correspondência de Post, concluímos que o problema da Ambigüidade é indecidível. ■

**Teorema 11.6.2** *Não existe algoritmo para decidir se duas gramáticas livres de contexto arbitrárias,  $G_1$  e  $G_2$ , geram uma cadeia em comum ou não, isto é, se*

$$L(G_1) \cap L(G_2) \neq \emptyset$$

*ou não.*

**DEMONSTRAÇÃO:** Considere  $G_1$  como sendo a gramática  $G_A$  e  $G_2$  como sendo a gramática  $G_B$  da prova do teorema 11.6.1. Suponha que  $L(G_A)$  e  $L(G_B)$  têm um elemento em comum, isto é,

$$S_A \xrightarrow{*} w_{i_1} \dots w_{i_k} a_{i_k} \dots a_{i_1}$$

e

$$S_B \xrightarrow{*} v_{i_1} \dots v_{i_k} a_{i_k} \dots a_{i_1}.$$

Então o par  $(A, B)$  teria uma solução-CP. Inversamente, se o par não tem uma solução-CP, então  $L(G_A)$  e  $L(G_B)$  não podem ter um elemento em comum. Portanto,  $L(G_A) \cap L(G_B) \neq \emptyset$  se, e somente se,  $(A, B)$  tem uma solução-CP. ■

Existe uma variedade de outros resultados conhecidos nessa mesma linha. Alguns deles podem ser reduzidos ao problema da correspondência de Post, enquanto outros são mais fáceis de serem resolvidos usando-se alguns resultados intermediários.

A existência dessa variedade de problemas indecidíveis para a classe das linguagens livres de contextos pode parecer num princípio surpreendente além de deixar em evidência limitações para computações em áreas nas quais tentamos uma abordagem algorítmica. Por exemplo, seria interessante que pudéssemos dizer se uma linguagem de programação definida em BNF é ambígua ou não, ou se duas diferentes especificações de uma linguagem são, realmente, equivalentes. Mas os resultados que estabelecemos nos dizem que isso não é possível. Seria uma perda de tempo procurar um algoritmo para essas tarefas. É bom ter em mente que isto não afasta a possibilidade de que haja maneiras de obter respostas para problemas específicos. O que o resultado da indecibilidade nos diz é que não existe um algoritmo geral, e que qualquer solução que obtivermos devemos usar a estrutura específica do problema.

### 11.7 Semi-decibilidade, Co-semi-decibilidade e Divergência

Problemas redutíveis ao problema da parada têm como característica o fato de existir uma solução parcial para ele, ou seja uma solução que é capaz de responder com segurança sempre que a resposta for sim ou sempre que for não. Por exemplo para o próprio problema da parada a máquina de Turing universal seria capaz de responder sempre “sim”, quando a máquina de Turing  $M$  parar para  $w$ . Um problema de decisão para um conjunto  $A$  é **semi-decidível** se existe uma máquina de Turing capaz de responder afirmativamente quando um elemento pertence ao conjunto  $A$  embora nem sempre consiga responder negativamente quando o elemento não pertence ao conjunto  $A$ . Assim, trivialmente todo problema decidível é semi-decidível. Analogamente, um problema de decisão para um conjunto  $A$  é **co-semi-decidível** se existe uma máquina de Turing capaz de responder negativamente quando for o caso que  $a \notin A$ , embora nem sempre consiga responder positivamente quando  $a \in A$ . Assim, claramente um problema de decisão para um conjunto  $A$  é semi-decidível se, e somente se, o problema de decisão para  $\bar{A}$  é co-semi-decidível. Observe ainda que um problema de decisão é semi e co-semi-decidível se, e somente se, ele é decidível. Mas, isto não significa que se  $A$  for semi-decidível e  $\bar{A}$  co-semi-decidível, implique que  $A$  é decidível.

Claramente existem problemas que são semi-decidíveis ou co-semi-decidíveis que não são decidíveis. Por exemplo, o problema da parada é semi-decidível, e como vimos neste capítulo, não é decidível. Obviamente, o seu complemento, também conhecido como **problema da divergência**, é co-semi-decidível mas não é decidível. Mas será que há problemas que sejam completamente indecidíveis? ou seja que não sejam nem semi ou co-semi-decidíveis? A resposta é sim. Um exemplo de um problema completamente indecidível é o **problema da parada duplo**, enunciado a seguir:

*dada a descrição de duas máquinas de Turing sobre um alfabeto  $\Sigma$ ,  $M_1$  e  $M_2$ , e uma entrada  $w \in \Sigma^*$ , quando iniciadas na configuração inicial  $q_0w$ ,  $M_1$  efetua uma computação que pára enquanto  $M_2$  efetua uma computação que não pára?*

## 11.8. Exercícios

---

### 11.8 Exercícios

1. Mostre que os seguintes problemas são indecidíveis:
  - (a) Dada uma máquina de Turing  $M$  arbitrária, decidir se a máquina de Turing  $M$  computa uma função constante ou não.
  - (b) Dada uma máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  arbitrária e um  $w_s \in \Gamma^*$  arbitrário, decidir se existe ou não uma entrada  $w_e \in \Sigma^*$  tal que a máquina de Turing  $M$  quando iniciada no estado  $q_0$  e com  $w_e$  na fita de entrada pára num estado final com  $w_s$  na fita.
  - (c) Mostre que o seguinte problema é indecidível. Dada qualquer máquina de Turing  $M$ ,  $a \in \Gamma$ , e  $w \in \Sigma^+$ , determinar quando ou não o símbolo  $a$  é ou não escrito na fita, quando  $M$  é aplicado a  $w$ .
  - (d) Dadas duas máquinas de Turing  $M_1$  e  $M_2$  arbitrárias, decidir se  $M_1$  e  $M_2$  computam a mesma função ou não.
  - (e) Dada uma máquina de Turing  $M$  arbitrária, decidir se ela computa uma função com uma imagem finita ou infinita.
  - (f) Seja  $\Gamma$  um alfabeto e  $w_s \in \Gamma^*$ . Dada uma máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  arbitrária, decidir se existe ou não uma entrada  $w_e \in \Sigma^*$  tal que a máquina de Turing  $M$  quando iniciada no estado  $q_0$  e com  $w_e$  na fita de entrada pára num estado final com  $w_s$  na fita. Observe que  $\Gamma$  e  $w_s$  são fixos.
  - (g) Dada uma máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  arbitrária, decidir se existe ou não uma máquina de Turing  $\overline{M}$  que reconheça a linguagem  $\overline{L(M)}$ .
  - (h) Dada uma máquina de Turing  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  arbitrária, decidir se ela computa uma função injetiva ou não.
2. Dos problemas acima identifique quais deles são semi-decidíveis e quais co-semi-decidíveis. Justifique.
3. Seja  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, F \rangle$  uma máquina de Turing. Em que casos o problema de decidir se, dado um  $w_s \in \Gamma^*$  arbitrário, existe ou não uma entrada  $w_e \in \Sigma^*$  tal que a máquina de Turing  $M$  quando iniciada no estado  $q_0$  e com  $w_e$  na fita de entrada pára num estado final com  $w_s$  na fita, é decidível?
4. Seja  $A = \{001, 0011, 11, 101\}$  e  $B = \{01, 111, 111, 010\}$ . O par  $(A, B)$  tem uma solução-CP? caso positivo, tem uma solução-CPM?.
5. Dê uma solução-CP para as seguintes seqüências de cadeias sobre  $\{a, b\}$ .
  - $A = \{aba, baba, abaa, baab, ab\}$
  - $B = \{baaaba, bbab, baba, baa, a\}$

## Apêndice A

# Soluções de Alguns dos Exercícios Propostos

### A.1 Soluções de Exercícios do Capítulo 1

2. Demonstre que para quaisquer cadeias  $w$  e  $u$  e para todo  $n \geq 0$  temos que

$$(2b) |w^n| = n|w|,$$

**Resposta:** Provaremos por indução em  $n$ .

Base da Indução:  $|w^0| = |\lambda| = 0 = 0|w|$ .

Hipóteses indutiva: Suponha que  $|w^k| = k|w|$ .

Passo indutivo:  $|w^{k+1}| = |ww^k| = |w| + |w^k| = |w| + k|w| = (k+1)|w|$ .

$$(2d) (wu)^R = u^R w^R.$$

**Resposta:** Provaremos por indução em  $|u|$ .

Base da Indução:  $(w\lambda)^R = w^R = \lambda^R w^R$ .

Hipóteses indutiva: Suponha que  $(wu)^R = u^R w^R$  para todo  $u$  tal que  $|u| = k$ .

Passo indutivo: Para todo símbolo  $a$  do alfabeto de  $u$ ,  $(w(ua))^R = ((wu)a)^R = a(wu)^R = au^R w^R = (ua)^R w^R$ .

5. Mostre que para qualquer  $n, m \geq 0$  e linguagem  $\mathcal{L}$  temos que

$$(5a) (\mathcal{L}^n)^m = \mathcal{L}^{nm},$$

**Resposta:** Provaremos por indução em  $m$ .

Base da Indução:  $(\mathcal{L}^n)^0 = \{\lambda\} = \mathcal{L}^0 = \mathcal{L}^{n0}$ .

Hipóteses indutiva: Suponha que  $(\mathcal{L}^n)^k = \mathcal{L}^{nk}$ .

Passo indutivo:  $(\mathcal{L}^n)^{k+1} = (\mathcal{L}^n)^k \mathcal{L}^n = \mathcal{L}^{nk} \mathcal{L}^n = \mathcal{L}^{nk+n} = \mathcal{L}^{n(k+1)}$

$$(5c) (\mathcal{L}^R)^n = (\mathcal{L}^n)^R,$$

## A.2. Soluções de Exercícios do Capítulo 2

---

**Resposta:** Provaremos por indução em  $n$ .

Base da Indução:  $(\mathcal{L}^R)^0 = \{\lambda\} = \{\lambda\}^R = (\mathcal{L}^0)^R$ .

Hipóteses indutiva: Suponha que  $(\mathcal{L}^R)^k = (\mathcal{L}^k)^R$ .

Passo indutivo:  $(\mathcal{L}^R)^{k+1} = (\mathcal{L}^R)^k \mathcal{L}^R = (\mathcal{L}^k)^R \mathcal{L}^R = (\mathcal{L} \mathcal{L}^k)^R = (\mathcal{L}^{k+1})^R$

(5e)  $\mathcal{L}^P = ((\mathcal{L})^R)^S$ .

**Resposta:** Se  $\mathcal{L} = \emptyset$  então trivialmente  $\mathcal{L}^P = ((\mathcal{L})^R)^S$ . Provaremos por indução no tamanho da cadeia  $w$  que  $w \in \mathcal{L}^P$  se e somente se  $w \in ((\mathcal{L})^R)^S$  quando  $\mathcal{L} \neq \emptyset$ .

Base da Indução:  $\lambda \in \mathcal{L}^P$  pois  $\lambda$  é prefixo de qualquer cadeia. Por outro lado,  $\lambda \in (\mathcal{L})^R$  pois  $\lambda$  é sufixo de qualquer cadeia, e portanto,  $\lambda \in ((\mathcal{L})^R)^S$ .

Hipóteses indutiva: Suponha que para todo  $w$  tal que  $|w| = k$ ,  $w \in \mathcal{L}^P$  se e somente se  $w \in ((\mathcal{L})^R)^S$ .

Passo indutivo:  $wa \in \mathcal{L}^P$  se e somente se existe uma cadeia  $u \in \Sigma^*$  tal que  $wau \in \mathcal{L}$ , e isso só ocorre se e somente existe uma cadeia  $u \in \Sigma^*$  tal que  $(wau)^R = u^R a w^R \in \mathcal{L}^R$  (veja exercício (2d)), o qual é verdade se e somente se  $aw^R \in (\mathcal{L}^R)^S$ . Portanto,  $wa \in \mathcal{L}^P$  se e somente se  $(aw^R)^R \in ((\mathcal{L}^R)^S)^R$ , isto é, se e somente se,  $wa \in ((\mathcal{L}^R)^S)^R$ .

7. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  duas linguagens tais que  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ . Demonstre que

(7b)  $\mathcal{L}_1^R \subseteq \mathcal{L}_2^R$ ,

**Resposta:** Se  $w \in \mathcal{L}_1^R$  então  $w^R \in \mathcal{L}_1$ . Como por hipótese  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , então  $w^R \in \mathcal{L}_2$ . Logo,  $w \in \mathcal{L}_2^R$ .

(7d) Para toda linguagem  $\mathcal{L}$ ,  $\mathcal{L}_1 \circ \mathcal{L} \subseteq \mathcal{L}_2 \circ \mathcal{L}$ .

**Resposta:** Se  $w \in \mathcal{L}_1 \circ \mathcal{L}$  então existe  $u \in \mathcal{L}_1$  e  $v \in \mathcal{L}$  tais que  $w = uv$ . Como por hipótese,  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , então  $u \in \mathcal{L}_2$ . Logo,  $w = uv \in \mathcal{L}_2 \circ \mathcal{L}$ .

9. Descreva de maneira genérica em que situações  $\mathcal{L}^P = (\mathcal{L}^S)^R$ .

**Resposta:** Pelo exercício 5c isso só ocorrerá quando  $\mathcal{L} = \mathcal{L}^R$  isto é quando  $\mathcal{L}$  for um subconjunto dos palíndromos.

## A.2 Soluções de Exercícios do Capítulo 2

1. Construir um autômato para reconhecer somente a seqüência 10110.

**Resposta:** Um AFD que reconhece somente a cadeia 10110 é ilustrada na figura A.24.

3. Construir um autômato finito determinístico que reconheça qualquer cadeia contendo um número qualquer de cópias de 001, seguido por 1 e nenhuma outra cadeia. Isto é o AFD deve reconhecer a linguagem:

$$\{w \in \{0,1\}^* / w = 001^n \text{ para algum } n \geq 0\}$$

As cadeias reconhecidas por este autômato são do tipo: 1, 0011, 0010011, 0010010011, etc.

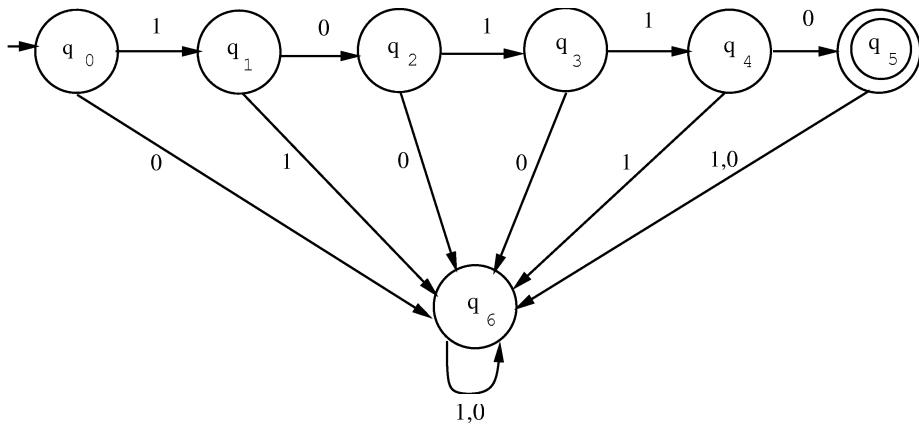


Figura A.1: AFD para o exercício 1.

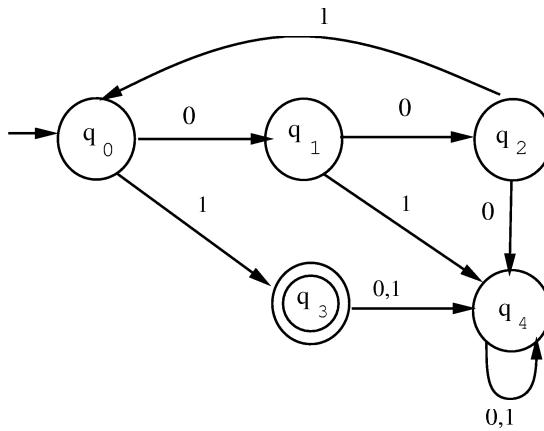


Figura A.2: AFD para o exercício 3.

**Resposta:** Um AFD que reconhece esta linguagem é ilustrado na figura A.2.

4. Para cada um dos seguintes casos, construir um AFD que com os símbolos de entrada  $a, b, c, d$  reconheça a descrição das cadeias dadas e não outras.

(4b) Qualquer cadeia terminando com um  $a$ .

**Resposta:** Um AFD que reconhece esta linguagem é ilustrado na figura A.3.

(4c) Qualquer cadeia consistindo de um número par de cópias de  $abb$ .

**Resposta:** Um AFD que reconhece esta linguagem é ilustrado na figura A.4.

(4d) Qualquer cadeia em  $\{a, bcd\}^*$ . Isto é a linguagem

$$\{\lambda, a, bcd, aa, abcd, bcdbcd, bcda, aaa, aabcd, \dots\}.$$

## A.2. Soluções de Exercícios do Capítulo 2

---

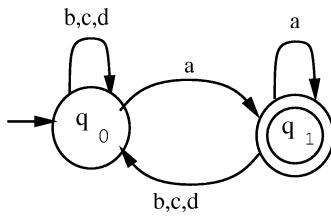


Figura A.3: AFD para o exercício (4b).

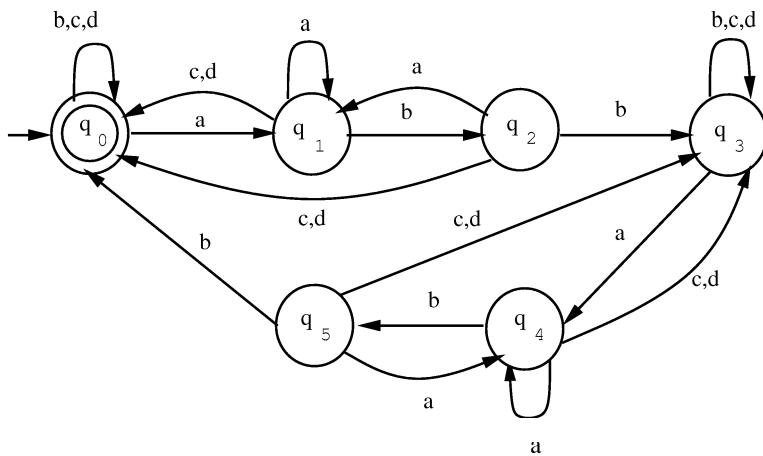


Figura A.4: AFD para o exercício (4c).

**Resposta:** O AFD da figura A.5 reconhece a linguagem  $\{a, bcd\}^*$ .

(4e) Qualquer cadeia contendo exatamente uma única cópia da cadeia  $abbb$ .

**Resposta:** Um AFD que reconhece esta linguagem é ilustrado na figura A.6.

5. Desenhe AFD's que reconheçam os subconjuntos de  $\{a, b\}^*$  consistindo de

(5a) Todas as cadeias com exatamente um  $a$ .

**Resposta:** O AFD na figura A.7 reconhece esta linguagem.

(5c) Todas as cadeias com mais do que três  $a$ 's.

**Resposta:** O AFD da figura A.8 reconhece esta linguagem.

6. Dar uma AFD que aceite os números, no sistema binário, que:

(6a) São múltiplos de 4.

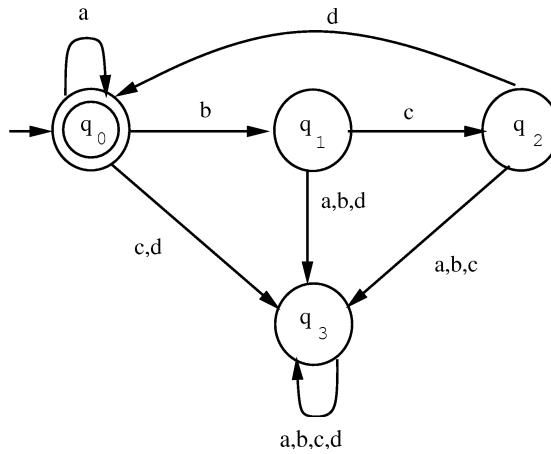


Figura A.5: AFD para o exercício (4d).

**Resposta:** Antes de mostrar o AFD que reconhece esta linguagens, devemos analisar que linguagem é esta.

Valor em Decimal	Valor em binário
0	0
4	100
8	1000
12	1100
16	10000
20	10100
:	:

Claramente, um número diferente de zero em binário é múltiplo de 4 se seus dois últimos dígitos são 00.

Nos consideraremos zeros à esquerda como não significativos, e portanto 0000100 e 100 devem ser aceitos, pois ambos representam o número 4. A cadeia vazia, por questão de simplicidade será também considerada como zero. Assim, sobre este entendimento, um AFD que reconhece esta linguagem é o AFD da figura A.9.

(6b) São múltiplos de 3.

**Resposta:** Para desenhar este AFD primeiro necessitamos entender a linguagem dos números binários e dentre eles analisar quais são múltiplos de 3. Um número é múltiplo de 3 se o resto da divisão dele por 3 é 0. Então a tabela a seguir mostra os números em decimal, binário e seu resto (em decimal) da divisão dele por 3.

## A.2. Soluções de Exercícios do Capítulo 2

---

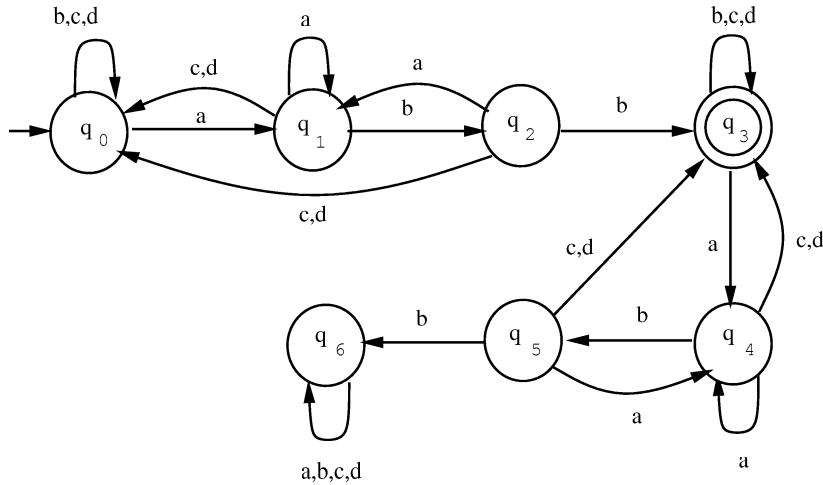


Figura A.6: AFD para o exercício (4d).

Valor em Decimal	Valor em Binário	Resto da Divisão por 3
0	0	0
1	1	1
2	10	2
3	11	0
4	100	1
5	101	2
6	110	0
7	111	1
8	1000	2
9	1001	0
10	1010	1
11	1011	2
12	1100	0
13	1101	1
14	1110	2
15	1111	0
\$\vdots\$	\$\vdots\$	\$\vdots\$

Novamente consideramos que zeros à esquerda não são significativos e que a cadeia vazia é considerada como zero.

Assim seria natural termos três estados. Um para quando o resto for 0 ( $q_0$ ), outro para quando for 1 ( $q_1$ ) e um último para quando o resto for 2 ( $q_2$ ). Como, resto zero significa que o número é múltiplo de 3, então ele seria estado final e como a cadeia vazia (ou seja antes ler qualquer símbolo) é considerada como 0, e 0 é múltiplo de 3, então esse estado também seria inicial.

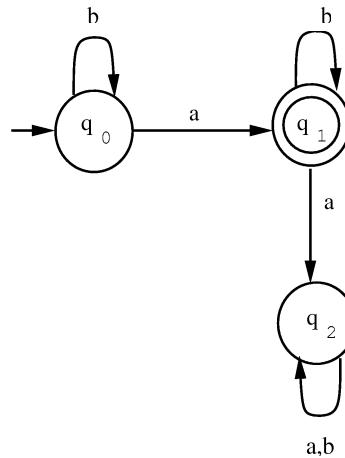


Figura A.7: AFD para o exercício (5a).

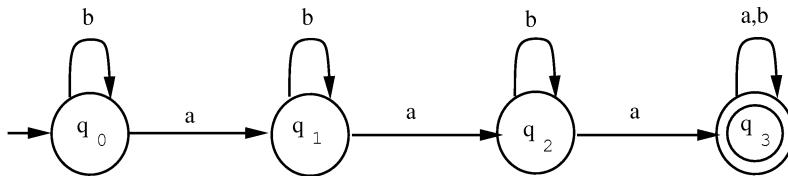


Figura A.8: AFD para o exercício (5c).

Observe que se o valor em decimal de um número binário  $w$  (denotaremos isto por  $\#w$ ), for  $n$ , então  $\#w0$  é  $2n$  e de  $\#w1$  é  $2n+1$ . Assim, se o resto da divisão por 3 de um número binário  $w$  for 0, então isso significa que  $\#w = 3k$  para algum  $k \in \mathbb{N}$ . Logo,  $\#w0 = 2\#w = 2(3k) = 3(2k)$  e portanto o resto da divisão de  $\#w0$  por 3 continuaria sendo 0, isto implica em que teríamos uma transição rotulada por 0 do estado  $q_0$  a ele mesmo. Analogamente,  $\#w1 = 2\#w + 1 = 2(3k) + 1 = 3(2k) + 1$  e portanto o resto da divisão  $\#w1$  por 3 passa a ser 1 o qual implica numa transição rotulada por 1 do estado  $q_0$  ao estado  $q_1$ . Agora se o resto da divisão de  $\#w$  por 3 for 1, então  $\#w = 3k + 1$  para algum  $k \in \mathbb{N}$  e portanto  $\#w0 = 2\#w = 2(3k + 1) = 3(2k) + 2$ . Logo, o resto da divisão de  $\#w0$  por 3 seria 2, o qual implica numa transição rotulada por 0 do estado  $q_1$  ao estado  $q_2$ . Analogamente,  $\#w1 = 2\#w + 1 = 2(3k + 1) + 1 = 3(2k) + 3 = 3(2k + 1)$ . Logo, o resto da divisão de  $\#w0$  por 3 seria 0, o qual implica numa transição rotulada por 1 do estado  $q_1$  ao estado  $q_0$ . Finalmente, se o resto da divisão de  $\#w$  por 3 for 2, então  $\#w = 3k + 2$  para algum  $k \in \mathbb{N}$  e portanto  $\#w0 = 2\#w = 2(3k + 2) = 3(2k) + 3 + 1 = 3(2k + 1) + 1$ . Logo, o resto da divisão de  $\#w0$  por 3 seria 1, o qual implica numa transição rotulada por 0 do estado  $q_2$  ao estado  $q_1$ . Analogamente,  $\#w1 = 2\#w + 1 = 2(3k + 2) + 1 = 3(2k) + 3 + 2 = 3(2k + 1) + 2$ . Logo, o resto da divisão de  $\#w0$  por 3 seria 2, o qual implica numa transição rotulada por 1 do estado  $q_2$  nele mesmo.

## A.2. Soluções de Exercícios do Capítulo 2

---

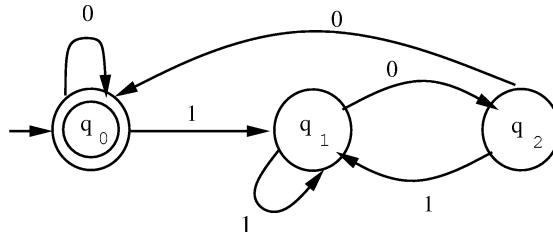


Figura A.9: AFD para o exercício (6a).

O AFD da figura A.10 ilustra este autômato.

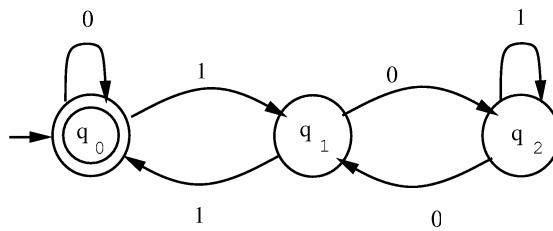


Figura A.10: AFD para o exercício (6b).

7. Encontre AFD's para as seguintes linguagens sobre  $\Sigma = \{a, b\}$ .

(7a)  $\mathcal{L} = \{w \in \Sigma^* / |w| \bmod 3 = 0\}$ ,

**Resposta:** O AFD da figura A.11 reconhece esta linguagem.

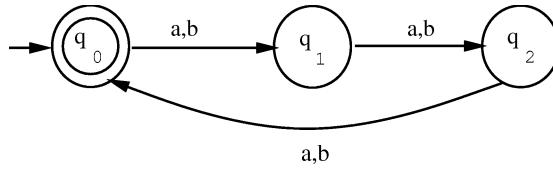


Figura A.11: AFD para o exercício (7a).

(7c)  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \bmod 3 > 1\}$ ,

**Resposta:** O AFD da figura A.12 reconhece esta linguagem.

(7e)  $\mathcal{L} = \{w \in \Sigma^* / 2 \leq \mathcal{N}_a(w) \leq 4\}$ ,

**Resposta:** O AFD da figura A.13 reconhece esta linguagem.

(7g)  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \notin \{1, 2, 3\}\}$ ,

**Resposta:** O AFD da figura A.14 reconhece esta linguagem.

(7i)  $\mathcal{L} = \{a^m b^n / mn \text{ é ímpar}\}$ ,

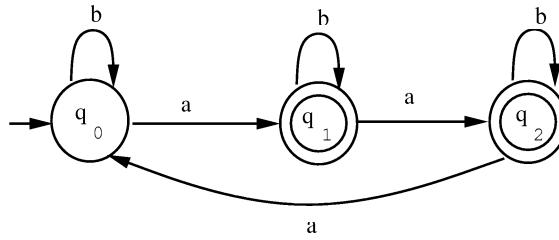


Figura A.12: AFD para o exercício (7c).

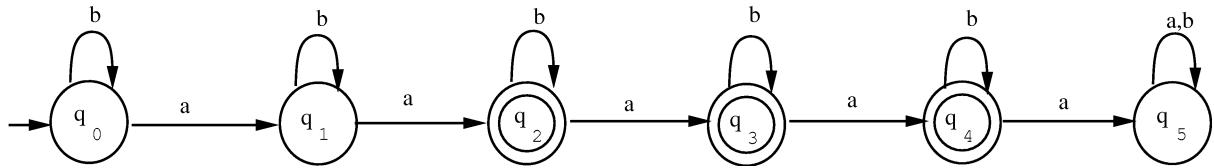


Figura A.13: AFD para o exercício (7e).

**Resposta:** O AFD da figura A.15 reconhece esta linguagem.

(7k)  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \text{ é ímpar e } |w| \text{ é par}\}$ , onde  $\mathcal{N}_a(w)$  é o número de  $a$ 's que ocorrem em  $w$  e  $|w|$  é o tamanho da cadeia  $w$ .

**Resposta:** O AFD da figura A.16 reconhece esta linguagem. Neste autômato o estado  $q_0$  reflete a situação que a cadeia lida até esse momento tem uma quantidade par de  $a$ 's e é de tamanho par, ou seja,  $\mathcal{N}_a(w)$  e  $|w|$  são pares. O estado  $q_1$  sintetiza a situação em que  $\mathcal{N}_a(w)$  e  $|w|$  são ímpares. O estado  $q_2$  captura a situação em que  $\mathcal{N}_a(w)$  é par enquanto  $|w|$  é ímpar. Finalmente, o estado  $q_3$  a situação desejada, ou seja,  $\mathcal{N}_a(w)$  é ímpar e  $|w|$  é par, e portanto é o estado final.

9. Considere o conjunto das cadeias, sobre  $\{0, 1\}$ , definido pela condição abaixo. Exiba um AFD para cada um desses conjuntos.

(9a) O primeiro e último símbolo sejam iguais.

**Resposta:** O AFD da figura A.17 reconhece esta linguagem.

(9c) Que reconheça a linguagem  $\mathcal{L} = \{vwv / v, w \in \{a, b\}^*\text{ e }|v| = 2\}$ .

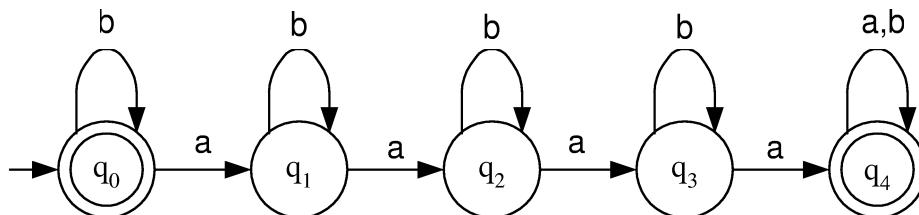


Figura A.14: AFD para o exercício (7g).

## A.2. Soluções de Exercícios do Capítulo 2

---

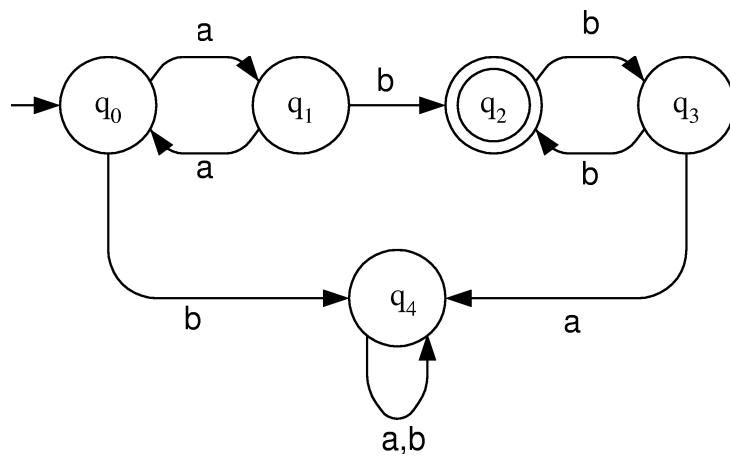


Figura A.15: AFD para o exercício (7i).

**Resposta:** O AFD da figura A.18 reconhece esta linguagem.

- (9e) Todo 00 é seguido imediatamente por 1. Por exemplo, as cadeias  $\lambda$ , 101, 0010, 0010111001 estão na linguagem, mas 0001 e 00100, não estão.

**Resposta:** O AFD da figura A.19 reconhece esta linguagem.

- (9g) Toda subcadeia de quatro símbolos tem, quando muito, dois zeros. Por exemplo, 0011100, 001101 e 1010101 estão na linguagem, mas 10100110 não está, pois uma de suas subcadeias de quatro símbolos, 0100, contém três zeros.

**Resposta:** O AFD da figura A.20 reconhece esta linguagem. Neste AFD cada estado, amenos do estado de morte  $q_3$ , “armazena” os três últimos dígitos lidos. No inicio, para evitar introduzir novos estados, o AFD assume que leu 111 antes de começar. O estado  $q_0$  então representa 111, o estado  $q_1$  representa 110, o estado  $q_2$  representa 100 o estado  $q_3$  representa a situação onde houve uma subcadeia de tamanho 4 com três 0’s, o estado  $q_4$  representa 101, o estado  $q_5$  representa 010, o estado  $q_6$  representa 001 e o estado  $q_7$  representa 011. Observe no AFD da figura A.20 que o estado de chegada das arestas que saem dos estados  $q_7$  e  $q_0$  com o mesmo rótulo coincidem, e portanto esses dois estados poderiam ser fusionados em um único estado.

- (9i) Qualquer ocorrência de dois zeros próximos (isto é sem qualquer 0 no meio), esteja separado por uma quantidade ímpar de uns. Por exemplo,  $\lambda$ , 111, 11011, 0111010111 e 110101110101 fazem parte desta linguagem, mas as cadeias 0010 e 111010110 não.

**Resposta:** O AFD da figura A.21 reconhece esta linguagem.

11. Seja  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  um AFD. Mostre que  $\delta^*(q, vw) = \delta^*(\delta^*(q, v), w)$  para todo  $v, w \in \Sigma^*$ .

**Resposta:** Provaremos por indução no tamanho de  $w$ .

Base da Indução: Se  $w = \lambda$  então  $\delta^*(q, vw) = \delta^*(q, v\lambda) = \delta^*(q, v)$  e  $\delta^*(\delta^*(q, v), w) = \delta^*(\delta^*(q, v), \lambda) = \delta^*(q, v)$  e portanto ambos são iguais.

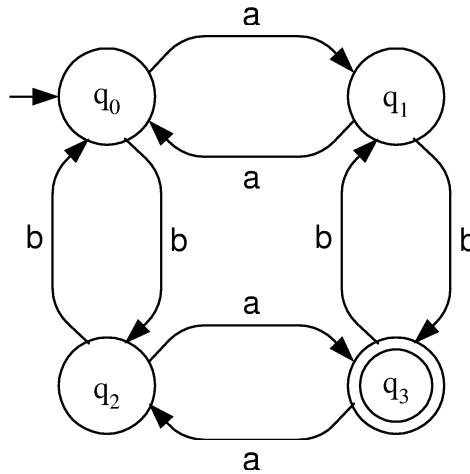


Figura A.16: AFD para o exercício (7k).

Hipóteses indutiva:  $\delta^*(q, vw) = \delta^*(\delta^*(q, v), w)$  para todo  $v, w \in \Sigma^*$  tal que  $|w| = k$

Passo indutivo: Seja  $a \in \Sigma$  e  $v, w \in \Sigma^*$  tal que  $|w| = k$ . Então

$$\begin{aligned} \delta^*(q, vwa) &= \delta(\delta^*(q, vw), a) && \text{por definição de } \delta^* \\ &= \delta(\delta^*(\delta^*(q, v), w), a) && \text{por hipótese indutiva} \\ &= \delta^*(\delta^*(q, v), wa) && \text{por definição de } \delta^* \end{aligned}$$

13. Para o AFN na figura 2.9, ache  $\delta^*(q_1, 100)$  e  $\delta^*(q_0, 00101)$ .

**Resposta:** Faremos isto passo a passo seguindo a definição de  $\delta^*$  para AFN sem  $\lambda$ -transições (equações 2.4 e 2.5).

$$\begin{aligned} \delta^*(q_1, 100) &= \bigcup_{q \in \delta^*(q_1, 1)} \delta(q, 0) \\ &= \bigcup_{q \in \bigcup_{q' \in \delta^*(q_1, 1)} \delta(q', 0)} \delta(q, 0) \\ &= \bigcup_{q \in \bigcup_{q' \in \bigcup_{q'' \in \delta^*(q_1, \lambda)} \delta(q'', a)} \delta(q', 0)} \delta(q, 0) \\ &= \bigcup_{q \in \bigcup_{q' \in \bigcup_{q'' \in \{q_1\}} \delta(q'', a)} \delta(q', 0)} \delta(q, 0) \\ &= \bigcup_{q \in \bigcup_{q' \in \delta(q_1, a)} \delta(q', 0)} \delta(q, 0) \\ &= \bigcup_{q \in \bigcup_{q' \in \{q_2\}} \delta(q', 0)} \delta(q, 0) \\ &= \bigcup_{q \in \delta(q_2, 0)} \delta(q, 0) \\ &= \bigcup_{q \in \{q_1\}} \delta(q, 0) \\ &= \delta(q_1, 0) = \emptyset \end{aligned}$$

Para o outro caso, usaremos a idéia intuitiva para a função de transição estendida e que foi expressa na definição 2.4.5.

## A.2. Soluções de Exercícios do Capítulo 2

---

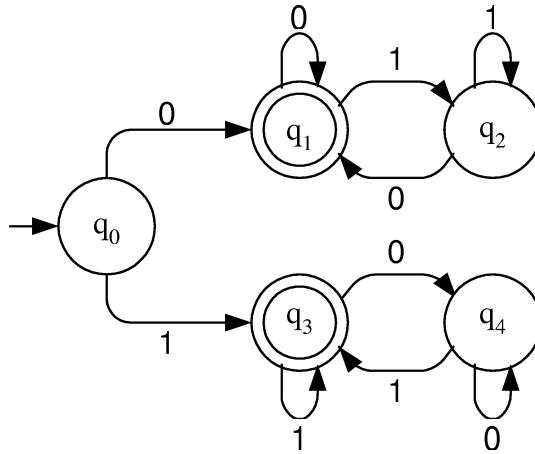


Figura A.17: AFD para o exercício (9a).

$$\begin{aligned}
 \delta^*(q_0, 00101) &= \delta^*(q_0, 0101) \cup \delta^*(q_1, 0101) && \text{pois } \delta(q_0, 0) = \{q_0, q_1\} \\
 &= (\delta^*(q_0, 101) \cup \delta^*(q_1, 101)) \cup \emptyset && \text{pois } \delta(q_0, 0) = \{q_0, q_1\} \text{ e } \delta(q_1, 1) = \emptyset \\
 &= \emptyset \cup \delta^*(q_2, 01) && \text{pois } \delta(q_0, 1) = \emptyset \text{ e } \delta(q_1, 1) = \{q_2\} \\
 &= \delta^*(q_1, 1) && \text{pois } \delta(q_2, 0) = \{q_1\} \\
 &= \delta^*(q_2, \lambda) && \text{pois } \delta(q_1, 1) = \{q_2\} \\
 &= \{q_2\}
 \end{aligned}$$

15. Seja  $\mathcal{L}$  a linguagem aceita pelo autômato da figura 2.2. Achar um AFD que aceite  $\mathcal{L}^2$ .

**Resposta:** Antes de desenhar o AFD que aceite a linguagem  $\mathcal{L}^2$ , onde  $\mathcal{L}$  é a linguagem reconhecida pelo autômato da figura 2.2 devemos analisar quem é  $\mathcal{L}$  e quem é  $\mathcal{L}^2$ .

Claramente,  $\mathcal{L} = \{a^n b / n \geq 0\}$  e portanto  $\mathcal{L}^2 = \{a^n b a^m b / n \geq 0 \text{ e } m \geq 0\}$ . Assim, o autômato da figura A.22 reconhece  $\mathcal{L}^2$ .

17. Projete um AFN, com não mais que cinco estados, para o conjunto

$$\{abab^n / n \geq 0\} \cup \{aba^n / n \geq 0\}$$

**Resposta:** Um AFN com cinco estados que reconhece esta linguagem é mostrado na figura A.23.

19. Construa um AFN que aceite a linguagem de todas as cadeias sobre o alfabeto  $\{0, 1\}$  com um número par de 1's ou um número ímpar de 0's.

**Resposta:** Um AFN que reconhece esta linguagem é ilustrado na figura A.24.

21. Use a construção do teorema 2.5.3 para converter os AFN's das figuras 2.11, 2.15, 2.24, 2.25, 2.26 e 2.27, e do exercício 19 para AFD's equivalentes. É possível respostas mais simples se fizermos os AFD's diretamente?

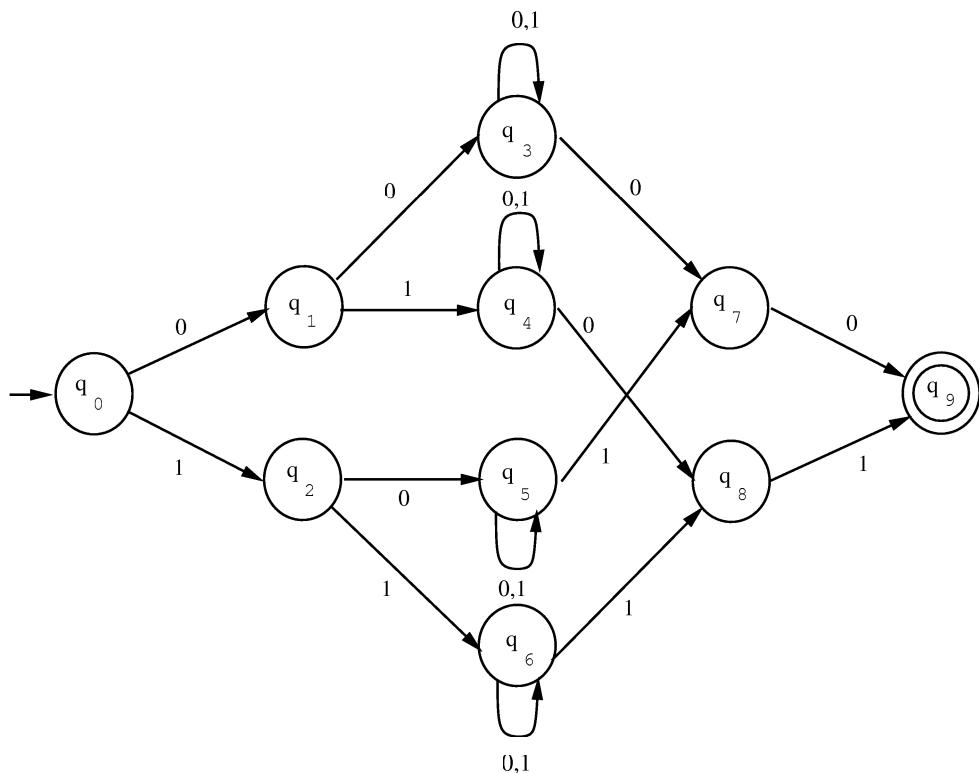


Figura A.18: AFD para o exercício (9c).

**Resposta:** Aqui só veremos dois casos.

1. A figura A.25 mostra a conversão do AFN do exercício 19 (figura A.24), para um AFD.

É possível diminuir a quantidade de estados no autômato, eliminando o estado inicial com suas  $\lambda$ -transições e levando um controle da paridade de 1's e 0's lidos, de modo similar ao autômato construído no exemplo 2.2.6. Assim, o AFD da figura A.26 é uma outra alternativa para reconhecer a linguagem do exercício 19 com menos estados que o da figura A.25.

2. A figura A.27 mostra a conversão do AFN da figura 2.24, para um AFD.

23. É verdade que para qualquer AFN  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  o complemento de  $L(M)$  é igual ao conjunto  $\{w \in \Sigma^* / \delta^*(q_0, w) \cap F = \emptyset\}$ ? Prove sua resposta.

**Respostas:** Sim, pois

$$\begin{aligned} \overline{L(M)} &= \sigma^* - L(M) \\ &= \Sigma^* - \{w \in \Sigma^* / \delta^*(q_0, w) \cap F \neq \emptyset\} \\ &= \{w / w \in \Sigma^* \text{ e } w \notin \{w \in \Sigma^* / \delta^*(q_0, w) \cap F \neq \emptyset\}\} \end{aligned}$$

Como  $\{w \in \Sigma^* / \delta^*(q_0, w) \cap F \neq \emptyset\} \subseteq \Sigma^*$ , temos que

$$\begin{aligned} \overline{L(M)} &= \{w \in \Sigma^* / \text{não é o caso que } \delta^*(q_0, w) \cap F \neq \emptyset\} \\ &= \{w \in \Sigma^* / \delta^*(q_0, w) \cap F = \emptyset\} \end{aligned}$$

## A.2. Soluções de Exercícios do Capítulo 2

---

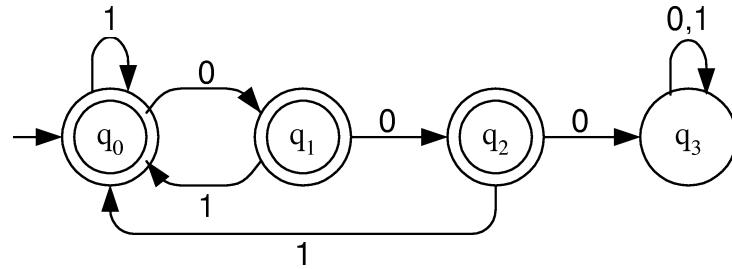


Figura A.19: AFD para o exercício (9e).

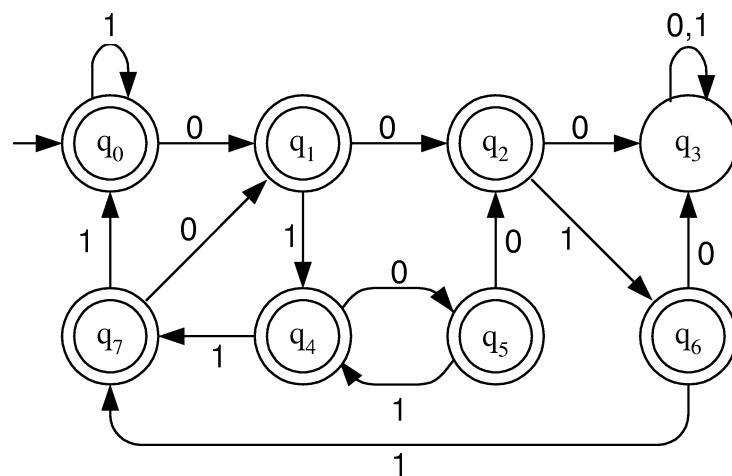


Figura A.20: AFD para o exercício (9g).

24. Mostre que se  $\mathcal{L}$  é reconhecida por um autômato finito (determinístico ou não), então  $\mathcal{L}^R$  e  $\overline{\mathcal{L}}$  também são reconhecidos por algum autômato finito.

**Resposta:** Se  $\mathcal{L}$  é uma linguagem regular, então existe um AFD  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  tal que  $L(M) = \mathcal{L}$ , isto é que reconhece  $\mathcal{L}$ . Seja o seguinte AFN  $M^R = \langle \widehat{Q}, \Sigma, \widehat{\delta}, \widehat{q}_0, \widehat{F} \rangle$  onde

1.  $\widehat{Q} = Q \cup \widehat{q}_0$
2.  $\widehat{q}_0 \notin Q$  e
3.  $\widehat{\delta} : \widehat{Q} \times \Sigma \cup \{\lambda\} \longrightarrow 2^{\widehat{Q}}$  é definido como segue:

$$\widehat{\delta}(q, a) = \begin{cases} \{q'\} & , \text{ se } q \in Q \text{ e } a \in \Sigma \text{ e } \delta(q', a) = q \\ \emptyset & , \text{ se } q = \widehat{q}_0 \text{ e } a \in \Sigma \\ \emptyset & , \text{ se } q \in Q \text{ e } a = \lambda \\ F & , \text{ se } q = \widehat{q}_0 \text{ e } a = \lambda \end{cases}$$

4.  $\widehat{F} = \{q_0\}$

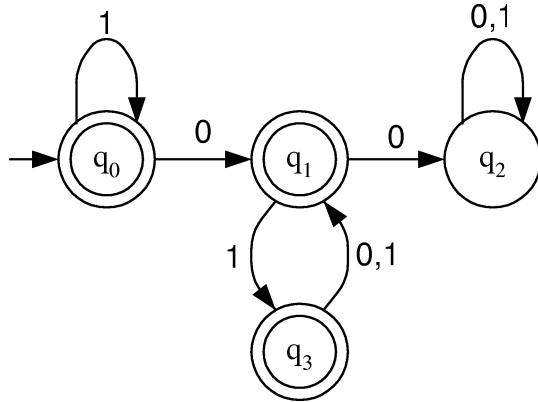


Figura A.21: AFD para o exercício (9i).

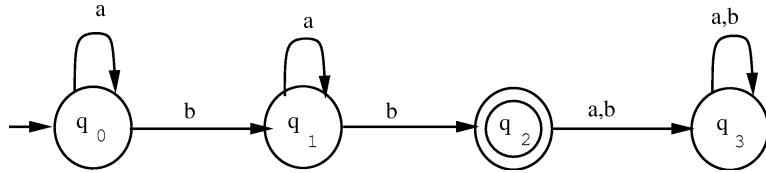


Figura A.22: AFD para o exercício 15.

Observe que o único que se fez foi inverter as setas no diagrama, tirar o status de estado inicial a  $q_0$  e pôr um novo estado como inicial ( $\hat{q}_0$ ), tirar o status dos estados finais e deixar como estado final somente a  $\{q_0\}$  (o antigo estado inicial); e acrescentar uma  $\lambda$ -transição do novo estado inicial aos antigos estados finais. Assim, se para um certo  $w \in \Sigma^*$  existe um caminho em  $M$  que leva do estado inicial  $q_0$  a um estado final  $q_f$ , isto é  $\delta^*(q_0, w) \in F$ , então aplicando a  $\lambda$ -transição de  $\hat{q}_0$  a  $q_f$  no AFN  $M^R$  e (como as setas estão invertidas) lendo  $w$  de direita para esquerda (ou seja consumindo  $w^R$ ) chegaremos necessariamente a  $q_0$  que agora é um estado final. Portanto  $w^R$  será aceito no AFN  $M^R$ . Assim,  $M^R$  reconhece a linguagem  $\mathcal{L}^R$ , isto é  $L(M^R) = \mathcal{L}^R$  e portanto esta linguagem é regular.

No caso da linguagem  $\overline{\mathcal{L}}$  podemos facilmente obter um AFD a partir de  $M$  que a reconheça. Seja  $\overline{M} = \langle Q, \Sigma, \delta, q_0, \overline{F} \rangle$ , onde  $\overline{F} = \Sigma^* - F$ . Claramente, se  $\delta(q_0, w) \in F$  (ou seja  $w$  é aceito por  $M$  e por tanto faz parte de  $\mathcal{L}$ ), então  $\delta(q_0, w) \notin \overline{F}$ , isto é,  $\delta(q_0, w) \in \overline{F}$  e portanto  $w$  é aceito por  $\overline{M}$ . Assim,  $\overline{M}$  reconhece a linguagem  $\overline{\mathcal{L}}$ , isto é  $L(\overline{M}) = \overline{\mathcal{L}}$  e portanto esta linguagem é regular.

25. Seja  $\Sigma = \{a, b\}$ . Construa um AFN, que não seja AFD, e depois transforme ele num AFD usando o algoritmo do teorema 2.5.3, para as seguintes linguagens:

$$(25a) \mathcal{L} = \{w \in \Sigma^* / w = uaaav \text{ para alguma cadeia } u \text{ e } v \text{ tal que } N_b(v) \text{ é múltiplo de 3}\},$$

**Resposta:** O AFN da figura A.28 reconhece esta linguagem.

$$(25c) \mathcal{L} = \{w \in \Sigma^* / w = uaaav, \text{ para alguma cadeia } u, v \in \Sigma^*, \text{ e } N_a(u) \text{ é ímpar}\},$$

## A.2. Soluções de Exercícios do Capítulo 2

---

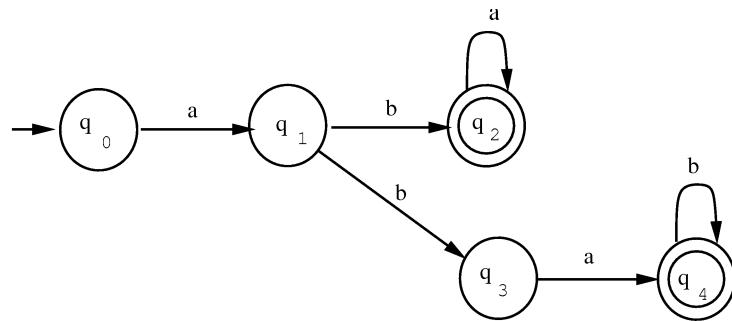


Figura A.23: AFN para o exercício 17.

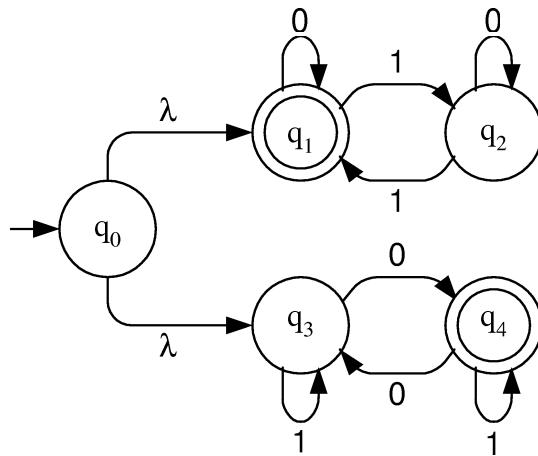


Figura A.24: AFN para o exercício 19.

**Resposta:** O AFN da figura A.29 reconhece esta linguagem.

(25e)  $\mathcal{L} = \{uv \in \Sigma^* / \mathcal{N}_a(u) = 3 \text{ e } \mathcal{N}_b(v) \text{ é múltiplo de } 3\}$ ,

**Resposta:** O AFN da figura A.30 reconhece esta linguagem.

27. Seja o AFD da figura A.31 e aplique o algoritmo da minimização de estados para achar um AFD mínimo equivalente a ele.

**Resposta:** Aplicando o passo 1 do algoritmo obtemos a tabela A.1

Agora aplicando o passo 2 (uma primeira rodada):

- como  $\delta(q_0, b) = q_3$ ,  $\delta(q_2, b) = q_4$ , e  $(q_3, q_4)$  está marcado, então colocamos uma marca na posição  $(q_0, q_2)$ .

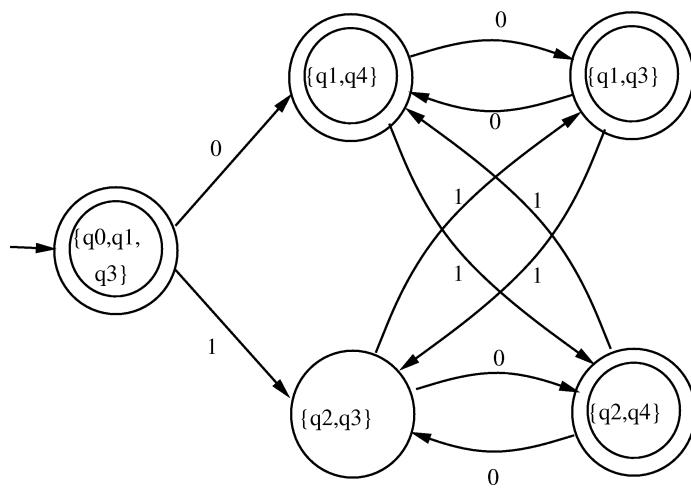


Figura A.25: AFD equivalente ao AFN da figura A.24.

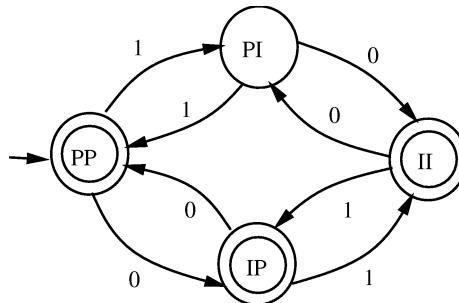


Figura A.26: AFD equivalente ao AFD da figura A.25.

- como  $\delta(q_0, a) = q_2$ ,  $\delta(q_3, a) = q_4$ , e  $(q_2, q_4)$  está marcado, então colocamos uma marca na posição  $(q_0, q_3)$ .
- como  $\delta(q_0, a) = q_2$ ,  $\delta(q_6, a) = q_4$ , e  $(q_2, q_4)$  está marcado, então colocamos uma marca na posição  $(q_0, q_6)$ .
- como  $\delta(q_0, b) = q_3$ ,  $\delta(q_7, b) = q_4$ , e  $(q_3, q_4)$  está marcado, então colocamos uma marca na posição  $(q_0, q_7)$ .
- como  $\delta(q_1, b) = q_3$ ,  $\delta(q_2, b) = q_4$ , e  $(q_3, q_4)$  está marcado, então colocamos uma marca na posição  $(q_1, q_2)$ .
- como  $\delta(q_1, a) = q_7$ ,  $\delta(q_3, a) = q_4$ , e  $(q_4, q_7)$  está marcado, então colocamos uma marca na posição  $(q_1, q_3)$ .
- como  $\delta(q_1, b) = q_3$ ,  $\delta(q_5, b) = q_1$ , e  $(q_1, q_3)$  está marcado, então colocamos uma marca na posição  $(q_1, q_5)$ .
- como  $\delta(q_1, a) = q_7$ ,  $\delta(q_6, a) = q_4$ , e  $(q_4, q_7)$  está marcado, então colocamos uma marca na posição  $(q_1, q_6)$ .

## A.2. Soluções de Exercícios do Capítulo 2

---

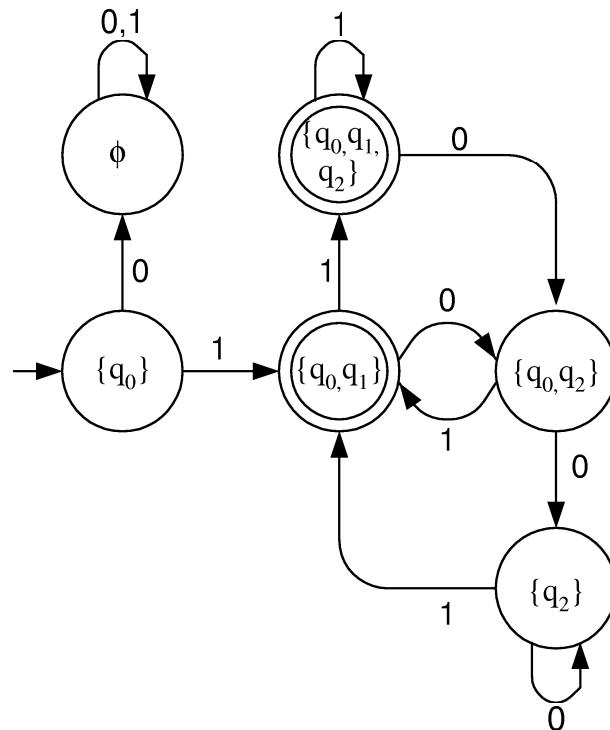


Figura A.27: AFD equivalente ao AFN da figura 2.24.

- como  $\delta(q_1, b) = q_3$ ,  $\delta(q_7, b) = q_4$ , e  $(q_3, q_4)$  está marcado, então colocamos uma marca na posição  $(q_1, q_7)$ .
- como  $\delta(q_2, b) = q_4$ ,  $\delta(q_3, b) = q_5$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca na posição  $(q_2, q_3)$ .
- como  $\delta(q_2, b) = q_4$ ,  $\delta(q_5, b) = q_1$ , e  $(q_1, q_4)$  está marcado, então colocamos uma marca na posição  $(q_2, q_5)$ .
- como  $\delta(q_2, b) = q_4$ ,  $\delta(q_6, b) = q_5$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca na posição  $(q_2, q_6)$ .
- como  $\delta(q_3, a) = q_4$ ,  $\delta(q_5, a) = q_5$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca

$q_1$							
$q_2$							
$q_3$							
$q_4$	×	×	×	×			
$q_5$					×		
$q_6$					×		
$q_7$					×		
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

Tabela A.1: Estados que não são do mesmo tipo no AFD da Figura A.31.

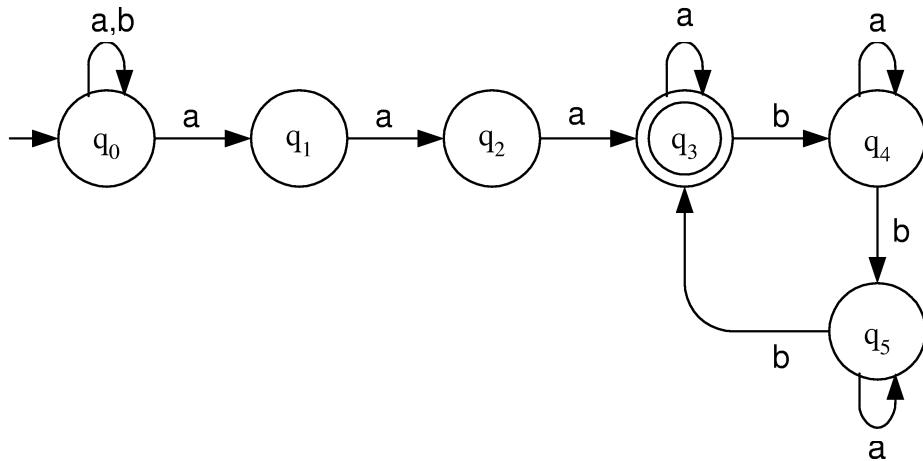


Figura A.28: AFN para o exercício (25a).

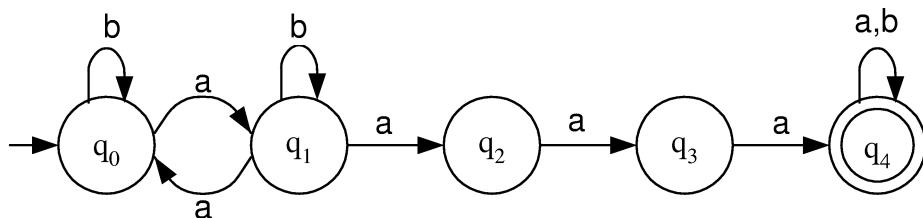


Figura A.29: AFN para o exercício (25c).

na posição  $(q_3, q_5)$ .

- como  $\delta(q_3, a) = q_4$ ,  $\delta(q_7, a) = q_5$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca na posição  $(q_3, q_7)$ .
- como  $\delta(q_5, a) = q_5$ ,  $\delta(q_6, a) = q_4$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca na posição  $(q_5, q_6)$ .
- como  $\delta(q_5, b) = q_1$ ,  $\delta(q_7, b) = q_4$ , e  $(q_1, q_4)$  está marcado, então colocamos uma marca na posição  $(q_5, q_7)$ .
- como  $\delta(q_6, a) = q_4$ ,  $\delta(q_7, a) = q_5$ , e  $(q_4, q_5)$  está marcado, então colocamos uma marca na posição  $(q_6, q_7)$ .

Agora aplicando o passo 2 (uma segunda rodada):

- como  $\delta(q_0, a) = q_2$ ,  $\delta(q_5, a) = q_5$ , e  $(q_2, q_5)$  está marcado, então colocamos uma marca na posição  $(q_0, q_5)$ .

A tabela A.2 apresenta o resultado de marcar as posições acima na tabela A.1.

Observe que as três posições em branco não podem ser marcadas pois:

## A.2. Soluções de Exercícios do Capítulo 2

---

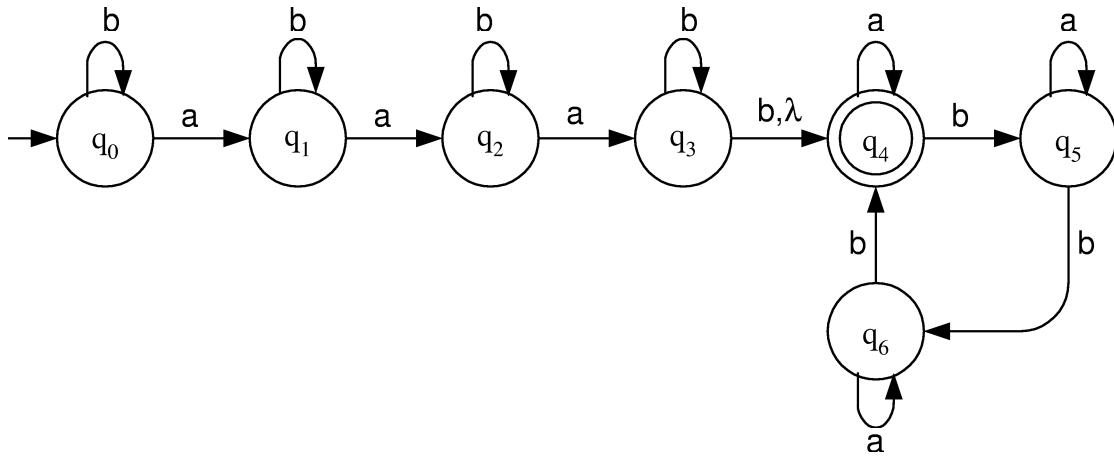


Figura A.30: AFN para o exercício (25e).

$q_1$							
$q_2$	x	x					
$q_3$	x	x	x				
$q_4$	x	x	x	x			
$q_5$	x	x	x	x	x		
$q_6$	x	x	x		x	x	
$q_7$	x	x		x	x	x	x
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$

Tabela A.2: Estados que são equivalentes no AFD da Figura A.31.

- $\delta(q_0, a) = q_2$ ,  $\delta(q_1, a) = q_7$ , e  $(q_2, q_7)$  não está marcado e  $\delta(q_0, b) = \delta(q_1, b)$ .
- $\delta(q_2, a) = \delta(q_7, a)$  e  $\delta(q_2, a) = \delta(q_7, a)$ .
- $\delta(q_3, a) = \delta(q_6, a)$  e  $\delta(q_3, a) = \delta(q_6, a)$ .

Assim, a tabela A.2 apresenta os estados que são equivalentes no AFD da Figura A.31 e portanto podemos determinar as seguintes classes de equivalências:

- $[q_0] = \{q_0, q_1\}$
- $[q_2] = \{q_2, q_7\}$
- $[q_3] = \{q_3, q_6\}$
- $[q_4] = \{q_4\}$
- $[q_5] = \{q_5\}$

Estas classes de equivalências serão os estados do AFD mínimo. O estado inicial é  $[q_0]$  e o estado final é  $[q_4]$ . As transições são descritas a seguir:

- $\delta_{/\equiv}([q_0], a) = [\delta(q_0, a)] = [q_2]$

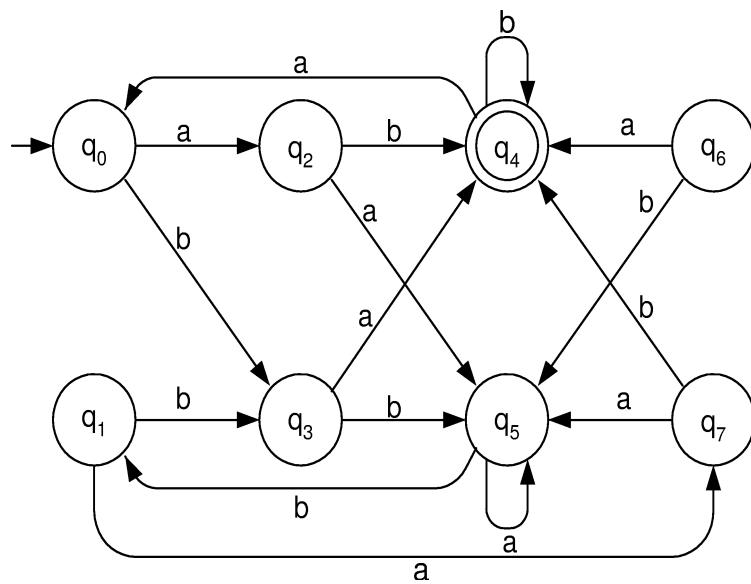


Figura A.31: AFD não mínimo do exercício 27

- $\delta_{/\equiv}([q_0], b) = [\delta(q_0, b)] = [q_3]$
- $\delta_{/\equiv}([q_2], a) = [\delta(q_2, a)] = [q_5]$
- $\delta_{/\equiv}([q_2], b) = [\delta(q_2, b)] = [q_4]$
- $\delta_{/\equiv}([q_3], a) = [\delta(q_3, a)] = [q_4]$
- $\delta_{/\equiv}([q_3], b) = [\delta(q_3, b)] = [q_5]$
- $\delta_{/\equiv}([q_4], a) = [\delta(q_4, a)] = [q_0]$
- $\delta_{/\equiv}([q_4], b) = [\delta(q_4, b)] = [q_4]$
- $\delta_{/\equiv}([q_5], a) = [\delta(q_5, a)] = [q_5]$
- $\delta_{/\equiv}([q_5], b) = [\delta(q_5, b)] = [q_1] = [q_0]$

Assim a figura A.32 é o AFD mínimo correspondente ao AFD da figura A.31.

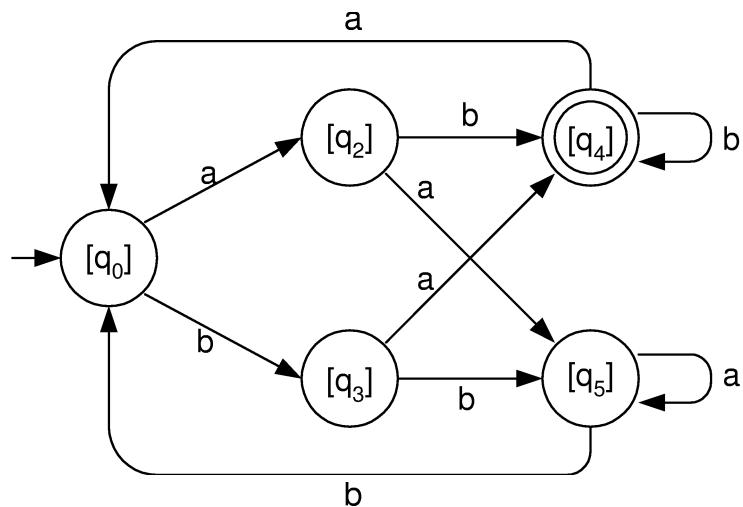


Figura A.32: AFD  $M_{\equiv}$  obtido a partir do AFD da Figura A.31.

### A.3 Soluções de Exercícios do Capítulo 3

3. Escrever expressões regulares para as seguintes linguagens no alfabeto  $\{0, 1\}$ :

(3a) Todas as cadeias terminando em 01.

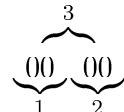
**Resposta:**  $(0 + 1)^*01$

(3c) Todas as cadeias com um número par de zeros.

**Resposta:**  $(01 * 0 + 1)^*$

(3d) Todas as cadeias com no máximo duas ocorrências da subcadeia 00.

**Resposta:** Entenderemos neste enunciado que a cadeia 0000 contém 3 ocorrências da subcadeia 00



Analogamente a cadeia 000 contém exatamente duas ocorrências da subcadeia 00.

Nesta compreensão, algumas expressões regulares que descrevem esta linguagem são:

1.  $(01 + 1)^*(001(01 + 1)^*00 + 000 + 00 + \lambda)(10 + 1)^*$
2.  $(01 + 1)^*((00 + \lambda)(1 + 10)^*(100 + \lambda) + 000)(10 + 1)^*$
3.  $(01 + 1)^*(00((10 + 1)^*0(10 + 1)^* + 1 + \lambda) + \lambda + 0)$
4.  $(01 + 1)^*00(10 + 1)^*0(10 + 1)^* + (01 + 1)^*001^* + (01 + 1)^*$

(3e) de todas as cadeias que contenham as subcadeias 000 e 111.

**Resposta:**  $((0 + 1)^*000(0 + 1)^*111(0 + 1)^*) + ((0 + 1)^*111(0 + 1)^*000(0 + 1)^*)$

(3g) Todas as cadeias que para alguma ocorrência de dois zeros eles estejam separados por uma subcadeia de tamanho  $3i$ , para algum  $i \geq 0$ .

**Resposta:** Observe que uma cadeia com dois zeros consecutivos faz parte da linguagem.

$$(0 + 1)^*0(101 + 111)^*0(0 + 1)^*$$

(3j) Todas as cadeias com uma quantidade ímpar de zeros.

**Resposta:**  $1^*0(1^*01^*0)^*1^*$

(3l) Todas as cadeias que não contenham qualquer ocorrência da subcadeia 000.

**Resposta:**  $(1 + 01 + 001)^*$ .

(3n) Todas as cadeias com exatamente uma ocorrência da subcadeia 111.

### A.3. Soluções de Exercícios do Capítulo 3

---

**Resposta:**  $(0 + 10 + 110)^*111(0 + 10 + 110)^*$

- (3p) Todas as cadeias com exatamente duas ocorrências da subcadeia 000 e as quais estejam separadas por uma quantidade ímpar de uns.

**Resposta:**  $(1 + 01 + 001)^*0001(11)^*000(1 + 01 + 001)^*.$

5. Demonstre que:

$$(5a) (a + b)^* \equiv (a^*b^*)^*$$

**Resposta:**

$$\begin{aligned} L((a + b)^*) &= (L(a + b))^* \\ &= (L(a) \cup L(b))^* \\ &= (\{a\} \cup \{b\})^* \\ &= \{a, b\}^* \\ &= \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\} \end{aligned}$$

$$\begin{aligned} L((a^*b^*)^*) &= (L(a^*b^*))^* \\ &= (L(a^*)L(b^*))^* \\ &= (L(a)^*L(b)^*)^* \\ &= (\{a\}^*\{b\}^*)^* \\ &= (\{\lambda, a, aa, aaa, \dots\}\{\lambda, b, bb, bbb, \dots\})^* \\ &= \{\lambda, b, bb, \dots, a, ab, abb, \dots, aa, aab, \dots\}^* \\ &= \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\} \end{aligned}$$

Uma maneira mais formal de demonstrar a equivalência  $L((a + b)^*) = L((a^*b^*)^*) = \{a, b\}^*$  seria usando indução no tamanho da cadeia. A base da indução seria a cadeia vazia ( $\lambda$ ) a qual claramente pertence a ambas linguagens. A hipótese indutiva seria uma cadeia arbitaria  $w \in L((a + b)^*)$  e  $w \in L((a^*b^*)^*)$ . O passo indutivo seria mostrar que  $wa, wb \in L((a + b)^*)$  e  $wa, wb \in L((a^*b^*)^*)$ .

$$(5c) a^*b^* \not\equiv (ab)^*$$

**Resposta:** Como por definição  $L(a^*b^*) = L(a^*)L(b^*)$  resulta fácil de ver que  $b \in L(a^*b^*)$ , pois  $b = \lambda b$ ,  $\lambda \in L(a^*)$  e  $b \in L(b^*)$ . Por outro lado, como  $L((ab)^*) = \{ab\}^* = \{\lambda, ab, abab, ababab, \dots\}$  então  $b \notin L((ab)^*)$  e portanto  $L(a^*b^*) \neq L((ab)^*)$ .

$$(5e) aa^* \equiv a^*a$$

**Resposta:**

$$\begin{aligned}
 L(aa^*) &= L(a)L(a^*) \\
 &= \{a\}L(a)^* \\
 &= \{a\}\{a\}^* \\
 &= \{a\}\{\lambda, a, aa, aaa, \dots\} \\
 &= \{a, aa, aaa, aaaa, \dots\}
 \end{aligned}$$

$$\begin{aligned}
 L(a^*a) &= L(a^*)L(a) \\
 &= L(a)^*\{a\} \\
 &= \{a\}^*\{a\} \\
 &= \{\lambda, a, aa, aaa, \dots\}\{a\} \\
 &= \{a, aa, aaa, aaaa, \dots\}
 \end{aligned}$$

7. Achar autômatos finitos que aceitem as seguintes linguagens

**Resposta:** É claro que poderíamos usar o algoritmo do teorema 3.4.1, mas este é muito tedioso e longo, pelo que optamos por construir diretamente o AFN a partir da intuição da linguagem.

(7a)  $L(aa^*(a + b))$

**Resposta:** Um AFN que reconhece a linguagem  $L(aa^*(a + b))$  é ilustrado na figura A.33. Observe que neste autômato cada sub-expressão regular de  $aa^*(a + b)$  tem um segmento do AFN que “reconhece” essa expressão.

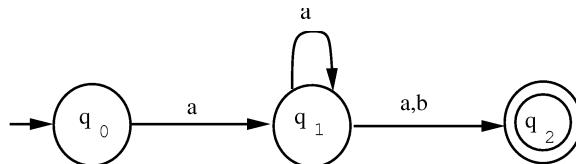


Figura A.33: AFN que reconhece a linguagem  $L(aa^*(a + b))$

(7c)  $L((ab + b)^*(a + \lambda))^*$

**Resposta:** Um AFN que reconhece a linguagem  $L((ab + b)^*(a + \lambda))^*$  é ilustrado na figura A.34 parte (A). No entanto, note que  $a \in L((ab + b)^*(a + \lambda))$ , pois  $\lambda \in L((ab + b)^*)$ ,  $a \in L(a + \lambda)$  e  $\lambda a = a$ . Analogamente,  $b \in L((ab + b)^*(a + \lambda))$ , pois  $b \in L((ab + b)^*)$ ,  $\lambda \in L(a + \lambda)$  e  $b\lambda = b$ . Logo,

$$L((ab + b)^*(a + \lambda)) = L((ab + b)^*(a + \lambda)) + a + b$$

Portanto,

$$L(((ab + b)^*(a + \lambda))^*) = L(((ab + b)^*(a + \lambda)) + a + b)^* = L((a + b)^*)$$

### A.3. Soluções de Exercícios do Capítulo 3

---

Logo, um AFN para a expressão regular equivalente  $(a + b)^*$  é ilustrado na figura A.34 parte (B). Claramente, ambos autômatos são equivalentes, isto é, reconhecem a mesma linguagem, mas o primeiro é inspirado na forma da expressão regular  $((ab + b)^*(a + \lambda))^*$  enquanto que o segundo é produto de uma análise da linguagem em questão.

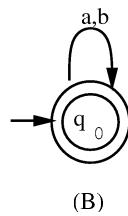
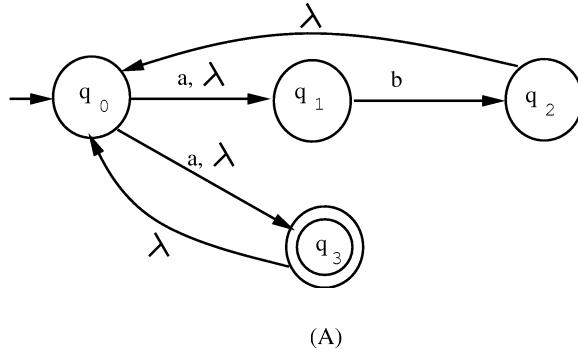


Figura A.34: AFN que reconhece a linguagem  $L((ab + b)^*(a + \lambda))^*$

(7e)  $L(aa^*bb^*aa^*)$

**Resposta:** Um AFN que reconhece a linguagem  $L(aa^*bb^*aa^*)$  é ilustrado na figura A.35.

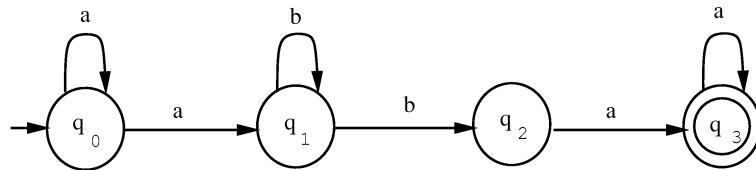


Figura A.35: AFN que reconhece a linguagem  $L(aa^*bb^*aa^*)$

(7g)  $L(a^*(b(bb)^*aa)^*)$

**Resposta:** Um AFN que reconhece a linguagem  $L(a^*(b(bb)^*aa)^*)$  é ilustrado na figura A.36.

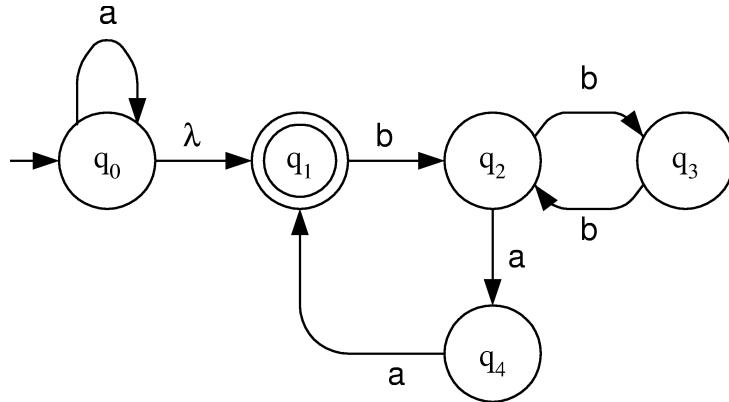


Figura A.36: AFN que reconhece a linguagem  $L(a^*(b(bb)^*aa)^*)$

8. Construir seguindo o algoritmo da seção 3.4.2 a expressão regular que denota a linguagem reconhecida pelo AFD da figura 3.14.

**Resposta:** Segundo esse algoritmo devemos achar  $R_{1,1}^2$ , mas para isso precisaremos achar outros  $R_{i,j}^k$  intermediários. Assim, a seguir acharei primeiro eles.

$$R_{1,1}^0 = \lambda$$

$$R_{1,2}^0 = 0 + 1$$

$$R_{2,1}^0 = 1$$

$$R_{2,2}^0 = \lambda + 0$$

$$\begin{aligned} R_{1,1}^1 &= R_{1,1}^0(R_{1,1}^0)^*R_{1,1}^0 + R_{1,1}^0 \\ &= \lambda(\lambda)^*\lambda + \lambda \\ &= \lambda \end{aligned}$$

$$\begin{aligned} R_{1,2}^1 &= R_{1,1}^0(R_{1,1}^0)^*R_{1,2}^0 + R_{1,2}^0 \\ &= \lambda(\lambda)^*(0+1) + (0+1) \\ &= 0+1 \end{aligned}$$

$$\begin{aligned} R_{2,1}^1 &= R_{2,1}^0(R_{1,1}^0)^*R_{1,1}^0 + R_{2,1}^0 \\ &= 1(\lambda)^*\lambda + 1 \\ &= 1 \end{aligned}$$

$$\begin{aligned} R_{2,2}^1 &= R_{2,1}^0(R_{1,1}^0)^*R_{1,2}^0 + R_{2,2}^0 \\ &= 1(\lambda)^*(0+1) + (\lambda+0) \\ &= 1(0+1) + \lambda + 0 \\ &= \lambda + 0 + 10 + 11 \end{aligned}$$

$$\begin{aligned} R_{1,1}^2 &= R_{1,2}^1(R_{2,2}^1)^*R_{2,1}^1 + R_{1,1}^1 \\ &= (0+1)(\lambda+0+10+11)^*1 + \lambda \end{aligned}$$

Observe que, pelo fato de ser um AFD simples, teria sido possível obter a expressão regular

### A.3. Soluções de Exercícios do Capítulo 3

---

$$((0+1)0^*1)^*$$

de forma direta se analisando o autômato. Embora não muito evidente, ambas expressões regulares são equivalentes.

9. Construir uma gramática linear à direita e uma linear à esquerda para as linguagens

(9a)  $L((aab^*abab)^*)$

**Resposta:** Uma gramática linear à direita que gera esta linguagem é definida por  $G = \langle \{S, B\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow aaB \mid \lambda$$

$$B \longrightarrow bB \mid ababS$$

Uma gramática linear à esquerda para esta mesma linguagem teria as seguintes produções:

$$S \longrightarrow Babab \mid \lambda$$

$$B \longrightarrow Bb \mid Saa$$

(9c)  $L((a+b)^*aaa)$

**Resposta:** Uma gramática linear à direita que gera esta linguagem é definida por  $G = \langle \{S\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow aS \mid bS \mid aaa$$

Uma gramática linear à esquerda para esta mesma linguagem teria as seguintes produções:

$$S \longrightarrow Aaaa$$

$$A \longrightarrow Aa \mid Ab \mid \lambda$$

(9e)  $L((a(aa)^*(bb)^*)^*aa^*)$

**Resposta:** Uma gramática linear à direita que gera esta linguagem é definida por  $G = \langle \{S, A, B, C\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow aA \mid aC$$

$$A \longrightarrow aaA \mid B$$

$$B \longrightarrow bbB \mid S$$

$$C \longrightarrow aC \mid a$$

Uma gramática linear à esquerda para esta mesma linguagem teria as seguintes produções:

$$S \longrightarrow Sa \mid Aa$$

$$A \longrightarrow B \mid \lambda$$

$$B \longrightarrow Bbb \mid C$$

$$C \longrightarrow Caa \mid Aa$$

11. Construir uma gramática regular que gere cada uma das seguintes linguagens sobre o alfabeto  $\{a, b\}$ .

(11a) Todas as cadeias que terminem com três  $a'$ s.

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** Uma gramática regular (linear à direita) que gera esta linguagem é definida por  $G = \langle \{S\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow aS \mid bS \mid aaa$$

- (11c) Todas as cadeias diferentes de  $(aaa)^k$ , para qualquer  $k \geq 0$ .

**Resposta:** Uma gramática regular (linear à direita) que gera esta linguagem é definida por  $G = \langle \{S, A\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow bA \mid aaaS \mid a \mid aa$$

$$A \longrightarrow aA \mid bA \mid \lambda$$

- (11e) Todas as cadeias da forma  $aawaa$ , onde  $|w|$  é múltiplo de três.

**Resposta:** Uma gramática regular (linear à direita) que gera esta linguagem é definida por  $G = \langle \{S, A\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções

$$S \longrightarrow aaA$$

$$A \longrightarrow aaaA \mid aabA \mid abaA \mid abbA \mid baaA \mid babA \mid bbaA \mid bbbA \mid aa$$

- (11g) Todas as cadeias com uma quantidade par de  $a$ 's mas sem qualquer ocorrência da subcadeia  $aaa$ .

**Resposta:** Uma gramática linear à direita que gera esta linguagem é definida por  $G = \langle \{S, A, B, C, D, E\}, \{a, b\}, S, P \rangle$ , onde  $P$  é composto das seguintes produções.

$$S \longrightarrow bS \mid aA \mid \lambda$$

$$A \longrightarrow bB \mid aE$$

$$B \longrightarrow bB \mid aC$$

$$C \longrightarrow bS \mid aD \mid \lambda$$

$$D \longrightarrow bB$$

$$E \longrightarrow bS \mid \lambda$$

Nesta gramática  $S$  representa a situação onde até esse momento foi gerado uma quantidade par de  $a$ 's e onde o último símbolo gerado não foi  $a$ .  $A$  representa a situação onde até esse momento foi gerado uma quantidade ímpar de  $a$ 's, o último símbolo foi  $a$  mas o antepenúltimo não foi  $a$ .  $B$  representa a situação onde até esse momento foi gerado uma quantidade ímpar de  $a$ 's e onde o último símbolo não foi  $a$ .  $C$  representa a situação onde até esse momento foi gerado uma quantidade par de  $a$ 's, o último símbolo foi  $a$  mas o antepenúltimo não foi  $a$ .  $D$  representa a situação onde até esse momento foi gerado uma quantidade ímpar de  $a$ 's, os dois últimos símbolos foram  $a$ 's mas o anterior a eles (caso haja algum) não foi  $a$ .  $E$  representa a situação onde até esse momento foi gerado uma quantidade par de  $a$ 's, os dois últimos símbolos foram  $a$ 's mas o anterior a eles (caso haja algum) não foi  $a$ .

12. Achar as gramáticas regulares para as linguagens reconhecidas pelos autômatos da figura 3.15 seguindo o algoritmo da subseção 3.6.2.

### A.3. Soluções de Exercícios do Capítulo 3

---

**Resposta:** As gramáticas regulares seguintes descrevem as linguagens reconhecidas pelos AFD's da figura 3.15.

- (12a) A gramática linear à direita resultante de aplicar o algoritmo da subseção 3.6.2 ao AFD da figura 3.15.a, é definida por  $G = \langle \{q_0, \dots, q_4\}, \{a, b\}, q_0, P \rangle$ , onde  $P$  é composto das seguintes produções.

$$\begin{aligned} q_0 &\longrightarrow bq_0 \mid aq_1 \\ q_1 &\longrightarrow aq_1 \mid bq_2 \\ q_2 &\longrightarrow bq_2 \mid aq_3 \\ q_3 &\longrightarrow aq_4 \mid bq_1 \mid \lambda \\ q_4 &\longrightarrow aq_4 \mid bq_4 \end{aligned}$$

- (12c) A gramática linear à direita resultante de aplicar o algoritmo da subseção 3.6.2 ao AFD da figura 3.15.c, é definida por  $G = \langle \{q_0, \dots, q_4\}, \{a, b\}, q_0, P \rangle$ , onde  $P$  é composto das seguintes produções.

$$\begin{aligned} q_0 &\longrightarrow aq_1 \mid bq_3 \\ q_1 &\longrightarrow aq_1 \mid bq_2 \mid \lambda \\ q_2 &\longrightarrow aq_1 \mid bq_2 \\ q_3 &\longrightarrow aq_4 \mid bq_3 \mid \lambda \\ q_4 &\longrightarrow aq_4 \mid bq_3 \end{aligned}$$

13. Construir um AFN, seguindo o algoritmo da subseção 3.6.1, para cada uma das seguintes gramáticas

$$\begin{aligned} (13a) \quad S &\longrightarrow abA; \\ A &\longrightarrow baB; \\ B &\longrightarrow aA \mid bb \end{aligned}$$

**Resposta:** O AFN ilustrado na figura A.37 reconhece a mesma linguagem gerada por esta gramática regular.

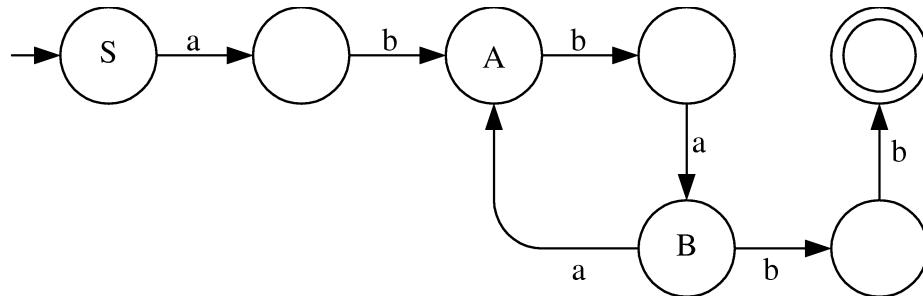


Figura A.37: AFN que reconhece a linguagem gerada pela gramática regular do exercício (13a).

$$\begin{aligned} (13c) \quad S &\longrightarrow aA \mid bS \mid \lambda; \\ A &\longrightarrow aB \mid bS \mid \lambda; \\ B &\longrightarrow aaS \mid bS \mid \lambda \end{aligned}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

**Resposta:** O AFN ilustrado na figura A.38 reconhece a mesma linguagem gerada por esta gramática regular.

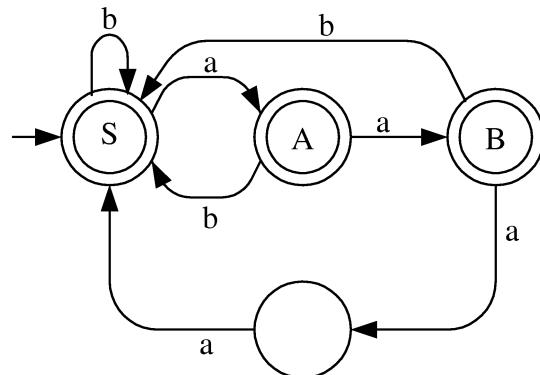


Figura A.38: AFN que reconhece a linguagem gerada pela gramática regular do exercício (13c).

## A.4. Soluções de Exercícios do Capítulo 4

---

### A.4 Soluções de Exercícios do Capítulo 4

1. Prove que toda linguagem finita é regular.

**Resposta:** Se  $L$  é uma linguagem finita sobre um alfabeto  $\Sigma$ , então  $L = \{w_1, \dots, w_n\}$  para algum  $n \geq 0$ . Cada  $w_i$  tem a seguinte forma:  $w_i = a_{i_1} \cdots a_{i_{k_i}}$ . Claramente o AFN  $M = \langle \{q_0, q_1, \dots, q_{1_{k_1}}, \dots, q_{n_1}, \dots, q_{n_{k_n}}\}, \Sigma, q_0, \delta, \{q_{1_{k_1}}, q_{2_{k_2}}, \dots, q_{n_{k_n}}\} \rangle$  onde para cada  $i = 1, \dots, n$ ,  $j = 1, \dots, k_i - 1$  e  $a \in \Sigma$ , temos que

$$\delta(q_0, a) = \{q_{i_1} : a_{i_1} = a\}$$

$$\delta(q_{i_j}, a) = \begin{cases} \emptyset & \text{se } a \neq a_{i_{j+1}} \\ \{q_{i_{j+1}}\} & \text{se } a = a_{i_{j+1}} \end{cases}$$

$$\delta(q_{i_{k_i}}, 0) = \emptyset$$

3. Quais das seguintes igualdades são verdadeiras para todas as linguagens regulares e todos os homomorfismos? Justifique.

$$(3a) h(\mathcal{L}_1 \cup \mathcal{L}_2) = h(\mathcal{L}_1) \cup h(\mathcal{L}_2)$$

**Resposta:** Se  $w \in h(\mathcal{L}_1 \cup \mathcal{L}_2)$ , então  $w = h(v)$  para algum  $v \in \mathcal{L}_1 \cup \mathcal{L}_2$ . Se  $v \in \mathcal{L}_1$ , então  $h(v) \in h(\mathcal{L}_1)$  e portanto  $w \in h(\mathcal{L}_1) \cup h(\mathcal{L}_2)$ . Analogamente, se  $v \in \mathcal{L}_2$ , então  $h(v) \in h(\mathcal{L}_2)$  e portanto  $w \in h(\mathcal{L}_1) \cup h(\mathcal{L}_2)$ .

Por outro lado, se  $w \in h(\mathcal{L}_1) \cup h(\mathcal{L}_2)$ , então  $w \in h(\mathcal{L}_1)$  ou  $w \in h(\mathcal{L}_2)$ . Se  $w \in h(\mathcal{L}_1)$ , então existe  $v \in \mathcal{L}_1$  tal que  $h(v) = w$ . Logo,  $v \in \mathcal{L}_1 \cup \mathcal{L}_2$  e  $h(v) = w \in h(\mathcal{L}_1 \cup \mathcal{L}_2)$ . Analogamente, se  $w \in h(\mathcal{L}_2)$ , então existe  $v \in \mathcal{L}_2$  tal que  $h(v) = w$ . Logo,  $v \in \mathcal{L}_1 \cup \mathcal{L}_2$  e  $h(v) = w \in h(\mathcal{L}_1 \cup \mathcal{L}_2)$ .

Portanto, esta igualdade é verdadeira.

$$(3b) h(\mathcal{L}_1 \cap \mathcal{L}_2) = h(\mathcal{L}_1) \cap h(\mathcal{L}_2)$$

**Resposta:** Se  $w \in h(\mathcal{L}_1 \cap \mathcal{L}_2)$ , então  $w = h(v)$  para algum  $v \in \mathcal{L}_1 \cap \mathcal{L}_2$  e portanto  $v \in \mathcal{L}_1$  e  $v \in \mathcal{L}_2$ . Assim,  $w \in h(\mathcal{L}_1)$  e  $w \in h(\mathcal{L}_2)$ . Logo,  $w \in h(\mathcal{L}_1) \cap h(\mathcal{L}_2)$ .

No entanto a reversa de esta igualdade não é correta, isto é se  $w \in h(\mathcal{L}_1) \cap h(\mathcal{L}_2)$ , não necessariamente é verdade que  $w \in h(\mathcal{L}_1 \cap \mathcal{L}_2)$ . Provaremos isto através do seguinte contra-exemplo: Seja  $\mathcal{L}_1 = \{ab\}$ ,  $\mathcal{L}_2 = \{ba\}$  e  $h(a) = h(b) = ab$ . Trivialmente,

$$\begin{aligned} h(\mathcal{L}_1) \cap h(\mathcal{L}_2) &= h(\{ab\}) \cap h(\{ba\}) \\ &= \{h(ab)\} \cap \{h(ba)\} \\ &= \{abab\} \cap \{abab\} \\ &= \{abab\} \end{aligned}$$

$$\begin{aligned} h(\mathcal{L}_1 \cap \mathcal{L}_2) &= h(\{ab\} \cap \{ba\}) \\ &= h(\emptyset) \\ &= \emptyset \end{aligned}$$

Por outro lado, se  $w \in h(\mathcal{L}_1) \cap h(\mathcal{L}_2)$ , então  $w \in h(\mathcal{L}_1)$  e  $w \in h(\mathcal{L}_2)$ . Logo,  $v \in \mathcal{L}_1 \cup \mathcal{L}_2$  e  $h(v) = w \in h(\mathcal{L}_1 \cup \mathcal{L}_2)$ . Analogamente, Se  $w \in h(\mathcal{L}_2)$ , então existe  $v \in \mathcal{L}_2$  tal que  $h(v) = w$ . Logo,  $v \in \mathcal{L}_1 \cup \mathcal{L}_2$  e  $h(v) = w \in h(\mathcal{L}_1 \cup \mathcal{L}_2)$ .

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

Assim esta igualdade não é verdadeira.

$$(3c) h(\mathcal{L}^n) = h(\mathcal{L})^n$$

**Resposta:** Demonstraremos por indução em  $n$  que esta igualdade é correta.

**Base da Indução:** Se  $n = 0$ , então  $\mathcal{L}^n = \mathcal{L}^0 = \{\lambda\}$ . Logo,

$$\begin{aligned} h(\mathcal{L}^0) &= h(\{\lambda\}) \\ &= \{h(\lambda)\} \\ &= \{\lambda\} \\ &= h(\{\lambda\})^0 \end{aligned}$$

**Hipótese Indutiva:**  $h(\mathcal{L}^i) = h(\mathcal{L})^i$ .

$$\begin{aligned} h(\mathcal{L}^{i+1}) &= h(\mathcal{L}^i \mathcal{L}) \\ &= h(\mathcal{L}^i)h(\mathcal{L}) \\ &= h(\mathcal{L})^i h(\mathcal{L}) \\ &= h(\mathcal{L})^{i+1} \end{aligned}$$

Logo, podemos concluir que  $h(\mathcal{L}^n) = h(\mathcal{L})^n$ .

$$(3d) h(\mathcal{L}^*) = h(\mathcal{L})^*$$

**Resposta:**  $w \in h(\mathcal{L}^*)$  se, e somente se, existe  $v \in \mathcal{L}^*$  tal que  $w = h(v)$ . Por definição de fecho estrela,  $v \in \mathcal{L}^*$  se, e somente se,  $v \in \mathcal{L}^n$ , para algum inteiro não negativo  $n$ .  $v \in \mathcal{L}^n$  se, e somente se,  $w \in h(\mathcal{L}^n)$ . Pelo item anterior temos que  $w \in h(\mathcal{L}^n)$  se, e somente se,  $w \in h(\mathcal{L})^n$ . Como,  $w \in h(\mathcal{L})^n$  se, e somente se,  $w \in h(\mathcal{L})^*$ , podemos concluir que  $h(\mathcal{L}^*) = h(\mathcal{L})^*$ .

$$(3e) h(\mathcal{L}^R) = h(\mathcal{L})^R$$

**Resposta:** Esta igualdade é incorreta, pois se  $\mathcal{L} = \{ab\}$  e  $h(a) = aab$  e  $h(b) = bba$ , então

$$\begin{aligned} h(\mathcal{L}^R) &= h(\{ab\}^R) \\ &= h(\{ba\}) \\ &= bbaaab \\ h(\mathcal{L})^R &= h(\{ab\})^R \\ &= (aabbba)^R \\ &= abbbaa \end{aligned}$$

Logo, neste caso,  $h(\mathcal{L}^R) \neq h(\mathcal{L})^R$

$$(3f) h(\overline{\mathcal{L}}) = \overline{h(\mathcal{L})}$$

**Resposta:** Esta igualdade é incorreta, pois se  $\mathcal{L} = \{ab\}$  e  $h(a) = a$  e  $h(b) = a$ , então

$$\begin{aligned} h(\overline{\mathcal{L}}) &= h(\{a, b\}^* - \{ab\}) \\ &= \{a\}^* \end{aligned}$$

#### A.4. Soluções de Exercícios do Capítulo 4

---

$$\begin{aligned}\overline{h(\mathcal{L})} &= \overline{h(\{ab\})} \\ &= \overline{\{aa\}} \\ &= \{a,b\}^* - \{aa\}\end{aligned}$$

Logo, neste caso  $h(\overline{\mathcal{L}}) \neq \overline{h(\mathcal{L})}$

4. Na prova do teorema 4.1.8, mostre que  $h(r)$  é uma expressão regular. Então, mostre que  $h(r)$  denota  $h(\mathcal{L})$ .

**Resposta:** Provaremos por indução em  $r$  que  $h(r)$  é uma expressão regular.

**Base da Indução:** Se  $r$  é uma expressão regular primitiva, isto é,  $r = \lambda$ ,  $r = \emptyset$  ou  $r = a$  para algum  $a \in \Sigma_1$ , então  $h(r)$  ou é  $\lambda$ , ou é  $\emptyset$  ou é uma cadeia  $w \in \Sigma_2$ , respectivamente, em todos os casos  $h(r)$  é um expressão regular.

**Hipóteses Indutiva:** Assuma que  $r_1$  e  $r_2$  são expressões regulares tais que  $h(r_1)$  e  $h(r_2)$  também são expressões regulares.

**Passo Indutivo:** Se  $r = r_1 + r_2$ , então

$$\begin{aligned}h(r) &= h(r_1 + r_2) \\ &= h(r_1) + h(r_2)\end{aligned}$$

que é uma expressão regular.

Se  $r = r_1r_2$ , então

$$\begin{aligned}h(r) &= h(r_1r_2) \\ &= h(r_1)h(r_2)\end{aligned}$$

que é uma expressão regular.

Se  $r = r_1^*$ , então

$$\begin{aligned}h(r) &= h(r_1^*) \\ &= h(r_1)^*\end{aligned}$$

que é uma expressão regular.

Assim podemos concluir que  $h(r)$  é uma expressão regular sempre que  $r$  é uma expressão regular.

Por outro lado, devemos mostrar que  $h(r)$  denota  $h(\mathcal{L})$ . Isto implica em mostrar que se  $v \in L(h(r))$ , então  $v = h(w)$  para algum  $w \in L(r)$  e vice-versa, isto é que se  $w \in L(r)$  então  $h(w) \in L(h(r))$ . Mas isto equivale a mostrar que  $L(h(r)) = h(L(r))$ . Faremos isto por indução na complexidade da expressão regular  $r$ .

**Base da Indução:** Seja  $r$  é uma expressão regular primitiva.

Se  $r$  é  $\lambda$  então

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
 L(h(\lambda)) &= L(\lambda) \\
 &= \{\lambda\} \\
 &= \{h(\lambda)\} \\
 &= h(\{\lambda\}) \\
 &= h(L(\lambda)).
 \end{aligned}$$

Se  $r$  é  $\emptyset$  então

$$\begin{aligned}
 L(h(\emptyset)) &= L(\emptyset) \\
 &= \emptyset \\
 &= h(\emptyset) \\
 &= h(L(\emptyset)).
 \end{aligned}$$

Se  $r$  é um  $a \in \Sigma_1$  então  $L(r) = \{a\}$  e portanto

$$\begin{aligned}
 L(h(a)) &= \{h(a)\} \\
 &= h(\{a\}) \\
 &= h(L(a)).
 \end{aligned}$$

Logo,  $L(h(r)) = h(L(r))$  quando  $r$  é uma expressão regular primitiva.

**Hipóteses Indutiva:** Assuma que  $r_1$  e  $r_2$  são expressões regulares tais que  $L(h(r_1)) = h(L(r_1))$  e  $L(h(r_2)) = h(L(r_2))$ .

**Passo Indutivo:** Se  $r = r_1 + r_2$ , então

$$\begin{aligned}
 L(h(r)) &= L(h(r_1 + r_2)) \\
 &= L(h(r_1) + h(r_2)) \\
 &= L(h(r_1)) \cup L(h(r_2)) \\
 &= h(L(r_1)) \cup h(L(r_2)) \\
 &= h(L(r_1) \cup L(r_2)) \\
 &= h(L(r_1 + r_2)).
 \end{aligned}$$

Se  $r = r_1 r_2$ , então

$$\begin{aligned}
 L(h(r)) &= L(h(r_1 r_2)) \\
 &= L(h(r_1)h(r_2)) \\
 &= L(h(r_1))L(h(r_2)) \\
 &= h(L(r_1))h(L(r_2)) \\
 &= h(L(r_1)L(r_2)) \\
 &= h(L(r_1 r_2)).
 \end{aligned}$$

Se  $r = r_1^*$ , então

$$\begin{aligned}
 L(h(r)) &= L(h(r_1^*)) \\
 &= L(h(r_1))^* \\
 &= L(h(r_1))^* \\
 &= h(L(r_1))^* \\
 &= h(L(r_1^*)).
 \end{aligned}$$

Logo, podemos concluir que  $L(h(r)) = h(L(r))$  para qualquer expressão regular  $r$ .

5. Mostre que a família das linguagens regulares é fechada sobre união finita e intersecção finita, isto é, se  $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n$  são linguagens regulares então

#### A.4. Soluções de Exercícios do Capítulo 4

---

$$\bigcup_{i=1}^n \mathcal{L}_i \quad \text{e} \quad \bigcap_{i=1}^n \mathcal{L}_i$$

também são.

**Resposta:** Provaremos por indução em  $n$  o caso da união finita.

**Base da Indução:** Se  $n = 1$ , então  $\bigcup_{i=1}^n \mathcal{L}_i = \bigcup_{i=1}^1 \mathcal{L}_i = \mathcal{L}_1$ . Que claramente é regular.

**Hipóteses Indutiva:** Assuma que  $\bigcup_{i=1}^n \mathcal{L}_i$  é regular.

**Passo Indutivo:** Provaremos que  $\bigcup_{i=1}^{n+1} \mathcal{L}_i$  é regular. Como

$$\bigcup_{i=1}^{n+1} \mathcal{L}_i = \left( \bigcup_{i=1}^n \mathcal{L}_i \right) \cup \mathcal{L}_{n+1}$$

então  $\bigcup_{i=1}^{n+1} \mathcal{L}_i$  é uma união de duas linguagens regulares. Logo, pelo teorema 4.1.1,  $\bigcup_{i=1}^{n+1} \mathcal{L}_i$  é uma linguagem regular.

6. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  duas linguagens sobre os alfabetos  $\Sigma_1$  e  $\Sigma_2$ , respectivamente. O NEM de duas linguagens é

$$NEM(\mathcal{L}_1, \mathcal{L}_2) = \{w \in (\Sigma_1 \cup \Sigma_2)^* / w \notin \mathcal{L}_1 \text{ e } w \notin \mathcal{L}_2\}.$$

Mostre que a família das linguagens regulares é fechada sobre NEM.

**Resposta:** Primeiro mostraremos passo a passo que  $NEM(\mathcal{L}_1, \mathcal{L}_2) = \overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}$ .

$$\begin{aligned} NEM(\mathcal{L}_1, \mathcal{L}_2) &= \{w \in (\Sigma_1 \cup \Sigma_2)^* / w \notin \mathcal{L}_1 \text{ e } w \notin \mathcal{L}_2\} \\ &= \{w \in (\Sigma_1 \cup \Sigma_2)^* / \neg(w \in \mathcal{L}_1) \text{ e } \neg(w \in \mathcal{L}_2)\} \\ &= \{w \in (\Sigma_1 \cup \Sigma_2)^* / \neg(w \in \mathcal{L}_1 \text{ ou } w \in \mathcal{L}_2)\} \\ &= \overline{\{w \in (\Sigma_1 \cup \Sigma_2)^* / w \in \mathcal{L}_1 \text{ ou } w \in \mathcal{L}_2\}} \\ &= \overline{\{w \in (\Sigma_1 \cup \Sigma_2)^* / w \in \mathcal{L}_1\}} \cup \overline{\{w \in (\Sigma_1 \cup \Sigma_2)^* / w \in \mathcal{L}_2\}} \\ &= \overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2} \end{aligned}$$

Como já foi provado que as linguagens regulares são fechadas sobre complemento e união, podemos concluir que se  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são linguagens regulares então  $NEM(\mathcal{L}_1, \mathcal{L}_2)$  também é uma linguagem regular.

- 7 Mostre que as linguagens regulares são fechadas sobre a união disjunta, onde a união disjunta de dois conjuntos (linguagens)  $\mathcal{L}_1$  e  $\mathcal{L}_2$  é definida por

$$\mathcal{L}_1 \uplus \mathcal{L}_2 = \{w0 / w \in \mathcal{L}_1\} \cup \{w1 / w \in \mathcal{L}_2\}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
 \mathcal{L}_1 \uplus \mathcal{L}_2 &= \{w0 / w \in \mathcal{L}_1\} \cup \{w1 / w \in \mathcal{L}_2\} \\
 \text{Resposta:} \quad &= \{w / w \in \mathcal{L}_1\}\{0\} \cup \{w / w \in \mathcal{L}_2\}\{1\} \\
 &= \mathcal{L}_1\{0\} \uplus \mathcal{L}_2\{1\}
 \end{aligned}$$

Como  $\{0\}$  e  $\{1\}$  são trivialmente linguagens regulares, e a classe das linguagens regulares são fechadas sobre união e concatenação, temos que se  $\mathcal{L}_1$  e  $\mathcal{L}_2$  são linguagens regulares então sua união disjunta, isto é  $\mathcal{L}_1 \uplus \mathcal{L}_2$ , também é uma linguagem regular.

10. Seja  $\mathcal{L}$  uma linguagem sobre um alfabeto  $\Sigma$ . Conforme visto no capítulo 1, a linguagem de **sufixos** de  $\mathcal{L}$ , denotada por  $\mathcal{L}^S$ , é definida como

$$\mathcal{L}^S = \{w \in \Sigma^* / vw \in \mathcal{L} \text{ para algum } v \in \Sigma^*\}.$$

Demonstre que a classe das linguagens regulares é fechada sobre sufixos.

**Resposta:** Primeiro mostraremos que para qualquer linguagem  $\mathcal{L}$  (regular ou não),  $\mathcal{L}^S = ((\mathcal{L}^R)^P)^R$ , onde  $\mathcal{L}^R$  é linguagem reversa de  $\mathcal{L}$  e  $\mathcal{L}^P$  é a linguagem de prefixos de  $\mathcal{L}$ .

$$\begin{aligned}
 \mathcal{L}^S &= \{w \in \Sigma^* / vw \in \mathcal{L} \text{ para algum } v \in \Sigma^*\} \\
 &= \{w \in \Sigma^* / ((vw)^R)^R \in \mathcal{L} \text{ para algum } v \in \Sigma^*\} \\
 &= \{w \in \Sigma^* / (w^Rv^R)^R \in \mathcal{L} \text{ para algum } v \in \Sigma^*\} \\
 &= \{w \in \Sigma^* / w^Rv^R \in \mathcal{L}^R \text{ para algum } v \in \Sigma^*\} \\
 &= \{w^R \in \Sigma^* / w^Rv^R \in \mathcal{L}^R \text{ para algum } v \in \Sigma^*\}^R \\
 &= \{w \in \Sigma^* / vv \in \mathcal{L}^R \text{ para algum } v \in \Sigma^*\}^R \\
 &= ((\mathcal{L}^R)^P)^R
 \end{aligned}$$

Como a classe das linguagens regulares é fechada sobre prefixos e reversa, então podemos concluir que se  $\mathcal{L}$  é regular então  $\mathcal{L}^S$  também é regular.

13. Sejam  $\mathcal{L}_1$  e  $\mathcal{L}_2$  linguagens sobre alfabetos  $\Sigma_1$  e  $\Sigma_2$ , respectivamente. Defina o operador  $\wedge_2$  por

$$\mathcal{L}_1 \wedge_2 \mathcal{L}_2 = \{uv / u \in \mathcal{L}_1 \cup \mathcal{L}_2 \text{ e } v \in (\Sigma_1 \cup \Sigma_2)^*\}$$

Mostre que as linguagens regulares são fechadas sobre a operação  $\wedge_2$  entre linguagens.

**Resposta:** Sejam  $G_1 = \langle V_1, \Sigma_1, S_1, P_1 \rangle$  e  $G_2 = \langle V_2, \Sigma_2, S_2, P_2 \rangle$  duas gramáticas lineares à direita tais que  $L(G_1) = \mathcal{L}_1$  e  $L(G_2) = \mathcal{L}_2$ . Seja  $X, S \notin V_1 \cup V_2$ . Sem perca de generalidade podemos considerar que  $V_1 \cap V_2 = \emptyset$ . Claramente, a gramática  $G_1 \wedge_2 G_2 = \langle V_1 \cup V_2 \cup \{S, X\}, \Sigma_1 \cup \Sigma_2, S, P_1 \wedge_X P_2 \rangle$  onde

$P_1 \wedge_X P_2 = \{A \longrightarrow x \in P_1 \cup P_2 / x \notin (\Sigma_1 \cup \Sigma_2)^*\} \cup \{A \longrightarrow xX / A \longrightarrow x \in P_1 \cup P_2 \text{ e } x \in (\Sigma_1 \cup \Sigma_2)^*\} \cup \{S \longrightarrow S_1, S \longrightarrow S_2\} \cup \{X \longrightarrow aX / a \in \Sigma_1 \cup \Sigma_2\} \cup \{X \longrightarrow \lambda\}$ , é tal que  $L(G_1 \wedge_2 G_2) = L(G_1) \wedge_2 L(G_2) = \mathcal{L}_1 \wedge_2 \mathcal{L}_2$ .

Por exemplo, se  $P_1$  consistir das seguintes produções:

$$\begin{aligned}
 S_1 &\longrightarrow aaS_1 \mid bA \\
 A &\longrightarrow bbA \mid aaS_1 \mid \lambda
 \end{aligned}$$

#### A.4. Soluções de Exercícios do Capítulo 4

---

e  $P_2$  consistir das seguintes produções:

$$S_2 \longrightarrow aS_2 \mid bS_2 \mid abB$$

$$B \longrightarrow baB \mid C$$

$$C \longrightarrow ccC \mid cc$$

Então  $P_1 \wedge_X P_2$  consistiria das seguintes produções:

$$S \longrightarrow S_1 \mid S_2$$

$$S_1 \longrightarrow aaS_1 \mid bA$$

$$A \longrightarrow bbA \mid aaS_1 \mid X$$

$$S_2 \longrightarrow aS_2 \mid bS_2 \mid abB$$

$$B \longrightarrow baB \mid C$$

$$C \longrightarrow ccC \mid ccX$$

$$X \longrightarrow aX \mid bX \mid cX \mid \lambda$$

15. Mostre que existe um algoritmo para determinar se  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , para qualquer linguagem regular  $\mathcal{L}_1$  e  $\mathcal{L}_2$ .

**Resposta:** Observe que em conjuntos  $A \subseteq B$  se e somente se  $\overline{B} \cap A = \emptyset$ . Como as linguagens regulares são fechadas sobre complemento e intersecção, então  $\overline{\mathcal{L}_2} \cap \mathcal{L}_1$  é uma linguagem regular. De fato seria possível construir algoritmamente um AFD que reconheça essa linguagem baseada em AFD's que reconhecem  $\mathcal{L}_1$  e  $\mathcal{L}_2$ . Agora aplicamos o algoritmo para detectar se um AFD reconhece uma linguagem vazia ou não ao AFD que reconhece  $\overline{\mathcal{L}_2} \cap \mathcal{L}_1$ . Se tivermos como resposta “sim” então  $\mathcal{L}_1 \subseteq \mathcal{L}_2$  e caso contrário  $\mathcal{L}_1 \not\subseteq \mathcal{L}_2$ .

17. Seja a tabela 4.1. A primeira coluna representa os rótulos de cada fila na tabela enquanto a primeira fila representa os rótulos de cada coluna na tabela. Para cada  $x, y \in \Sigma = \{a, b, c\}$  denote por  $T(x, y)$  o conteúdo da tabela para a posição  $(x, y)$ . Seja a seguinte função  $A : \Sigma^+ \rightarrow \Sigma$  definida por

- $A(a) = a$ , para cada  $a \in \Sigma$  e
- $A(wa) = T(A(w), a)$ , para cada  $w \in \Sigma^+$  e  $a \in \Sigma$ .

Demonstre que a seguinte linguagem é regular:

$$\mathcal{L} = \{w \in \Sigma^+ / A(w) = A(w^R)\}$$

**Resposta:** Primeiro devemos entender bem o problema. A notação  $T(b, a)$  é o conteúdo da tabela 4.1 na posição da linha  $b$  com a coluna  $a$  e portanto  $T(b, a) = c$ , já  $T(a, b) = a$ . Assim,

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
A(aababcc) &= T(A(aababc), c) \\
&= T(T(A(aabab), c), c) \\
&\quad \vdots \\
&= T(T(T(T(T(a, a), b), a), b), c), c) \\
&= T(T(T(T(a, b), a), b), c), c) \\
&= T(T(T(T(a, a), b), c), c) \\
&= T(T(T(a, b), c), c) \\
&= T(T(a, c), c) \\
&= T(c, c) \\
&= a
\end{aligned}$$

Por outro lado,

$$\begin{aligned}
A((aababcc)^R) &= A(ccbabaa) \\
&= T(A(ccbaba), a) \\
&= T(T(A(ccbab), a), a) \\
&\quad \vdots \\
&= T(T(T(T(T(c, c), b), a), b), a), a) \\
&= T(T(T(T(a, b), a), b), a), a) \\
&= T(T(T(T(a, a), b), a), a) \\
&= T(T(T(a, b), a), a) \\
&= T(T(a, a), a) \\
&= T(a, a) \\
&= a
\end{aligned}$$

Portanto,  $A(aababcc) = A(ccbabaa)$  o que implica que  $aababcc \in \mathcal{L}$ .

Como, trivialmente, para todo palíndromo  $w$  temos que  $A(w) = A(w^R)$ , então esta linguagem contém propriamente os palíndromos e portanto é infinita. Observe que o lema do bombeamento não pode ser usado para provar que  $\mathcal{L}$  é regular, pois ele só garante que  $\mathcal{L}$  for regular então satisfaz as condições, mas só fato de satisfazer as condições não implica em que seja regular.

Certamente esta linguagem não é fácil de descrever sem ter que recorrer à tabela associativa por tanto a missão de achar um autômato finito para esta linguagem de forma direta é não trivial. Porém, aqui só faremos uma prova de que existe tal autômato finito. Sejam os AFD da figura A.39. Claramente, eles reconhecem as seguintes linguagens

- $L_a = \{w \in \{a, b, c\} / A(w) = A(w^R) = a\}$
- $L_b = \{w \in \{a, b, c\} / A(w) = A(w^R) = b\}$
- $L_c = \{w \in \{a, b, c\} / A(w) = A(w^R) = c\}$

Por outro lado, é fácil ver que

$$\mathcal{L} = (L_a \cap L_a^R) \cup (L_b \cap L_b^R) \cup (L_c \cap L_c^R).$$

Logo, como as linguagens regulares são fechadas sobre intersecção, união e reversa, então  $\mathcal{L}$  é uma linguagem regular.

#### A.4. Soluções de Exercícios do Capítulo 4

---

Esta prova só mostra a existência, mas a partir dela com algo de esforço seria possível construir um AFD que reconheça  $\mathcal{L}$ .

18. Seja  $\Sigma = \{0, 1\}$ . Mostre que é possível aplicar o lema do bombeamento para as seguintes linguagens regulares.

$$(18a) \mathcal{L} = \{w \in \Sigma^* / 0110 \text{ é um prefixo de } w\}$$

**Resposta:** Seja  $m = 5$ ,  $w \in \mathcal{L}$  tal que  $|w| \geq m$ ,  $x = 0110$ , e  $y$  com  $z$  cadeias tais que  $w = xyz$  e  $|y| = 1$ . Claramente  $|xy| = m$ . Claramente, para cada  $i = 0, 1, \dots$ , temos que  $xy^i z = 0110y^i z \in \mathcal{L}$ .

$$(18c) \mathcal{L} = \{w \in \Sigma^* / w = u111v \text{ para algum } u, v \in \Sigma^*\}$$

**Resposta:** Seja  $m = 5$ ,  $w \in \mathcal{L}$  tal que  $|w| \geq m$ . Como,  $w = u111v$ , se  $u = \lambda$  então considere  $x = 111$ , e  $y$  com  $z$  cadeias tais que  $w = xyz$  e  $|y| = 1$ . Claramente  $|xy| < m$ . Claramente, para cada  $i = 0, 1, \dots$ , temos que  $xy^i z = 111y^i z \in \mathcal{L}$ . Se  $u \neq \lambda$ , então considere  $x = \lambda$ , e  $y$  com  $z$  cadeias tais que  $w = xyz$  e  $|y| = 1$ . Assim, a subcadeia 111 ocorre em  $z$  e portanto  $z = u111v$  para algum  $u, v \in \Sigma^*$ . Logo,  $xy^i z = y^i z = y^i u111v \in \mathcal{L}$ .

19. Mostre que as linguagens sobre  $\Sigma = \{a, b\}$ , definidas a seguir, não são regulares.

$$(19a) \mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}$$

**Resposta:** Para demonstrar que esta linguagem não é regular, devemos mostrar que ela não satisfaz o lema do bombeamento, ou analogamente que ela satisfaz o corolário do lema do bombeamento (corolário 4.3.6). Claramente  $\mathcal{L}$  é infinita.

Seja  $m$  um inteiro positivo qualquer e  $w = a^m b^m$ . Claramente  $w \in \mathcal{L}$ . Seja,  $x, y$  e  $z$  cadeias tais que  $w = xyz$ ,  $|xy| \leq m$  e  $|y| \geq 1$ . Então,  $x = a^i$  e  $y = a^j$  (portanto  $z = a^{m-(i+j)}b^m$ ), para algum  $i$  e  $j$  satisfazendo  $i + j \leq m$  e  $j \geq 1$ . A cadeia  $xy^2z \notin \mathcal{L}$ , pois  $xy^2z = a^i a^j a^j a^{m-(i+j)}b^m = a^{m+j}b^m$ . Como  $j \geq 1$ , então  $m + j > m$  e portanto  $xy^2z = a^{m+j}b^m \notin \mathcal{L}$ .

$$(19b) \mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \neq \mathcal{N}_b(w)\}$$

**Resposta:** Note que esta linguagem é o complemento da linguagem do exercício anterior (Exercício 19a). Como linguagens regulares são fechadas sobre complemento então se  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \neq \mathcal{N}_b(w)\}$  for regular a linguagem do Exercício 19a também seria regular, mas já foi provado que ela não é regular, chegando a uma contradição. Logo, podemos concluir que  $\mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) \neq \mathcal{N}_b(w)\}$  não é regular.

Para demonstrar que esta linguagem não é regular usando diretamente o corolário do lema do bombeamento (corolário 4.3.6) considere um inteiro positivo  $m$  qualquer e  $w = a^m b^{2m!}$ . Claramente  $w \in \mathcal{L}$ . Seja,  $x, y$  e  $z$  cadeias tais que  $w = xyz$ ,  $|xy| \leq m$  e  $|y| \geq 1$ . Então,  $x = a^i$  e  $y = a^j$  (portanto  $z = a^{m-(i+j)}b^{m!+m}$ ), para algum  $i$  e  $j$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

satisfazendo  $i + j \leq m$  e  $j \geq 1$ . Note que  $\frac{m!}{j} + 1 = 1 + \prod_{k=1}^{j-1} k \times \prod_{k'=j+1}^m k'$  e portanto é um inteiro positivo. Mas,

$$\begin{aligned} xy^{\frac{m!}{j}} z &= a^i (a^j)^{\frac{m!}{j}+1} a^{m-(i+j)} b^{m!+m} \\ &= a^i a^{j \times (\frac{m!}{j}+1)} a^{m-(i+j)} b^{m!+m} \\ &= a^i a^{m!+j} a^{m-(i+j)} b^{m!+m} \\ &= a^{i+m!+j+m-(i+j)} b^{m!+m} \\ &= a^{m!+m} b^{m!+m} \notin \mathcal{L}. \end{aligned}$$

Portanto,  $\mathcal{L}$  não é uma linguagem regular.

(19d)  $\mathcal{L} = \{a^m b^n / m > n\}$

**Resposta:** Para demonstrar que esta linguagens não é regular, devemos mostrar que ela não satisfaz o lema do bombeamento, ou analogamente que ela satisfaz o corolário do lema do bombeamento (corolário 4.3.6). Claramente  $\mathcal{L}$  é infinita.

Seja  $m$  um inteiro positivo qualquer e  $w = a^m b^{m-1}$ . Claramente  $w \in \mathcal{L}$ . Seja,  $x, y$  e  $z$  cadeias tais que  $w = xyz$ ,  $|xy| \leq m$  e  $|y| \geq 1$ . Então,  $x = a^i$  e  $y = a^j$  (portanto  $z = a^{m-(i+j)} b^{m-1}$ ), para algum  $i$  e  $j$  satisfazendo  $i + j \leq m$  e  $j \geq 1$ . A cadeia  $xy^0 z \notin \mathcal{L}$ , pois  $xy^0 z = a^i a^{m-(i+j)} b^{m-1} = a^{m-j} b^{m-1}$ . Como  $j \geq 1$ , então  $m - j \leq m - 1$  e portanto  $xy^0 z = a^{m-j} b^{m-1} \notin \mathcal{L}$ .

(19e)  $\mathcal{L} = \{a^m b^n / m \neq n\}$

**Resposta:** Para demonstrar que esta linguagens não é regular, devemos mostrar que ela não satisfaz o lema do bombeamento, ou analogamente que ela satisfaz o corolário do lema do bombeamento (corolário 4.3.6). Claramente  $\mathcal{L}$  é infinita.

Seja  $m$  um inteiro positivo qualquer e  $w = a^m b^{m!+m}$ . Claramente  $w \in \mathcal{L}$ . Seja,  $x, y$  e  $z$  cadeias tais que  $w = xyz$ ,  $|xy| \leq m$  e  $|y| \geq 1$ . Então,  $x = a^i$  e  $y = a^j$  (portanto  $z = a^{m-(i+j)} b^{m!+m}$ ), para algum  $i$  e  $j$  satisfazendo  $i + j \leq m$  e  $j \geq 1$ . A cadeia  $xy^{\frac{m!}{j}+1} z \notin \mathcal{L}$ , pois

$$\begin{aligned} xy^{\frac{m!}{j}+1} z &= a^i a^{j(\frac{m!}{j}+1)} a^{m-(i+j)} b^{m!+m} \\ &= a^i a^{j(\frac{m!}{j}+1)} a^{m-(i+j)} b^{m!+m} \\ &= a^{m-j} a^{m!+j} b^{m!+m} \\ &= a^m a^{m!} b^{m!+m} \\ &= a^{m!+m} b^{m!+m}. \end{aligned}$$

21. Seja  $\Sigma = \{a, b\}$  e  $\widehat{\cdot}: \Sigma^* \rightarrow \Sigma^*$  a operação definida recursivamente a seguir:

- $\widehat{\lambda} = \lambda$
- $\widehat{wa} = \widehat{w}aa$
- $\widehat{wb} = \widehat{w}bb$

Mostre usando o lema do bombeamento que a linguagem  $\mathcal{L} = \{w\widehat{w} / w \in \Sigma^*\}$  não é regular.

#### A.4. Soluções de Exercícios do Capítulo 4

---

**Resposta:** Para demonstrar que esta linguagens não é regular, devemos mostrar que ela não satisfaz o lema do bombeamento, ou analogamente que ela satisfaz o corolário do lema do bombeamento (corolário 4.3.6). Claramente  $\mathcal{L}$  é infinita.

Seja  $m$  um inteiro positivo qualquer e  $w = a^m b$ . Claramente a cadeia  $w\hat{w} = a^m b b^m a \in \mathcal{L}$ . Seja,  $x, y$  e  $z$  cadeias tais que  $w\hat{w} = xyz$ ,  $|xy| \leq m$  e  $|y| \geq 1$ . Então,  $x = a^i$  e  $y = a^j$  (portanto  $z = a^{m-(i+j)} b b^m a$ ), para algum  $i$  e  $j$  satisfazendo  $i + j \leq m$  e  $j \geq 1$ . A cadeia  $xy^2z \notin \mathcal{L}$ , pois  $\underline{xy^2z} = a^i a^j a^j a^{m-(i+j)} b b^m a = a^{m+j} b b^m a$ . Como  $j \geq 1$ , então  $m + j > m$  e portanto  $a^{m+j} b \neq b^m a$ . Logo,  $a^{m+j} b b^m a \notin \mathcal{L}$ .

## A.5 Soluções de Exercícios do Capítulo 5

3. Achar as gramáticas livres do contexto para as seguintes linguagens (com  $m \geq 0$ ,  $n \geq 0$  e  $k \geq 0$ ).

$$(3a) \mathcal{L} = \{a^n b^m / n \neq m - 1\},$$

**Resposta:** Observe que

$$\begin{aligned}\mathcal{L} &= \{a^n b^m / n \neq m - 1\} \\ &= \{a^n b^m / n > m - 1\} \cup \{a^n b^m / n < m - 1\} \\ &= \{a^n b^m / n \geq m\} \cup \{a^n b^m / n + 2 \leq m\} \\ &= \{a^n b^m / n \geq m\} \cup \{a^n b^k b b / n \leq k\}\end{aligned}$$

Assim,  $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ , onde  $\mathcal{L}_1 = \{a^n b^m / n \geq m\}$  e  $\mathcal{L}_2 = \{a^n b^k b b / n \leq k\}$

Uma gramática livre do contexto que gera a linguagem  $\mathcal{L}_1$  é a seguinte

$$\begin{aligned}S_1 &\longrightarrow aS_1b \mid A \\ A &\longrightarrow aA \mid \lambda\end{aligned}$$

Por outro lado, uma gramática livre do contexto que gera a linguagem  $\mathcal{L}_2$  é a seguinte

$$\begin{aligned}S_2 &\longrightarrow aS_2b \mid B \\ B &\longrightarrow bB \mid bb\end{aligned}$$

A união destas duas linguagens se faz simplesmente (pois não tem variáveis em comum) juntando ambas gramáticas e adicionando uma nova variável inicial e duas produções começando dela, uma para  $S_1$  e a outra para  $S_2$ . Assim, o resultado desta junção é a gramática livre do contexto:

$$\begin{aligned}S &\longrightarrow S_1 \mid S_2 \\ S_1 &\longrightarrow aS_1b \mid A \\ A &\longrightarrow aA \mid \lambda \\ S_2 &\longrightarrow aS_2b \mid B \\ Blra &\quad bB \mid bb\end{aligned}$$

$$(3c) \mathcal{L} = \{a^n b^m / n \leq m + 3\},$$

**Resposta:**

$$\begin{aligned}S &\longrightarrow aSb \mid A \\ A &\longrightarrow Ab \mid aaa\end{aligned}$$

$$(3e) \mathcal{L} = \{a^n b^m c^k / n = m \text{ ou } m \leq k\},$$

**Resposta:** Esta linguagem é a união das linguagem  $\mathcal{L}_1$  e  $\mathcal{L}_2$ , onde  $\mathcal{L}_1 = \{a^n b^m c^k / n = m\}$  e  $\mathcal{L}_2 = \{a^n b^m c^k / m \leq k\}$ . Assim, uma gramática livre do contexto para esta linguagem resulta da união das respectivas gramáticas livres do contexto para  $\mathcal{L}_1$  e  $\mathcal{L}_2$ .

Uma gramática livre do contexto que gera  $\mathcal{L}_1$  é a seguinte:

## A.5. Soluções de Exercícios do Capítulo 5

---

$$\begin{array}{l} S_1 \longrightarrow BC \\ B \longrightarrow aBb \mid \lambda \\ C \longrightarrow cC \mid \lambda \end{array}$$

Uma gramática livre do contexto que gera  $\mathcal{L}_2$  é a seguinte:

$$\begin{array}{l} S_2 \longrightarrow AD \\ A \longrightarrow aA \mid \lambda \\ D \longrightarrow bDc \mid E \\ E \longrightarrow cE \mid \lambda \end{array}$$

Portanto, como ambas gramáticas não têm variáveis em comum, uma gramática livre do contexto que gera  $\mathcal{L}_1 \cup \mathcal{L}_2$ , isto é  $\mathcal{L}$ , é a seguinte:

$$\begin{array}{l} S \longrightarrow S_1 \mid S_2 \\ S_1 \longrightarrow BC \\ B \longrightarrow aBb \mid \lambda \\ C \longrightarrow cC \mid \lambda \\ S_2 \longrightarrow AD \\ A \longrightarrow aA \mid \lambda \\ D \longrightarrow bDc \mid E \\ E \longrightarrow cE \mid \lambda \end{array}$$

(3g)  $\mathcal{L} = \{a^n b^m c^k / n = m \text{ ou } m \neq k\}$ ,

**Resposta:** Esta linguagem é a união das linguagens:  $\mathcal{L}_1 = \{a^n b^m c^k / n = m\}$ ,  $\mathcal{L}_2 = \{a^n b^m c^k / m < k\}$  e  $\mathcal{L}_3 = \{a^n b^m c^k / m > k\}$ . Uma gramática livre do contexto para  $\mathcal{L}_1$  foi dada no exercício anterior, já as gramáticas livres do contexto de  $\mathcal{L}_2$  e  $\mathcal{L}_3$  são as seguintes:

$$\begin{array}{l} S_2 \longrightarrow AD \\ A \longrightarrow aA \mid \lambda \\ D \longrightarrow bDc \mid E \\ E \longrightarrow cE \mid c \end{array}$$

$$\begin{array}{l} S_3 \longrightarrow AF \\ A \longrightarrow aA \mid \lambda \\ F \longrightarrow bFc \mid G \\ G \longrightarrow bG \mid b \end{array}$$

Assim, juntando as três gramáticas resulta na seguinte gramática livre de contexto:

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{array}{l}
 S \longrightarrow S_1 \mid S_2 \mid S_3 \\
 S_1 \longrightarrow BC \\
 B \longrightarrow aBb \mid \lambda \\
 C \longrightarrow cC \mid \lambda \\
 S_2 \longrightarrow AD \\
 A \longrightarrow aA \mid \lambda \\
 D \longrightarrow bDc \mid E \\
 E \longrightarrow cE \mid c \\
 S_3 \longrightarrow AF \\
 F \longrightarrow bFc \mid G \\
 G \longrightarrow bG \mid b
 \end{array}$$

(3i)  $\mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \text{ é ímpar}\}$

**Resposta:**

$$\begin{array}{l}
 S \longrightarrow bS \mid aA \\
 A \longrightarrow aS \mid bA \mid \lambda
 \end{array}$$

(3k)  $\mathcal{L} = \{a^k b^m a^n / 2k = n + m\},$

**Resposta:**

$$\begin{array}{l}
 S \longrightarrow aSaa \mid aAaa \mid aBba \\
 A \longrightarrow aAbb \mid abb \\
 B \longrightarrow aBbb \mid \lambda
 \end{array}$$

(3m)  $\mathcal{L} = \{a^k b^m a^n / m = 2(n + k) + 1\},$

**Resposta:**

$$\begin{array}{l}
 S \longrightarrow AbB \\
 A \longrightarrow aAbb \mid abb \\
 B \longrightarrow bbBa \mid bba
 \end{array}$$

(3o)  $\mathcal{L} = \{a^k b^m a^n b^p / m = k \text{ ou } n = p\}$

**Resposta:**

$$\begin{array}{l}
 S \longrightarrow AB \mid BA \\
 A \longrightarrow aAb \mid ab \\
 B \longrightarrow aB \mid bC \\
 C \longrightarrow bC \mid \lambda
 \end{array}$$

4. Desenhe uma gramática livre de contexto que gere todas as expressões regulares válidas sobre o alfabeto  $\{a, b\}$ . Por exemplo deve permitir gerar as cadeias  $(a + aa^*b)^* + (\lambda + aa)(bb)^*b$  e  $(ab + ba)^*(aa + bb)^*$ .

## A.5. Soluções de Exercícios do Capítulo 5

---

**Resposta:** A gramática deve gerar todas as expressões regulares sobre o alfabeto  $\{a, b\}$ . Assim, por exemplo, deve gerar as expressões regulares  $a$ ,  $\emptyset$ ,  $a+b$ ,  $(a+b)^*b$  e  $(aab+b)^*(a+\lambda)$ . Portanto o conjunto de símbolos terminais da gramática deve conter todos os possíveis símbolos usados numa expressão regular sobre este alfabeto. Logo, o conjunto de símbolos terminais deveria ser:  $\{a, b, \lambda, \emptyset, (,), +, \cdot, ^*\}$ , porém para não confundir a cadeia vazia  $\lambda$  numa gramática livre do contexto com o símbolo terminal  $\lambda$ , usaremos o símbolo  $\underline{\lambda}$  para representar este último. Por uma questão de simplicidade a gramática se baseará na definição original de expressões regulares (definição 3.1.1) com uma única abreviação: a supressão do símbolo de concatenação “.”.

$$S \longrightarrow a \mid b \mid \underline{\lambda} \mid \emptyset \mid (S + S) \mid (S)^* \mid SS$$

Observe que esta gramática permitiria expressões regulares do tipo  $(a\underline{\lambda}\emptyset + bb(\emptyset)^*)^*$ . Mas observe que isto também é permitido pela definição 3.1.1. Caso nos quisermos coibir expressões regulares que possuam uma sub-expressão regular da forma  $r_1r_2$  com  $r_1$  sendo  $\lambda$  ou  $\emptyset$  e  $r_2$  não, ou vice-versa, teríamos que modificar a gramática livre do contexto para:

$$\begin{array}{l} S \longrightarrow \underline{\lambda} \mid \emptyset \mid S_1 \\ S_1 \longrightarrow a \mid b \mid (S + S) \mid (S)^* \mid S_1S_1 \end{array}$$

Observe que isto não diminui o poder de expressão das expressões regulares, pois  $L(r\lambda) = L(r)$  e  $L(r\emptyset) = L(\emptyset)$ .

5. Achar uma gramática livre do contexto para a linguagem

$$\mathcal{L} = \{a^n w w^R b^n / w \in \{a, b\}^*, n \geq 1\}$$

**Resposta:**

$$\begin{array}{l} S \longrightarrow aSb \mid aAb \\ A \longrightarrow aAa \mid bAb \mid \lambda \end{array}$$

6. Defina uma gramática livre do contexto para a linguagem de todas cadeias no alfabeto  $\{a, b\}$  que não são palíndromos. Isto é, para a linguagem

$$\mathcal{L} = \{w \in \{a, b\}^* / w \neq w^R\}$$

**Resposta:**

$$\begin{array}{l} S \longrightarrow aSa \mid bSb \mid A \\ A \longrightarrow aBb \mid bBa \\ B \longrightarrow aB \mid bB \mid \lambda \end{array}$$

7. Seja  $\mathcal{L} = \{a^n b^n / n \geq 0\}$

(7a) Mostre que  $\mathcal{L}^2$  é livre do contexto.

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** Antes devemos analisar quem é  $\mathcal{L}^2$ .  $\mathcal{L}^2$  são todas as cadeias  $w$  tal que  $w = uv$  e  $u \in \mathcal{L}$  e  $v \in \mathcal{L}$ , mas  $u$  e  $v$  não precisam ser as mesmas cadeias, por tanto  $u = a^n b^n$  para algum  $n \geq 0$  e  $v = a^m b^m$  para algum  $m \geq 0$ . Assim,

$$\mathcal{L}^2 = \{a^n b^n a^m b^m / n \geq 0 \text{ e } m \geq 0\}$$

Uma gramática livre do contexto para esta linguagem é a seguinte:

$$\begin{array}{l} S \longrightarrow AA \\ A \longrightarrow aAb \mid \lambda \end{array}$$

8. Mostre uma árvore de derivação para a cadeia  $aabb$ , com a gramática

$$\begin{array}{l} S \longrightarrow AB \mid \lambda, \\ A \longrightarrow aB, \\ B \longrightarrow Sb. \end{array}$$

Dê uma descrição da linguagem gerada por essa gramática.

**Resposta:** Uma árvore de derivação para a cadeia  $aabb$  é ilustrada na figura A.40. A linguagem gerada por esta gramática livre do contexto consiste de todas as cadeias da forma  $a^n b^{2n}$  com  $n \geq 0$ .

11. Mostre que a gramática seguinte é ambígua

$$\begin{array}{l} S \longrightarrow AB \mid aaB, \\ A \longrightarrow a \mid Aa, \\ B \longrightarrow b. \end{array}$$

**Resposta:** Uma gramática é ambígua se existem duas derivações mais à esquerda para uma mesma cadeia. Neste caso a cadeia  $aab$  a qual tem as seguintes derivações mais à esquerda:

- 1)  $S \Rightarrow AB \Rightarrow AaB \Rightarrow aaB \Rightarrow aab$
- 2)  $S \Rightarrow aaB \Rightarrow aab$

12. Construa uma gramática não ambígua, equivalente à gramática do exercício anterior.

**Resposta:** A Ambigüidade na gramática anterior só ocorre por causa da produção  $S \longrightarrow aaB$ , que só é usada para gerar a cadeia  $aab$ . Logo retirando esta produção da gramática, como abaixo, obtemos uma gramática equivalente à anterior mas não ambígua.

$$\begin{array}{l} S \longrightarrow AB \\ A \longrightarrow a \mid Aa, \\ B \longrightarrow b. \end{array}$$

14. Mostre que toda S-gramática é não-ambígua.

## A.5. Soluções de Exercícios do Capítulo 5

---

**Resposta:** Seja  $G = \langle V, T, S, P \rangle$  uma S-gramática. Lembre que em S-gramáticas todas as produções têm a forma  $A \rightarrow ax$  com  $a \in T$  e  $x \in V^*$ . Se  $A \rightarrow ax$  e  $A \rightarrow ay$  são produções em  $P$ , então necessariamente  $x = y$ .

Suponha que ela é ambígua. Então existe uma cadeia, digamos  $a_1 \dots a_n \in T^*$  com pelo menos duas derivações mais à esquerda. Observe que neste caso a variável mais à esquerda é trocada, e portanto a cada etapa um único símbolo é gerado. Logo, duas derivações mais à esquerda realizam as seguintes etapas:

$$S \Rightarrow a_1x_1 \Rightarrow a_1a_2x_2 \Rightarrow \dots \Rightarrow a_1 \dots a_{n-1}x_n \Rightarrow a_1 \dots a_n \quad (\text{A.1})$$

e

$$S \Rightarrow a_1y_1 \Rightarrow a_1a_2y_2 \Rightarrow \dots \Rightarrow a_1 \dots a_{n-1}y_n \Rightarrow a_1 \dots a_n \quad (\text{A.2})$$

com  $x_i, y_i \in V^+$ .

Mostraremos por indução que cada forma sentencial  $a_1 \dots a_k x_k = a_1 \dots a_k y_k$ .

**Base da Indução:**  $k = 1$ . Se  $S \Rightarrow a_1x_1$  e  $S \Rightarrow a_1y_1$  então  $S \rightarrow a_1x_1$  e  $S \rightarrow a_1y_1$  estão ambas em  $P$ . Mas, pela S-condição  $x_1 = y_1$ .

**Hipóteses Indutiva:** Suponha que para algum  $k < n$ ,  $x_k = y_k$ .

**Passo Indutivo:** Seja  $A$  a variável mais à esquerda de  $x_k$  (e pela hipótese indutiva também de  $y_k$ ). Como  $a_1 \dots a_k x_k \Rightarrow a_1 \dots a_k a_{k+1} x_{k+1}$  e  $a_1 \dots a_k y_k \Rightarrow a_1 \dots a_k a_{k+1} y_{k+1}$ , então  $A \rightarrow a_{k+1} x_{k+1}$  e  $A \rightarrow a_{k+1} y_{k+1}$ . Logo pela S-condição,  $x_{k+1} = y_{k+1}$

Assim, para qualquer  $k < n$  temos que  $x_k = y_k$  e por conseguinte a derivação (A.1) é igual à derivação (A.2).

16. Use o teorema 5.5.3, para remover produções esquerda-recursivas nas gramáticas

$$(16a) \quad S \rightarrow abS \mid SS \mid \lambda$$

**Resposta:** A única produção esquerda-recursiva é  $S \rightarrow SS$ , portanto, pelo teorema 5.5.3, na gramática acima eliminamos essa produção e em seu lugar colocamos as produções  $S \rightarrow abSZ$ ,  $S \rightarrow Z$ ,  $Z \rightarrow S$  e  $Z \rightarrow SZ$ , ou seja obtemos a seguinte gramática:

$$\begin{aligned} S &\rightarrow abS \mid abSZ \mid \lambda \mid Z \\ Z &\rightarrow S \mid SZ \end{aligned}$$

$$(16c) \quad S \rightarrow aZ \mid SbS, \\ Z \rightarrow aZb \mid \lambda.$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** Só temos uma produção esquerda-recursiva. Realizando as substituições apontadas pelo teorema 5.5.3 à gramática, obtemos a seguinte gramática livre do contexto:

$$\begin{aligned} S &\longrightarrow aZ \mid aZY \\ Z &\longrightarrow aZb \mid \lambda \\ Y &\longrightarrow Sb \mid SbY \end{aligned}$$

19. Elimine as  $\lambda$ -produções de

$$\begin{aligned} S &\longrightarrow AaB \mid aCaB, \\ A &\longrightarrow aBa \mid \lambda \mid B, \\ B &\longrightarrow bbA \mid \lambda, \\ C &\longrightarrow bbbC \mid AB. \end{aligned}$$

**Resposta:** Claramente  $A$  e  $B$  são anuláveis. Como há uma produção  $C \longrightarrow AB$ , então  $C$  também é anulável. Logo o conjunto  $V_N$  de todas as variáveis anuláveis na gramática é  $\{A, B, C\}$ . Agora colocamos todas produções que não são  $\lambda$ -produções e adicionamos aquelas substituindo em todas as combinações possíveis as variáveis  $A$  e  $B$  por  $\lambda$ , isto resulta na seguinte gramática:

$$\begin{aligned} S &\longrightarrow AaB \mid aCaB \mid Aa \mid aB \mid a \mid aCa \mid aaB \mid aa, \\ A &\longrightarrow aBa \mid B \mid aa, \\ B &\longrightarrow bbA \mid bb, \\ C &\longrightarrow bbbC \mid bbb \mid AB \mid A \mid B. \end{aligned}$$

21. Elimine produções unitárias,  $\lambda$ -produções, produções inúteis e produções esquerda-recursivas nas seguintes gramáticas livres do contexto:

$$\begin{aligned} (21d) \quad S &\longrightarrow AbAaBa \mid ABSa \mid aAAbC, \\ A &\longrightarrow aAa \mid \lambda, \\ B &\longrightarrow bbB \mid BCa \mid aD \mid \lambda, \\ C &\longrightarrow cAE \mid aCCa, \\ D &\longrightarrow E \mid F, \\ E &\longrightarrow aCb, \\ F &\longrightarrow FaF \mid FSaA, \\ G &\longrightarrow SaH, \\ H &\longrightarrow aA \mid \lambda. \end{aligned}$$

**Resposta:** Primeiro eliminaremos produções inúteis usando o algoritmo na prova do teorema 5.5.7. Este é constituído de duas partes. Da primeira parte obtemos as seguintes produções:

$$\begin{aligned} S &\longrightarrow AbAaBa \mid ABSa, \\ A &\longrightarrow aAa \mid \lambda, \\ B &\longrightarrow bbB \mid \lambda, \end{aligned}$$

## A.5. Soluções de Exercícios do Capítulo 5

---

$$G \longrightarrow SaH,$$

$$H \longrightarrow aA \mid \lambda.$$

Agora aplicando a segunda parte obtemos:

$$S \longrightarrow AbAaBa \mid ABSa,$$

$$A \longrightarrow aAa \mid \lambda,$$

$$B \longrightarrow bbB \mid \lambda,$$

Agora eliminaremos  $\lambda$ -produções e aplicamos o algoritmo do teorema 5.5.10, resultando na gramática:

$$S \longrightarrow AbAaBa \mid ABSa \mid bAaBa \mid AbaBa \mid AbAaa \mid baBa \mid bAaa \mid Abaa \mid baa \mid ASa \mid BSa \mid Sa,$$

$$A \longrightarrow aAa \mid aa,$$

$$B \longrightarrow bbB \mid bb,$$

Como não há produções unitárias passamos a remover produções esquerda recursivas usando o algoritmo do teorema 5.5.3. Note que só existe uma produção esquerda recursiva:  $S \longrightarrow Sa$ . A remoção resulta na seguinte gramática:

$$S \longrightarrow AbAaBa \mid ABSa \mid bAaBa \mid AbaBa \mid AbAaa \mid baBa \mid bAaa \mid Abaa \mid baa \mid ASa \mid BSa \mid AbAaBaZ \mid ABSaZ \mid bAaBaZ \mid AbaBaZ \mid AbAaaZ \mid baBaZ \mid bAaaZ \mid AbaaZ \mid baaZ \mid ASaZ \mid BSaZ,$$

$$Z \longrightarrow a \mid aZ,$$

$$A \longrightarrow aAa \mid aa,$$

$$B \longrightarrow bbB \mid bb,$$

23. Converter as seguintes gramáticas à forma normal de Chomsky

$$(23a) S \longrightarrow aSb \mid ab$$

**Resposta:** Para transformar esta gramática livre do contexto numa gramática também livre do contexto equivalente na forma normal de Chomsky, a demonstração do teorema 5.6.2, sugere que primeiro se devem eliminar as produções unitárias e  $\lambda$ -produções, que neste caso não há, para depois seguir as seguintes duas etapas.

**Etapa 1:** Devemos substituir cada símbolo terminal, por exemplo  $a$ , nas produções por novas variáveis,  $B_a$ , e depois introduzir as produções  $B_a \longrightarrow a$ . Assim, aplicando esta etapa, a gramática acima se transforma na gramática:

$$\begin{aligned} S &\longrightarrow B_aSB_b \mid B_aB_b \\ B_a &\longrightarrow a \\ B_b &\longrightarrow b \end{aligned}$$

**Etapa 2:** Devemos substituir as produções que tenham mais de duas variáveis (suponha  $A \longrightarrow X_1 \dots X_n$ ) por duas produções, uma com duas variáveis ( $A \longrightarrow Y_1X_n$ ) e a outra com  $n - 1$  variáveis ( $Y \longrightarrow X_1 \dots X_{n-1}$ ), onde a variável  $Y$  é nova. Repetimos este processo até só ficarem produções com duas variáveis. Como neste caso temos uma única produção com mais de duas variáveis à direita ( $S \longrightarrow B_aSB_b$ ), esta etapa só tem um passo. Assim, a gramática resultante dessa etapa é:

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
 S &\longrightarrow YB_b \mid B_aB_b \\
 Y &\longrightarrow B_aS \\
 B_a &\longrightarrow a \\
 B_b &\longrightarrow b
 \end{aligned}$$

a qual está na forma normal de Chomsky.

$$\begin{aligned}
 (23c) \quad S &\longrightarrow abAB, \\
 A &\longrightarrow bAB \mid \lambda, \\
 B &\longrightarrow BAa \mid A \mid \lambda.
 \end{aligned}$$

**Resposta:** Novamente seguiremos o algoritmo da demonstração do teorema 5.6.2 para transformar a gramática acima numa equivalente na forma normal de Chomsky. Neste caso primeiro temos que eliminar produções unitárias e  $\lambda$ -produções. Eliminando  $\lambda$ -produções obtemos a seguinte gramática:

$$\begin{aligned}
 S &\longrightarrow abAB \mid abB \mid abA \mid ab \\
 A &\longrightarrow bAB \mid bB \mid b \\
 B &\longrightarrow BAa \mid A \mid Aa \mid Ba \mid a
 \end{aligned}$$

Eliminando agora produções unitárias obtemos a seguinte gramática:

$$\begin{aligned}
 S &\longrightarrow abAB \mid abB \mid abA \mid ab \\
 A &\longrightarrow bAB \mid bB \mid b \\
 B &\longrightarrow BAa \mid bAB \mid bB \mid b \mid Aa \mid Ba \mid a
 \end{aligned}$$

Seguindo a etapa 1, obtemos a seguinte gramática:

$$\begin{aligned}
 S &\longrightarrow B_aB_bAB \mid B_aB_bB \mid B_aB_bA \mid B_aB_b \\
 A &\longrightarrow B_bAB \mid B_bB \mid b \\
 B &\longrightarrow BAB_a \mid B_bAB \mid B_bB \mid b \mid AB_a \mid BB_a \mid a \\
 B_a &\longrightarrow a \\
 B_b &\longrightarrow b
 \end{aligned}$$

Seguindo a etapa 2 obtemos a seguinte gramática, faremos isto introduzindo a menor quantidade de variáveis possíveis, ou seja usaremos o bom senso em vez do “algoritmo” da demonstração do teorema 5.6.2:

$$\begin{aligned}
 S &\longrightarrow Y_1Y_2 \mid Y_1B \mid Y_1A \mid B_aB_b \\
 Y_1 &\longrightarrow B_aB_b \\
 Y_2 &\longrightarrow AB \\
 A &\longrightarrow B_bY_2 \mid B_bB \mid b \\
 B &\longrightarrow Y_3B_a \mid B_bY_2 \mid B_bB \mid b \mid AB_a \mid BB_a \mid a \\
 Y_3 &\longrightarrow BA \\
 B_a &\longrightarrow a \\
 B_b &\longrightarrow b
 \end{aligned}$$

25. Converter as seguintes gramáticas na forma normal de Greibach.

$$(25a) \quad S \longrightarrow aaS \mid SS \mid aa.$$

## A.5. Soluções de Exercícios do Capítulo 5

---

**Resposta:** A demonstração do teorema 5.6.6 define 4 etapas para transformar uma gramática livre do contexto numa equivalente na forma normal de Greibach. Seguiremos cada uma dessas etapas.

**Etapa 1:** Transformaremos primeiro à forma normal de Chomsky. Na primeira etapa obtemos a seguinte gramática:

$$\begin{aligned} S &\longrightarrow B_a B_a S \mid SS \mid B_a B_a \\ B_a &\longrightarrow a \end{aligned}$$

Na segunda etapa obtemos a forma normal de Chomsky:

$$\begin{aligned} S &\longrightarrow YS \mid SS \mid B_a B_a \\ Y &\longrightarrow B_a B_a \\ B_a &\longrightarrow a \end{aligned}$$

**Etapa 2:** Rotulando as variáveis obtemos a seguinte gramática:

$$\begin{aligned} A_1 &\longrightarrow A_2 A_1 \mid A_1 A_1 \mid A_3 A_3 \\ A_2 &\longrightarrow A_3 A_3 \\ A_3 &\longrightarrow a \end{aligned}$$

**Etapa 3:** Usamos os teoremas 5.5.1 e 5.5.3 para deixar todas as produções na forma prevista nesta etapa. Como resultado obtemos a seguinte gramática:

$$\begin{aligned} A_1 &\longrightarrow A_2 A_1 \mid A_2 A_1 Z_1 \mid A_3 A_3 \mid A_3 A_3 Z_1 \\ Z_1 &\longrightarrow A_1 A_1 \mid A_1 A_1 Z_1 \\ A_2 &\longrightarrow A_3 A_3 \\ A_3 &\longrightarrow a \end{aligned}$$

**Etapa 4:** Usamos o teorema 5.5.1 para transformar a gramática resultante da etapa anterior numa forma normal de Greibach. Faremos isto passo a passo. Primeiro substituiremos a ocorrência mais à esquerda de  $A_3$  no lado direito da produção  $A_2 \longrightarrow A_3 A_3$  (Observe que a única produção tendo  $A_3$  no lado esquerdo satisfaz as condições de Greibach), isto resulta na seguinte gramática:

$$\begin{aligned} A_1 &\longrightarrow A_2 A_1 \mid A_2 A_1 Z_1 \mid A_3 A_3 \mid A_3 A_3 Z_1 \\ Z_1 &\longrightarrow A_1 A_1 \mid A_1 A_1 Z_1 \\ A_2 &\longrightarrow a A_3 \\ A_3 &\longrightarrow a \end{aligned}$$

Agora todas as produções com  $A_2$  ou  $A_3$  no lado esquerdo estão na forma normal de Greibach. Agora substituiremos a variável mais à esquerda de cada produção tendo  $A_1$  no lado esquerdo (observe que por se encontrar nesta etapa, essa variável só pode ser  $A_2$  ou  $A_3$ ). Como resultado obtemos a seguinte gramática:

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
 A_1 &\longrightarrow aA_3A_1 \mid aA_3A_1Z_1 \mid aA_3 \mid aA_3Z_1 \\
 Z_1 &\longrightarrow A_1A_1 \mid A_1A_1Z_1 \\
 A_2 &\longrightarrow aA_3 \\
 A_3 &\longrightarrow a
 \end{aligned}$$

Finalmente fazemos o mesmo com as produções cujo lado esquerdo seja  $Z_1$ . Observe que neste caso, a variável mais à esquerda do lado direito da produção sempre será da forma  $A_i$ , como todos os  $A_i$ 's estão na forma normal de Greibach esta substituição acarretará em que as produções com  $Z_1$  no lado esquerdo também estejam na forma normal de Greibach. Isto resulta na seguinte gramática:

$$\begin{aligned}
 A_1 &\longrightarrow aA_3A_1 \mid aA_3A_1Z_1 \mid aA_3 \mid aA_3Z_1 \\
 Z_1 &\longrightarrow aA_3A_1A_1 \mid aA_3A_1Z_1A_1 \mid aA_3A_1 \mid aA_3Z_1A_1 \\
 &\quad aA_3A_1A_1Z_1 \mid aA_3A_1Z_1A_1Z_1 \mid aA_3A_1Z_1 \mid aA_3Z_1A_1Z_1 \\
 A_2 &\longrightarrow aA_3 \\
 A_3 &\longrightarrow a
 \end{aligned}$$

$$\begin{aligned}
 (25c) \quad S &\longrightarrow aA \mid Bbb, \\
 A &\longrightarrow aBa \mid aa, \\
 B &\longrightarrow Abb \mid Cab, \\
 C &\longrightarrow Aba \mid Bab \mid aba
 \end{aligned}$$

**Resposta:** A demonstração do teorema 5.6.6 define 4 etapas para transformar uma gramática livre do contexto numa equivalente na forma normal de Greibach. Seguiremos cada uma dessas etapas.

**Etapa 1:** Transformaremos primeiro à forma normal de Chomsky. Na primeira etapa obtemos a seguinte gramática:

Forma normal Pre-Chomsky: Por simplicidade usaremos nomes de variáveis  $X$  e  $Y$  em vez de  $B_a$  e  $B_b$  respectivamente.

$$\begin{aligned}
 S &\longrightarrow XA \mid BYY, \\
 A &\longrightarrow XBX \mid XX, \\
 B &\longrightarrow AYY \mid CXY, \\
 C &\longrightarrow AYX \mid BXY \mid XYX \\
 X &\longrightarrow a, \\
 Y &\longrightarrow b.
 \end{aligned}$$

Forma normal de Chomsky:

$$\begin{aligned}
 S &\longrightarrow XA \mid BD, \\
 A &\longrightarrow XE \mid XX, \\
 B &\longrightarrow AD \mid CF, \\
 C &\longrightarrow AG \mid BF \mid FX \\
 D &\longrightarrow YY, \\
 E &\longrightarrow BX, \\
 F &\longrightarrow XY,
 \end{aligned}$$

## A.5. Soluções de Exercícios do Capítulo 5

---

$$G \longrightarrow YX,$$

$$X \longrightarrow a,$$

$$Y \longrightarrow b.$$

**Etapa 2:** Antes de rotular as variáveis, para diminuir passos nas etapas posteriores, é aconselhável analisar qual a melhor rotulação. Neste caso, é bom olhar o grafo de dependência entre as variáveis (considerando só as do lado esquerdo de uma produção e a mais à esquerda do lado direito da mesma produção). A figura A.41 mostra tal grafo de dependência. A ordem entre  $B$  e  $C$  não é fundamental, pois eles estão entrelaçados. Porem como  $B$  tem menor produções que  $C$  consideraremos  $B$  precedendo  $C$ . Assim, a ordem considerada é:  $S, E, B, C, A, D, F, G, X, Y$  e rotulando essas variáveis nessa ordem obtemos a seguinte gramática:

$$\begin{aligned} V_1 &\longrightarrow V_9V_5 \mid V_3V_6, \\ V_5 &\longrightarrow V_9V_2 \mid V_9V_9, \\ V_3 &\longrightarrow V_5V_6 \mid V_4V_7, \\ V_4 &\longrightarrow V_5V_8 \mid V_3V_7 \mid V_7V_9 \\ V_6 &\longrightarrow V_{10}V_{10}, \\ V_2 &\longrightarrow V_3V_9, \\ V_7 &\longrightarrow V_9V_{10}, \\ V_8 &\longrightarrow V_{10}V_9, \\ V_9 &\longrightarrow a, \\ V_{10} &\longrightarrow b. \end{aligned}$$

**Etapa 3:** Usamos os teoremas 5.5.1 e 5.5.3 para deixar todas as produções na forma prevista nesta etapa. Primeiro observe que a única produção que não satisfaz as condições desta etapa é  $V_4 \longrightarrow V_3V_7$ . Assim, aplicando o teorema 5.5.1 obtemos a seguinte gramática:

$$\begin{aligned} V_1 &\longrightarrow V_9V_5 \mid V_3V_6, \\ V_5 &\longrightarrow V_9V_2 \mid V_9V_9, \\ V_3 &\longrightarrow V_5V_6 \mid V_4V_7, \\ V_4 &\longrightarrow V_5V_8 \mid V_5V_6V_7 \mid V_4V_7V_7 \mid V_7V_9 \\ V_6 &\longrightarrow V_{10}V_{10}, \\ V_2 &\longrightarrow V_3V_9, \\ V_7 &\longrightarrow V_9V_{10}, \\ V_8 &\longrightarrow V_{10}V_9, \\ V_9 &\longrightarrow a, \\ V_{10} &\longrightarrow b. \end{aligned}$$

Agora temos uma produção esquerda recursiva ( $V_4 \longrightarrow V_4V_7V_7$ ). Eliminamos esta usando o teorema 5.5.3. Isto resulta na seguinte gramática:

$$\begin{aligned} V_1 &\longrightarrow V_9V_5 \mid V_3V_6, \\ V_5 &\longrightarrow V_9V_2 \mid V_9V_9, \\ V_3 &\longrightarrow V_5V_6 \mid V_4V_7, \end{aligned}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

$$\begin{aligned}
 V_4 &\longrightarrow V_5V_8 \mid V_5V_6V_7 \mid V_7V_9 \mid V_5V_8Z_1 \mid V_5V_6V_7Z_1 \mid V_7V_9Z_1 \\
 Z_1 &\longrightarrow V_7V_7 \mid V_7V_7Z_1 \\
 V_6 &\longrightarrow V_{10}V_{10}, \\
 V_2 &\longrightarrow V_3V_9, \\
 V_7 &\longrightarrow V_9V_{10}, \\
 V_8 &\longrightarrow V_{10}V_9, \\
 V_9 &\longrightarrow a, \\
 V_{10} &\longrightarrow b.
 \end{aligned}$$

**Etapa 4:** Usamos o teorema 5.5.1 para transformar a gramática resultante da etapa anterior numa na forma normal de Greibach. Observe que  $V_9$  e  $V_{10}$  estão na forma normal de Greibach e que as produções de  $V_5$  a  $V_8$  começam com uma delas. Por tanto faremos todas estas substituições de uma única vez:

$$\begin{aligned}
 V_1 &\longrightarrow V_9V_5 \mid V_3V_6, \\
 V_5 &\longrightarrow aV_2 \mid aV_9, \\
 V_3 &\longrightarrow V_5V_6 \mid V_4V_7, \\
 V_4 &\longrightarrow V_5V_8 \mid V_5V_6V_7 \mid V_7V_9 \mid V_5V_8Z_1 \mid V_5V_6V_7Z_1 \mid V_7V_9Z_1 \\
 Z_1 &\longrightarrow V_7V_7 \mid V_7V_7Z_1 \\
 V_6 &\longrightarrow bV_{10}, \\
 V_2 &\longrightarrow V_3V_9, \\
 V_7 &\longrightarrow aV_{10}, \\
 V_8 &\longrightarrow bV_9, \\
 V_9 &\longrightarrow a, \\
 V_{10} &\longrightarrow b.
 \end{aligned}$$

Agora substituiremos todas as variáveis mais à esquerda das produções de:

$$\begin{aligned}
 V_4) \quad V_4 &\longrightarrow aV_2V_8 \mid aV_9V_8 \mid aV_2V_6V_7 \mid aV_9V_6V_7 \mid aV_{10}V_9 \mid aV_2V_8Z_1 \mid aV_9V_8Z_1 \mid \\
 &\quad aV_2V_6V_7Z_1 \mid aV_9V_6V_7Z_1 \mid aV_{10}V_9Z_1 \\
 V_3) \quad V_3 &\longrightarrow aV_2V_6 \mid aV_9V_6 \mid aV_2V_8V_7 \mid aV_9V_8V_7 \mid aV_2V_6V_7V_7 \mid aV_9V_6V_7V_7 \mid \\
 &\quad aV_{10}V_9V_7 \mid aV_2V_8Z_1V_7 \mid aV_9V_8Z_1V_7 \mid aV_2V_6V_7Z_1V_7 \mid \\
 &\quad aV_9V_6V_7Z_1V_7 \mid aV_{10}V_9Z_1V_7, \\
 V_2) \quad V_2 &\longrightarrow aV_2V_6V_9 \mid aV_9V_6V_9 \mid aV_2V_8V_7V_9 \mid aV_9V_8V_7V_9 \mid aV_2V_6V_7V_7V_9 \mid \\
 &\quad aV_9V_6V_7V_7V_9 \mid aV_{10}V_9V_7V_9 \mid aV_2V_8Z_1V_7V_9 \mid aV_9V_8Z_1V_7V_9 \mid \\
 &\quad aV_2V_6V_7Z_1V_7V_9 \mid aV_9V_6V_7Z_1V_7V_9 \mid aV_{10}V_9Z_1V_7V_9, \\
 V_1) \quad V_1 &\longrightarrow aV_5 \mid aV_2V_6V_6 \mid aV_9V_6V_6 \mid aV_2V_8V_7V_6 \mid aV_9V_8V_7V_6 \mid aV_2V_6V_7V_7V_6 \mid \\
 &\quad aV_9V_6V_7V_7V_6 \mid aV_{10}V_9V_7V_6 \mid aV_2V_8Z_1V_7V_6 \mid aV_9V_8Z_1V_7V_6 \mid \\
 &\quad aV_2V_6V_7Z_1V_7V_6 \mid aV_9V_6V_7Z_1V_7V_6 \mid aV_{10}V_9Z_1V_7V_6,
 \end{aligned}$$

Finalmente de  $Z_1$ :

$$Z_1 \longrightarrow aV_{10}V_7 \mid aV_{10}V_7Z_1.$$

Juntando todo obtemos a seguinte gramática:

## A.5. Soluções de Exercícios do Capítulo 5

---

$$\begin{aligned}
 V_1 \longrightarrow & \quad aV_5 \mid aV_2V_6V_6 \mid aV_9V_6V_6 \mid aV_2V_8V_7V_6 \mid aV_9V_8V_7V_6 \mid aV_2V_6V_7V_7V_6 \mid \\
 & \quad aV_9V_6V_7V_7V_6 \mid aV_{10}V_9V_7V_6 \mid aV_2V_8Z_1V_7V_6 \mid aV_9V_8Z_1V_7V_6 \mid \\
 & \quad aV_2V_6V_7Z_1V_7V_6 \mid aV_9V_6V_7Z_1V_7V_6 \mid aV_{10}V_9Z_1V_7V_6, \\
 V_3 \longrightarrow & \quad aV_2V_6 \mid aV_9V_6 \mid aV_2V_8V_7 \mid aV_9V_8V_7 \mid aV_2V_6V_7V_7 \mid aV_9V_6V_7V_7 \mid \\
 & \quad aV_{10}V_9V_7 \mid aV_2V_8Z_1V_7 \mid aV_9V_8Z_1V_7 \mid aV_2V_6V_7Z_1V_7 \mid \\
 & \quad aV_9V_6V_7Z_1V_7 \mid aV_{10}V_9Z_1V_7, \\
 V_4 \longrightarrow & \quad aV_2V_8 \mid aV_9V_8 \mid aV_2V_6V_7 \mid aV_9V_6V_7 \mid aV_{10}V_9 \mid aV_2V_8Z_1 \mid aV_9V_8Z_1 \mid \\
 & \quad aV_2V_6V_7Z_1 \mid aV_9V_6V_7Z_1 \mid aV_{10}V_9Z_1 \\
 Z_1 \longrightarrow & \quad aV_{10}V_7 \mid aV_{10}V_7Z_1 \\
 V_6 \longrightarrow & \quad bV_{10}, \\
 V_2 \longrightarrow & \quad aV_2V_6V_9 \mid aV_9V_6V_9 \mid aV_2V_8V_7V_9 \mid aV_9V_8V_7V_9 \mid aV_2V_6V_7V_7V_9 \mid \\
 & \quad aV_9V_6V_7V_7V_9 \mid aV_{10}V_9V_7V_9 \mid aV_2V_8Z_1V_7V_9 \mid aV_9V_8Z_1V_7V_9 \mid \\
 & \quad aV_2V_6V_7Z_1V_7V_9 \mid aV_9V_6V_7Z_1V_7V_9 \mid aV_{10}V_9Z_1V_7V_9, \\
 V_7 \longrightarrow & \quad aV_{10}, \\
 V_8 \longrightarrow & \quad bV_9, \\
 V_9 \longrightarrow & \quad a, \\
 V_{10} \longrightarrow & \quad b.
 \end{aligned}$$

## A.6 Soluções de Exercícios do Capítulo 6

3. Construa APN's que reconheçam as seguintes linguagens, sobre  $\Sigma = \{a, b\}$  (considere  $k, m, n, p \geq 1$ )

$$(3b) \mathcal{L} = \{a^m b^n / n \leq m \leq 3n\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$ $(q_0, AAz)$ $(q_0, AAAz)$		
$(q_0, A)$	$(q_0, AA)$ $(q_0, AAA)$ $(q_0, AAAA)$	$(q_1, \lambda)$	
$(q_1, A)$			$(q_1, \lambda)$
$(q_1, z)$			$(q_2, \lambda)$

onde o estado final é  $q_2$ .

$$(3e) \mathcal{L} = \{a^m b^n / n = m + 1 \text{ ou } m = n + 1\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_1, z)$ $(q_1, AAz)$		
$(q_1, z)$	$(q_1, Az)$		$(q_3, \lambda)$
$(q_1, A)$	$(q_1, AA)$	$(q_2, \lambda)$	
$(q_2, A)$		$(q_2, \lambda)$	
$(q_2, z)$			$(q_3, \lambda)$

onde o estado final é  $q_3$ .

$$(3g) \mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) = \mathcal{N}_b(w) + 1\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_2, z)$	$(q_1, Bz)$	
$(q_1, B)$	$(q_2, B)$	$(q_1, BB)$	
$(q_2, A)$	$(q_2, AA)$	$(q_2, \lambda)$	
$(q_2, B)$	$(q_2, \lambda)$	$(q_2, BB)$	
$(q_2, z)$	$(q_2, Az)$	$(q_2, Bz)$	$(q_3, \lambda)$

onde o estado final é  $q_3$ .

$$(3i) \mathcal{L} = \{w \in \Sigma^* / 2\mathcal{N}_a(w) \leq \mathcal{N}_b(w) \leq 3\mathcal{N}_a(w)\}$$

## A.6. Soluções de Exercícios do Capítulo 6

---

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, AAz) (q_0, AAAz)$	$(q_0, Bz)$	$(q_3, \lambda)$
$(q_0, B)$	$(q_1, \lambda)$	$(q_0, BB)$	
$(q_0, A)$	$(q_0, AAA) (q_0, AAAA)$	$(q_0, \lambda)$	
$(q_1, B)$			$(q_2, \lambda) (q_0, \lambda)$
$(q_1, A)$			$(q_0, AA) (q_0, AAA)$
$(q_2, B)$			$(q_0, \lambda)$
$(q_2, A)$			$(q_0, AA)$

onde o estado final é  $q_3$ .

$$(3k) \mathcal{L} = \{a^m b^n / n = \lfloor \frac{m}{2} \rfloor\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_1, z)$		
$(q_0, A)$	$(q_1, A)$	$(q_2, \lambda)$	
$(q_1, z)$	$(q_0, Az)$		$(q_3, \lambda)$
$(q_1, A)$	$(q_0, AA)$	$(q_2, \lambda)$	
$(q_2, A)$		$(q_2, \lambda)$	
$(q_2, z)$			$(q_3, \lambda)$

onde o estado final é  $q_3$ .

$$(3m) \mathcal{L} = \{a^k b^m a^n b^p / k = 2n \text{ ou } m = 2p\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_1, Az) (q_5, z)$		
$(q_0, A)$	$(q_1, AA)$	$(q_2, A)$	
$(q_1, A)$	$(q_0, A)$		
$(q_2, A)$	$(q_3, \lambda)$	$(q_2, A)$	
$(q_3, A)$	$(q_3, \lambda)$		
$(q_3, z)$		$(q_4, z)$	
$(q_4, z)$		$(q_4, z)$	$(q_{10}, \lambda)$
$(q_5, z)$	$(q_5, z)$	$(q_6, Az)$	
$(q_6, A)$		$(q_7, A)$	
$(q_7, A)$	$(q_8, A)$	$(q_6, AA)$	
$(q_8, A)$	$(q_8, A)$	$(q_9, \lambda)$	
$(q_9, A)$		$(q_9, \lambda)$	
$(q_9, z)$			$(q_{10}, \lambda)$

onde o estado final é  $q_{10}$ .

$$(3o) \mathcal{L} = \{a^k b^m a^n b^p / k = m \text{ ou } n \neq p\}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$		
$(q_0, A)$	$(q_0, AA)$	$(q_1, \lambda)$	
$(q_1, A)$	$(q_4, \lambda)$	$(q_1, \lambda)$	
$(q_1, z)$	$(q_2, z)$		
$(q_2, z)$	$(q_2, z)$	$(q_3, z)$	
$(q_3, z)$		$(q_3, z)$	$(q_8, \lambda)$
$(q_4, A)$			$(q_4, \lambda)$
$(q_4, z)$			$(q_5, Az)$
$(q_5, A)$	$(q_5, AA)$	$(q_6, \lambda)$	
$(q_6, A)$		$(q_6, \lambda)$	$(q_8, \lambda)$
$(q_6, z)$		$(q_7, z)$	
$(q_7, z)$		$(q_7, z)$	$(q_8, \lambda)$

onde o estado final é  $q_8$ .

$$(3q) \mathcal{L} = \{a^k b^m a^n b^p \mid k - m = n - p\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$		
$(q_0, A)$	$(q_0, AA)$	$(q_1, \lambda)$	
$(q_1, A)$	$(q_2, \lambda)$	$(q_1, \lambda)$	
$(q_1, z)$	$(q_3, Az)$	$(q_1, Bz)$	
$(q_1, B)$	$(q_5, BB)$	$(q_1, BB)$	
$(q_2, A)$	$(q_2, \lambda)$		
$(q_2, z)$	$(q_3, Az)$		
$(q_3, A)$	$(q_3, AA)$	$(q_4, \lambda)$	
$(q_4, A)$		$(q_4, \lambda)$	
$(q_4, z)$			$(q_7, \lambda)$
$(q_5, B)$	$(q_5, BB)$	$(q_6, \lambda)$	
$(q_6, B)$		$(q_6, \lambda)$	
$(q_6, z)$			$(q_7, \lambda)$

onde o estado final é  $q_8$ . Observe que este APN considera tanto  $k - m$  positivo quanto  $k - m$  negativo.

4. Construa APN's que reconheçam as seguintes linguagens sobre  $\Sigma = \{a, b, c\}$

$$(4a) \mathcal{L} = \{wcw^R \mid w \in \{a, b\}^*\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

## A.6. Soluções de Exercícios do Capítulo 6

---

	$a$	$b$	$c$	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_0, Bz)$	$(q_2, \lambda)$	
$(q_0, A)$	$(q_0, AA)$	$(q_0, BA)$	$(q_1, A)$	
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB)$	$(q_1, B)$	
$(q_1, A)$		$(q_1, \lambda)$		
$(q_1, B)$			$(q_1, \lambda)$	
$(q_1, z)$				$(q_2, \lambda)$

onde o estado final é  $q_2$ .

$$(4c) \mathcal{L} = \{a^m b^{m+n} c^n / n \geq 0, m \geq 0\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$c$	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_2, Bz)$		
$(q_0, A)$	$(q_0, AA)$	$(q_1, \lambda)$		
$(q_1, A)$		$(q_1, \lambda)$		
$(q_1, z)$		$(q_2, Bz)$		$(q_4, \lambda)$
$(q_2, B)$		$(q_2, BB)$	$(q_3, \lambda)$	
$(q_3, B)$			$(q_3, \lambda)$	
$(q_3, z)$				$(q_4, \lambda)$

onde o estado final é  $q_4$ .

$$(4e) \mathcal{L} = \{w \in \Sigma^* / \mathcal{N}_a(w) + \mathcal{N}_b(w) = \mathcal{N}_c(w)\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$c$	$\lambda$
$(q_0, z)$	$(q_0, Az)$	$(q_0, Az)$	$(q_0, Cz)$	$(q_1, \lambda)$
$(q_0, A)$	$(q_0, AA)$	$(q_0, AA)$	$(q_0, \lambda)$	
$(q_0, C)$	$(q_0, \lambda)$	$(q_0, \lambda)$	$(q_0, CC)$	

onde o estado final é  $q_1$ .

$$(4g) \mathcal{L} = \{a^k b^m c^n / m = 2n + k - 1\}$$

**Resposta:** A seguinte tabela descreve o APN que reconhece esta linguagem

	$a$	$b$	$c$	$\lambda$
$(q_0, z)$	$(q_0, Az)$			
$(q_0, A)$	$(q_0, AA)$	$(q_1, \lambda)$	$(q_0, \lambda)$	
$(q_1, A)$		$(q_1, \lambda)$		
$(q_1, z)$		$(q_2, Az)$		
$(q_2, A)$		$(q_2, AA)$	$(q_3, \lambda)$	
$(q_3, A)$				$(q_4, \lambda)$
$(q_3, z)$				$(q_5, \lambda)$
$(q_4, A)$			$(q_3, \lambda)$	

onde o estado final é  $q_5$ .

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

5. Que linguagem é aceita pelo APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_1, A), (q_2, \lambda)\}, \\ \delta(q_1, b, A) &= \{(q_1, B)\}, \\ \delta(q_1, b, B) &= \{(q_1, B)\}, \\ \delta(q_1, a, B) &= \{(q_2, \lambda)\} ?\end{aligned}$$

**Resposta:** Este APN aceita a linguagem

$$L(M) = \{ab^n / n \geq 0\} \cup \{a\}$$

6. Que linguagem aceita o APN anterior se  $F = \{q_0, q_1, q_2\}$ ?

**Resposta:** A linguagem

$$\{ab^n / n \geq 0\} \cup \{ab^n a / n \geq 0\} \cup \{\lambda\}$$

9. Construa um APN que reconheça a linguagem gerada pelas seguintes gramáticas livres de contexto:

$$\begin{aligned}(9a) \quad S &\longrightarrow aABB \mid aAA, \\ A &\longrightarrow aBB \mid a, \\ B &\longrightarrow bBB \mid A.\end{aligned}$$

**Resposta:** Observe que a gramática acima já está na forma normal pseudo-Greibach e portanto podemos aplicar diretamente o teorema 6.3.2, obtendo o seguinte apn:

	$a$	$b$	$\lambda$
$(q_0, z)$			
$(q_1, S)$	$(q_1, ABB) (q_1, AA)$		$(q_1, Sz)$
$(q_1, A)$	$(q_1, BB) (q_1, \lambda)$		
$(q_1, B)$		$(q_1, BB)$	$(q_1, A)$
$(q_1, z)$			$(q_2, \lambda)$

onde  $q_2$  é o estado final;

$$\begin{aligned}(9c) \quad S &\longrightarrow baB \mid aABA \mid ab, \\ A &\longrightarrow aA \mid a, \\ B &\longrightarrow bBb \mid baAbB \mid \lambda\end{aligned}$$

## A.6. Soluções de Exercícios do Capítulo 6

---

**Resposta:** Antes de aplicarmos o algoritmo no teorema 6.3.2, deveremos transformar esta gramática à forma normal pseudo-Greibach. Isto é simples, basta trocar cada símbolo terminal  $a$  e  $b$ , nos lados direitos das produções que não sejam os símbolos mais à esquerda por uma variável  $X$  e  $Y$ , respectivamente. Fazendo isto obtemos a seguinte gramática livre do contexto:

$$S \longrightarrow bXB \mid aABX \mid aY,$$

$$A \longrightarrow aA \mid a,$$

$$B \longrightarrow bBY \mid bXAYB \mid \lambda,$$

$$X \longrightarrow a,$$

$$Y \longrightarrow b.$$

Agora sim aplicamos o algoritmo e obtemos o seguinte APN:

	$a$	$b$	$\lambda$
$(q_0, z)$			
$(q_1, S)$	$(q_1, ABX) (q_1, Y)$	$(q_1, XB)$	$(q_1, Sz)$
$(q_1, A)$	$(q_1, A) (q_1, \lambda)$		
$(q_1, B)$		$(q_1, BY) (q_1, XAYB)$	$(q_1, \lambda)$
$(q_1, X)$	$(q_1, \lambda)$		
$(q_1, Y)$		$(q_1, \lambda)$	
$(q_1, z)$			$(q_2, \lambda)$

onde  $q_2$  é o estado final;

11. Encontre uma gramática livre do contexto que gere a linguagem aceita pelo APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Az)$		
$(q_0, A)$	$(q_1, \lambda)$	$(q_0, AA)$	
$(q_1, A)$			$(q_1, \lambda)$
$(q_1, z)$			$(q_2, \lambda)$

**Resposta:** Primeiro observemos que o APN satisfaz as duas condições iniciais, isto é tem um único estado final no qual só entra se a pilha estiver vazia e todas as transições têm a forma descrita na seção 6.3.2. Aplicando o algoritmo descrito na demonstração do teorema 6.3.6 obtemos a seguinte gramática livre do contexto:

$$\begin{aligned}
 (q_0 z q_0) &\longrightarrow a(q_0 A q_0)(q_0 z q_0) \mid a(q_0 A q_1)(q_1 z q_0) \mid a(q_0 A q_2)(q_2 z q_0) \\
 (q_0 z q_1) &\longrightarrow a(q_0 A q_0)(q_0 z q_1) \mid a(q_0 A q_1)(q_1 z q_1) \mid a(q_0 A q_2)(q_2 z q_1) \\
 (q_0 z q_2) &\longrightarrow a(q_0 A q_0)(q_0 z q_2) \mid a(q_0 A q_1)(q_1 z q_2) \mid a(q_0 A q_2)(q_2 z q_2) \\
 (q_0 A q_0) &\longrightarrow b(q_0 A q_0)(q_0 A q_0) \mid b(q_0 A q_1)(q_1 A q_0) \mid b(q_0 A q_2)(q_2 A q_0) \\
 (q_0 A q_1) &\longrightarrow b(q_0 A q_0)(q_0 A q_1) \mid b(q_0 A q_1)(q_1 A q_1) \mid b(q_0 A q_2)(q_2 A q_1) \\
 (q_0 A q_2) &\longrightarrow b(q_0 A q_0)(q_0 A q_2) \mid b(q_0 A q_1)(q_1 A q_2) \mid b(q_0 A q_2)(q_2 A q_2) \\
 (q_0 A q_1) &\longrightarrow a \\
 (q_1 A q_1) &\longrightarrow a \\
 (q_1 z q_2) &\longrightarrow \lambda
 \end{aligned}$$

a variável de inicio é  $(q_0 z q_2)$ . É claro que a gramática resultante tem várias produções inúteis.

13. Seja o seguinte APN

$$M = \langle \{q_0, q_1, q_2\}, \{a, b\}, \{A, B, z\}, \delta, q_0, z, \{q_2\} \rangle, \text{ onde}$$

	<i>a</i>	<i>b</i>	$\lambda$
$(q_0, z)$	$(q_0, Xz) (q_2, z)$	$(q_0, Yz) (q_2, z)$	
$(q_0, X)$	$(q_0, AX) (q_1, X) (q_2, \lambda)$	$(q_0, BX)$	
$(q_0, A)$	$(q_0, AA) (q_1, A) (q_1, \lambda)$	$(q_0, BA)$	
$(q_0, Y)$	$(q_0, AY)$	$(q_0, BY) (q_1, Y) (q_2, \lambda)$	
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB) (q_1, B) (q_1, \lambda)$	
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, X)$	$(q_2, \lambda)$		
$(q_1, B)$		$(q_1, \lambda)$	
$(q_1, Y)$		$(q_2, \lambda)$	

Observe que  $M$  reconhece a linguagem de todos os palíndromos diferentes de  $\lambda$ .

Determine o grau de não-determinismo explícito de  $M$  e construa, usando o algoritmo do teorema 6.4.3 um APN livre de não-determinismo explícito equivalente a  $M$ .

**Resposta:** De acordo com a definição 6.4.1, o grau de não-determinismo explícito de  $M$ , denotado por  $\deg(M)$  é

$$\begin{aligned}
 \deg(M) &= \max\{|\delta(q, a, A)| / (q, a, A) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma\} \\
 &= \max\{|\delta(q, a, A)| / (q, a, A) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \text{ e } |\delta(q, a, A)| \geq 1\} \cup \{0\} \\
 &= \max\{|\delta(q_0, a, z)|, |\delta(q_0, b, z)|, |\delta(q_0, a, A)|, |\delta(q_0, b, A)|, \\
 &\quad |\delta(q_0, a, B)|, |\delta(q_0, b, B)|, |\delta(q_0, a, X)|, |\delta(q_0, b, X)|, \\
 &\quad |\delta(q_0, a, Y)|, |\delta(q_0, b, Y)|, |\delta(q_1, a, A)|, |\delta(q_1, b, B)|, \\
 &\quad |\delta(q_1, a, X)|, |\delta(q_1, b, Y)|\} \cup \{0\}, \\
 &= \max\{0, 1, 2, 3\} = 3
 \end{aligned}$$

Observe que este APN esta livre de  $\lambda$ -transições como requerido pelo algoritmo do teorema 6.4.3. No primeiro ciclo do “Enquanto  $\deg(M) \geq 2$  faça”, obtemos o seguinte APN:

## A.6. Soluções de Exercícios do Capítulo 6

---

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Xz)$	$(q_0, Yz)$	$(q_3, z)$
$(q_0, X)$	$(q_0, AX)$	$(q_0, BX)$	$(q_3, X)$
$(q_0, A)$	$(q_0, AA)$	$(q_0, BA)$	$(q_3, A)$
$(q_0, Y)$	$(q_0, AY)$	$(q_0, BY)$	$(q_3, Y)$
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB)$	$(q_3, B)$
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, X)$	$(q_2, \lambda)$		
$(q_1, B)$		$(q_1, \lambda)$	
$(q_1, Y)$		$(q_2, \lambda)$	
$(q_3, z)$	$(q_2, z)$	$(q_2, z)$	
$(q_3, X)$	$(q_1, X)$ $(q_2, \lambda)$		
$(q_3, A)$	$(q_1, A)$ $(q_1, \lambda)$		
$(q_3, Y)$		$(q_1, Y)$ $(q_2, \lambda)$	
$(q_3, B)$		$(q_1, B)$ $(q_1, \lambda)$	

Como ainda  $\deg(M) \geq 2$  então fazemos mais um ciclo, resultando no seguinte APN:

	$a$	$b$	$\lambda$
$(q_0, z)$	$(q_0, Xz)$	$(q_0, Yz)$	$(q_3, z)$
$(q_0, X)$	$(q_0, AX)$	$(q_0, BX)$	$(q_3, X)$
$(q_0, A)$	$(q_0, AA)$	$(q_0, BA)$	$(q_3, A)$
$(q_0, Y)$	$(q_0, AY)$	$(q_0, BY)$	$(q_3, Y)$
$(q_0, B)$	$(q_0, AB)$	$(q_0, BB)$	$(q_3, B)$
$(q_1, A)$	$(q_1, \lambda)$		
$(q_1, X)$	$(q_2, \lambda)$		
$(q_1, B)$		$(q_1, \lambda)$	
$(q_1, Y)$		$(q_2, \lambda)$	
$(q_3, z)$	$(q_2, z)$	$(q_2, z)$	
$(q_3, X)$	$(q_1, X)$		$(q_4, X)$
$(q_3, A)$	$(q_1, A)$		$(q_4, A)$
$(q_3, Y)$		$(q_1, Y)$	$(q_4, Y)$
$(q_3, B)$		$(q_1, B)$	$(q_4, B)$
$(q_4, X)$	$(q_2, \lambda)$		
$(q_4, A)$	$(q_1, \lambda)$		
$(q_4, Y)$	$(q_2, \lambda)$		
$(q_4, A)$		$(q_1, \lambda)$	

Claramente  $\deg(M) = 1$  e portanto este APN é livre de não-determinismo explícito.

## A.7 Soluções de Exercícios do Capítulo 7

1. Mostre usando o lema do bombeamento que as seguintes linguagens são livres do contexto

$$(1a) \mathcal{L} = \{a^n b^n / n \text{ é um número ímpar}\}$$

**Resposta:** Seja  $m = 6$ . Se  $w \in \mathcal{L}$  é tal que  $|w| \geq m$  então  $w = a^n b^n$  com  $n \geq \frac{m}{2}$  sendo ímpar. Logo fazendo  $u = a^{n-3}$ ,  $v = aa$ ,  $x = ab$ ,  $y = bb$  e  $z = b^{n-3}$  temos trivialmente que  $w = uvxyz$ , que  $|vxy| = 6 = m$  e que  $|vy| = 4 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{a^{n-3}}_u \underbrace{a^{2i}}_{v^i} \underbrace{ab}_x \underbrace{b^{2i}}_{y^i} \underbrace{b^{n-3}}_z$$

Portanto,  $uv^i xy^i z = a^{(n-3)+2i+1} b^{(n-3)+2i+1} = a^{n-2+2i} b^{n-2+2i}$ . Logo, como  $n$  é ímpar então  $n = 2k + 1$  para algum número natural  $k \geq 1$ . Portanto,  $uv^i xy^i z = a^{2k-1+2i} b^{2k-1+2i} = a^{2(k+i)-1} b^{2(k+i)-1} = a^{2(k-1+i)+1} b^{2(k-1+i)+1}$ . Como claramente  $2(k-1+i)+1$  é ímpar então podemos concluir que  $uv^i xy^i z \in \mathcal{L}$ . Logo, pelo lema do bombeamento a linguagem  $\mathcal{L}$  é livre do contexto.

$$(1c) \mathcal{L} = \{a^n b^p / n \geq p \geq 1\}$$

**Resposta:** Seja  $m = 2$ . Se  $w \in \mathcal{L}$  é tal que  $|w| \geq m$  então  $w = a^n b^p$  para algum  $n \geq p \geq 1$ . Claramente,  $|w| \geq m$ . Assim, fazendo  $u = a^{n-1}$ ,  $v = a$ ,  $x = \lambda$ ,  $y = b$  e  $z = b^{p-1}$  temos trivialmente que  $w = uvxyz$ , que  $|vxy| = 2 = m$  e que  $|vy| = 2 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{a^{n-1}}_u \underbrace{a^i}_{v^i} \underbrace{\lambda}_x \underbrace{b^i}_{y^i} \underbrace{b^{p-1}}_z$$

Portanto,  $uv^i xy^i z = a^{n-1+i} b^{p-1+i}$  e como  $n \geq p$  então  $n-1+i \geq p-1+i$ . Assim, podemos concluir que  $uv^i xy^i z \in \mathcal{L}$ . Logo, pelo lema do bombeamento a linguagem  $\mathcal{L}$  é livre do contexto.

$$(1e) \mathcal{L} = \{w \in \{a, b\}^* / \mathcal{N}_a(w) \text{ é par}\}$$

**Resposta:** Seja  $m = 3$  e  $w \in \mathcal{L}$  tal que  $|w| \geq m$ . Então  $w = a_1 a_2 a_3 w'$  para algum  $a_1$ ,  $a_2$  e  $a_3$  em  $\{a, b\}$  e  $w \in \Sigma^*$ . Se  $a_1 = a_3 = b$  então fazendo  $u = \lambda$ ,  $v = b$ ,  $x = a_2$ ,  $y = b$  e  $z = w'$ , temos que  $w = uvxyz$ , que  $|vxy| = 3 = m$  e que  $|vy| = 2 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{\lambda}_u \underbrace{b^i}_{v^i} \underbrace{a_2}_x \underbrace{b^i}_{y^i} \underbrace{w'}_z$$

## A.7. Soluções de Exercícios do Capítulo 7

---

Portanto,  $uv^i xy^i z = b^i a_2 b^i w'$ . Logo,  $\mathcal{N}_a(uv^i xy^i z) = \mathcal{N}_a(a_2 w') = \mathcal{N}_a w$  que como sabemos é par. Logo, podemos concluir que para este caso  $uv^i xy^i z \in \mathcal{L}$ .

Uma outra possibilidade é que  $a_1 = a_3 = a$  então fazendo  $u = \lambda$ ,  $v = a$ ,  $x = a_2$ ,  $y = a$  e  $z = w'$ , temos que  $w = uvxyz$ , que  $|vxy| = 3 = m$  e que  $|vy| = 2 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{\lambda}_{u} \underbrace{a^i}_{v^i} \underbrace{a_2}_{x} \underbrace{a^i}_{y^i} \underbrace{w'}_{z}$$

Portanto,  $uv^i xy^i z = a^i a_2 a^i w'$ . Logo,  $\mathcal{N}_a(uv^i xy^i z) = \mathcal{N}_a(a^i a_2 a^i w') = \mathcal{N}_a(a_2 w') + 2i = \mathcal{N}_a w - 2 + 2i$ . Como  $\mathcal{N}_a w$  é par então  $\mathcal{N}_a w - 2 + 2i$  também é par. Assim, podemos concluir que para este caso  $uv^i xy^i z \in \mathcal{L}$ .

Se  $a_1 = a_2 = b$  então fazendo  $u = \lambda$ ,  $v = b$ ,  $x = \lambda$ ,  $y = b$  e  $z = a_3 w'$ , temos que  $w = uvxyz$ , que  $|vxy| = 2 \leq m$  e que  $|vy| = 2 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{\lambda}_{u} \underbrace{b^i}_{v^i} \underbrace{\lambda}_{x} \underbrace{b^i}_{y^i} \underbrace{a_3 w'}_{z}$$

Portanto,  $uv^i xy^i z = b^i b^i a_3 w'$ . Logo,  $\mathcal{N}_a(uv^i xy^i z) = \mathcal{N}_a(a_3 w') = \mathcal{N}_a w$  que como sabemos é par. Logo, podemos concluir que para este caso  $uv^i xy^i z \in \mathcal{L}$ .

Finalmente se  $a_1 = a_2 = a$  então fazendo  $u = \lambda$ ,  $v = a$ ,  $x = \lambda$ ,  $y = a$  e  $z = a_3 w'$ , temos que  $w = uvxyz$ , que  $|vxy| = 2 \leq m$  e que  $|vy| = 2 \geq 1$ . Ou seja  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. Seja  $i \in \mathbb{N}$  qualquer, então

$$uv^i xy^i z = \underbrace{\lambda}_{u} \underbrace{a^i}_{v^i} \underbrace{\lambda}_{x} \underbrace{a^i}_{y^i} \underbrace{a_3 w'}_{z}$$

Portanto,  $uv^i xy^i z = a^i a^i a_3 w'$ . Logo,  $\mathcal{N}_a(uv^i xy^i z) = \mathcal{N}_a(a^i a^i a_3 w') = \mathcal{N}_a(a_3 w') + 2i = \mathcal{N}_a w - 2 + 2i$ . Como  $\mathcal{N}_a w$  é par então  $\mathcal{N}_a w - 2 + 2i$  também é par. Assim, podemos concluir que para este caso  $uv^i xy^i z \in \mathcal{L}$ .

Observe que embora estes casos não sejam mutuamente excludentes eles cobrem todas as possibilidades.

$$(1g) \mathcal{L} = \{w \in \{a, b\}^* / w \neq w^R\}$$

**Resposta:** Seja  $m = 5$  e  $w \in \mathcal{L}$  tal que  $|w| \geq 5$ . Usaremos  $a_i$  para denotar o  $i$ -ésimo símbolo de esquerda a direita da cadeia  $w$ .

Se  $|w|$  for par, então analisaremos três casos

1. Se  $a_{\frac{|w|}{2}} = a_{\frac{|w|}{2}+1}$  então necessariamente  $a_{\frac{|w|}{2}-k} \neq a_{\frac{|w|}{2}+k+1}$  para algum  $k \geq 1$ . Assim, bombeando igualitariamente os símbolos do meio da cadeia ( $a_{\frac{|w|}{2}}$  e  $a_{\frac{|w|}{2}+1}$ ) a simetria positional e desigualdade entre  $a_{\frac{|w|}{2}-k}$  e  $a_{\frac{|w|}{2}+k+1}$  permanece. Assim, seja  $u = a_1 \dots a_{\frac{|w|}{2}-1}$ ,  $v = a_{\frac{|w|}{2}}$ ,  $x = \lambda$ ,  $y = a_{\frac{|w|}{2}+1}$  e  $z = a_{\frac{|w|}{2}+2} \dots a_{|w|}$ .

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

2. Se  $a_{\frac{|w|}{2}} \neq a_{\frac{|w|}{2}+1}$  e  $a_{\frac{|w|}{2}-1} = a_{\frac{|w|}{2}+2}$ . Neste caso fazemos  $u = a_1 \dots a_{\frac{|w|}{2}-2}$ ,  $v = a_{\frac{|w|}{2}}$ ,  $x = a_{\frac{|w|}{2}}^2 a_{\frac{|w|}{2}+1}$ ,  $y = a_{\frac{|w|}{2}+2}$  e  $z = a_{\frac{|w|}{2}+3} \dots a_{|w|}$ .
3. Se  $a_{\frac{|w|}{2}} \neq a_{\frac{|w|}{2}+1}$  e  $a_{\frac{|w|}{2}-1} \neq a_{\frac{|w|}{2}+2}$ . Neste caso fazemos  $u = a_1 \dots a_{\frac{|w|}{2}-1}$ ,  $v = a_{\frac{|w|}{2}}$ ,  $x = \lambda$ ,  $y = a_{\frac{|w|}{2}+1}$  e  $z = a_{\frac{|w|}{2}+2} \dots a_{|w|}$ .

Claramente  $u, v, x, y$  e  $z$  satisfazem as condições (7.1) e (7.2) do lema do bombeamento. E para qualquer  $i \in \mathbb{N}$ , temos que  $uv^i xy^i z \in \mathcal{L}$ .

O caso de  $|w|$  é análogo ao anterior, só considerando sempre o símbolo central como sendo o centro do  $x$ .

3. Mostre que as seguintes linguagens, sobre o alfabeto  $\{a, b, c\}$ , não são livres do contexto

(3a)  $\mathcal{L} = \{a^k b^m c^n / n = km\}$

**Resposta:** Seja  $m$  qualquer e  $w = a^m b^m c^{m^2}$ . Se  $w = uvxyz$  e  $|vxy| \leq m$  e  $|vy| \geq 1$  então temos os seguintes cinco casos

$$\begin{array}{c} a \dots a \underbrace{a \dots a}_{vxy} a \dots ab^m c^{m^2} \\ u \qquad \qquad \qquad z \\ \\ a^m b \dots b \underbrace{b \dots b}_{vxy} b \dots bc^{m^2} \\ u \qquad \qquad \qquad z \\ \\ a^m b^m c \dots c \underbrace{c \dots c}_{vxy} c \dots c \\ u \qquad \qquad \qquad z \\ \\ a \dots a \underbrace{a \dots ab}_{vxy} \dots b b \dots bc^{m^2} \\ u \qquad \qquad \qquad z \\ \\ a^m b \dots b \underbrace{b \dots bc}_{vxy} \dots c c \dots c \\ u \qquad \qquad \qquad z \end{array}$$

Os três primeiros são análogos assim como os dois últimos. Assim, só analisaremos o primeiro e quarto caso.

Para o primeiro caso, existe  $m_1, \dots, m_5$  tais que  $u = a^{m_1}$ ,  $v = a^{m_2}$ ,  $x = a^{m_3}$ ,  $y = a^{m_4}$  e  $z = a^{m_5} b^m c^{m^2}$ . Logo,  $m = \sum_{i=1}^5 m_i$  e  $m_2 + m_4 \geq 1$ . Porém,

$$\begin{aligned} uv^2 xy^2 z &= a^{m_1} a^{2m_2} a^{m_3} a^{2m_4} a^{m_5} b^m c^{m^2} \\ &= a^{m_1+m_2+m_2+m_3+m_4+m_4+m_5} b^m c^{m^2} \\ &= a^{m+m_2+m_4} b^m c^{m^2} \end{aligned}$$

Como  $m_2 + m_4 \geq 1$ , temos que  $a^{m+m_2+m_4} b^m c^{m^2} \notin \mathcal{L}$ .

Para o segundo caso, temos varias possíveis maneiras de distribuir o  $vxy$ . Nos analisaremos aqui só duas delas, as outras são análogas: na qual o  $v$  só contenha  $a$ 's, e  $y$  só  $b$ 's, e outra em que um deles mixture  $a$ 's com  $b$ 's.

Para o primeiro desses casos, teríamos que  $u = a^{m_1}$ ,  $v = a^{m_2}$ ,  $x = a^{m_3} b^{m_4}$ ,  $y = b^{m_5}$  e  $z = b^{m_6} c^{m^2}$ , com  $m_1 + m_2 + m_3 = m$ ,  $m_4 + m_5 + m_6 = m$  e  $m_2 + m_5 \geq 1$ .

Porém,

## A.7. Soluções de Exercícios do Capítulo 7

---

$$\begin{aligned}
 uv^2xy^2z &= a^{m_1}a^{2m_2}a^{m_3}b^{m_4}b^{2m_5}b^{m_6}c^{m^2} \\
 &= a^{m_1+m_2+m_2+m_3}b^{m_4+m_5+m_5+m_6}c^{m^2} \\
 &= a^{m+m_2}b^{m+m_5}c^{m^2}
 \end{aligned}$$

Como  $m_2 + m_5 \geq 1$ , temos que  $a^{m+m_2}b^{m+m_5}c^{m^2} \notin \mathcal{L}$ .

Analogamente, para o segundo desses casos teríamos que  $u = a^{m_1}$ ,  $v = a^{m_2}b^{m_3}$ ,  $x = b^{m_4}$ ,  $y = b^{m_5}$  e  $z = b^{m_6}c^{m^2}$ , com  $m_1 + m_2 = m$ ,  $m_3 + m_4 + m_5 + m_6 = m$  e  $m_2 \geq 1$  e  $m_3 \geq 1$ .

Porém,  $uv^2xy^2z = a^{m_1}a^{m_2}b^{m_3}a^{m_2}b^{m_3}b^{m_4}b^{2m_5}b^{m_6}c^{m^2}$ . Como  $m_2 \geq 1$  assim como  $m_3 \geq 1$ , temos que  $uv^2xy^2z \notin \mathcal{L}$ .

Logo, para qualquer forma de decompor  $w$  em cinco partes satisfazendo as condições do corolário 7.1.4 temos que  $uv^i xy^i z \notin \mathcal{L}$ , para algum  $i = 0, 1, 2, \dots$  e portanto por esse corolário podemos concluir que  $\mathcal{L}$  não é livre do contexto.

(3c)  $\mathcal{L} = \{a^k b^m c^n / m > k \text{ e } n > k\}$

**Resposta:** Seja  $m$  qualquer e  $w = a^m b^{m+1} c^{m+1}$ . Se  $w = uvxyz$  e  $|vxy| \leq m$  e  $|vy| \geq 1$  então temos os seguintes cinco casos

$$\begin{array}{c}
 \underbrace{a \dots a}_{u} \underbrace{a \dots a}_{vxy} \underbrace{a \dots ab^m c^{m^2}}_{z} \\
 \underbrace{a^m b \dots b}_{u} \underbrace{b \dots b}_{vxy} \underbrace{b b \dots bc^{m^2}}_{z} \\
 \underbrace{a^m b^m}_{u} \underbrace{c \dots c}_{vxy} \underbrace{c \dots c}_{z} \\
 \underbrace{a \dots a}_{u} \underbrace{a \dots ab}_{vxy} \underbrace{\dots b b \dots bc^{m^2}}_{z} \\
 \underbrace{a^m b \dots b}_{u} \underbrace{b \dots bc}_{vxy} \underbrace{\dots c c \dots c}_{z}
 \end{array}$$

O segundo com o terceiro caso são análogos assim como os dois últimos. Assim, só analisaremos o primeiro, segundo e quarto caso.

Para o primeiro caso, existe  $m_1, \dots, m_5$  tais que  $u = a^{m_1}$ ,  $v = a^{m_2}$ ,  $x = a^{m_3}$ ,  $y = a^{m_4}$  e  $z = a^{m_5}b^{m+1}c^{m+1}$ . Logo,  $m = \sum_{i=1}^5 m_i$  e  $m_2 + m_4 \geq 1$ . Porém, para  $i = 2$  temos que

$$\begin{aligned}
 uv^i xy^i z &= a^{m_1}a^{2m_2}a^{m_3}a^{2m_4}a^{m_5}b^{m+1}c^{m+1} \\
 &= a^{m_1+m_2+m_2+m_3+m_4+m_4+m_5}b^{m+1}c^{m+1} \\
 &= a^{m+m_2+m_4}b^{m+1}c^{m+1}
 \end{aligned}$$

Como  $m_2 + m_4 \geq 1$ , temos que  $uv^i xy^i z \notin \mathcal{L}$ .

O segundo e terceiro caso é análogo ao primeiro, mas considerando  $i = 0$  em vez de  $i = 2$ .

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

Para o quarto caso temos varias possíveis maneiras de distribuir o  $vxy$ . Nos analisaremos aqui só duas delas, as outras são análogas: na qual o  $v$  só contenha  $a$ 's, e  $y$  só  $b$ 's, e outra em que um deles mixture  $a$ 's com  $b$ 's.

Para o primeiro desses casos, teríamos que  $u = a^{m_1}$ ,  $v = a^{m_2}$ ,  $x = a^{m_3}b^{m_4}$ ,  $y = b^{m_5}$  e  $z = b^{m_6}c^{m^2}$ , com  $m_1 + m_2 + m_3 = m$ ,  $m_4 + m_5 + m_6 = m + 1$  e  $m_2 + m_5 \geq 1$ . Como  $m_2 + m_5 \geq 1$ , temos que ou  $m_2 \geq 1$  ou  $m_5 \geq 1$ . Se  $m_2 \geq 1$  então

$$\begin{aligned} uv^2xy^2z &= a^{m_1}a^{2m_2}a^{m_3}b^{m_4}b^{2m_5}b^{m_6}c^{m+1} \\ &= a^{m_1+m_2+m_2+m_3}b^{m_4+m_5+m_5+m_6}c^{m+1} \\ &= a^{m+m_2}b^{m+m_5}c^{m+1} \end{aligned}$$

E portanto  $a^{m+m_2}b^{m+m_5}c^{m+1} \notin \mathcal{L}$ , pois  $m + m_2 \geq m + 1$ .

Se  $m_5 \geq 1$  e  $m_2 = 0$  então

$$\begin{aligned} uv^0xy^0z &= a^{m_1}a^{0m_2}a^{m_3}b^{m_4}b^{0m_5}b^{m_6}c^{m+1} \\ &= a^{m_1+m_3}b^{m_4+m_6}c^{m+1} \\ &= a^mb^{m+1-m_5}c^{m+1} \end{aligned}$$

E portanto  $a^mb^{m+1-m_5}c^{m+1} \notin \mathcal{L}$ , pois  $m \geq m + 1 - m_5$ .

Analogamente, para o segundo desses casos teríamos que  $u = a^{m_1}$ ,  $v = a^{m_2}b^{m_3}$ ,  $x = b^{m_4}$ ,  $y = b^{m_5}$  e  $z = b^{m_6}c^{m+1}$ , com  $m_1 + m_2 = m$ ,  $m_3 + m_4 + m_5 + m_6 = m + 1$  e  $m_2 \geq 1$  e  $m_3 \geq 1$ .

Porém,  $uv^2xy^2z = a^{m_1}a^{m_2}b^{m_3}a^{m_2}b^{m_3}b^{m_4}b^{m_5}b^{m_6}c^{m+1}$ . Como  $m_2 \geq 1$  assim como  $m_3 \geq 1$ , temos que  $uv^2xy^2z \notin \mathcal{L}$ .

O quinto e último caso, é análogo ao quarto.

Logo, para qualquer forma de decompor  $w$  em cinco partes satisfazendo as condições do corolário 7.1.4 temos que  $uv^ixy^iz \notin \mathcal{L}$ , para algum  $i = 0, 1, 2, \dots$  e portanto por esse corolário podemos concluir que  $\mathcal{L}$  não é livre do contexto.

5. Mostre que a classe das linguagens livres do contexto é fechada sobre os seguintes operadores:

(5a) Homomorfismos,

**Resposta:** Seja  $h : \Sigma_1 \longrightarrow \Sigma_2^*$  um homomorfismo. Se  $\mathcal{L}$  é uma linguagem livre do contexto, sobre o alfabeto  $\Sigma_1$ , então devemos mostrar que sua imagem homomorfa é uma linguagem livre do contexto sobre o alfabeto  $\Sigma_2$ .

Seja  $G_1 = \langle V, \Sigma_1, S, P_1 \rangle$  uma gramática livre do contexto tal que  $L(G_1) = \mathcal{L}$ . Então  $G_2 = \langle V, \Sigma_2, S, P_2 \rangle$  onde  $P_2 = \{A \rightarrow h^*(y) / A \rightarrow y \in P_1\}$  com  $h^* : (\Sigma_1 \cup V)^* \rightarrow (\Sigma_2 \cup V)^*$  definida recursivamente como segue:

1.  $h^*(\lambda) = \lambda$ ,
2.  $h^*(wA) = h^*(w)A$  e
3.  $h^*(wa) = h^*(w)h(a)$

Aqui  $w \in (\Sigma_1^* \cup V)^*$ ,  $A \in V$  e  $a \in \Sigma_1$ .

Claramente,  $h(L(G_1)) = L(G_2)$  e portanto as linguagens livres do contexto são fechadas sobre homomorfismos arbitrários.

(5c) O operador reverso

## A.7. Soluções de Exercícios do Capítulo 7

---

**Resposta:** Seja  $G = \langle V, T, S, P \rangle$  uma gramática livre do contexto qualquer. Trivialmente,  $G^R = \langle V, T, S, P^R \rangle$  onde  $P^R = \{A \rightarrow y^R / A \rightarrow y \in P\}$  é uma gramática livre do contexto tal que  $L(G^R) = L(G)^R$  e por tanto as linguagens livres do contexto são fechadas sobre o operador reverso.

- (5e) Seja  $\mathcal{L}$  um linguagem sobre o alfabeto  $\{0, 1\}$  e  $\neg : \Sigma^* \rightarrow \Sigma^*$  definido por  $\neg(\lambda) = \lambda$ ,  $\neg(w0) = 1\neg(w)$  e  $\neg(w1) = 0\neg(w)$ . Define  $\neg\mathcal{L}$  como sendo

$$\neg\mathcal{L} = \{\neg(w) / w \in \mathcal{L}\}$$

**Resposta:** Seja  $G = \langle V, \{0, 1\}, S, P \rangle$  uma gramática livre do contexto qualquer e  $\bar{\phantom{x}} : (V \cup \{0, 1\})^* \rightarrow (V \cup \{0, 1\})^*$  definido recursivamente como segue:

1.  $\bar{\lambda} = \lambda$ ,
2.  $\bar{wA} = A\bar{w}$ ,
3.  $\bar{w0} = 1\bar{w}$  e
4.  $\bar{w1} = 0\bar{w}$

Trivialmente,  $\neg G = \langle V, \{0, 1\}, S, \neg P \rangle$  onde  $\neg P = \{A \rightarrow \bar{y} / A \rightarrow y \in P\}$  é uma gramática livre do contexto tal que  $L(\neg G) = \neg L(G)$  e por tanto as linguagens livres do contexto são fechadas sobre o operador  $\neg$ .

- (5g) Seja  $\mathcal{L}$  uma linguagem qualquer sobre um alfabeto  $\Sigma$  e  $\rho : \Sigma^* \rightarrow \Sigma^*$  definido por  $\rho(\lambda) = \lambda$  e  $\rho(wa) = \rho(w)aa$  para todo  $w \in \Sigma^*$  e  $a \in \Sigma$ . Define  $\mathcal{L}^\rho$  como sendo

$$\mathcal{L}^\rho = \{\rho(w) / w \in \mathcal{L}\}$$

**Resposta:** Seja  $G = \langle V, \Sigma, S, P \rangle$  uma gramática livre do contexto qualquer e  $\rho : (V \cup \Sigma)^* \rightarrow (V \cup \Sigma)^*$  definido recursivamente como segue:

1.  $\rho(\lambda) = \lambda$ ,
2.  $\rho(wA) = \rho(w)A$  e
3.  $\rho(wa) = \rho(w)aa$ .

Trivialmente,  $\rho(G) = \langle V, \Sigma, S, \rho(P) \rangle$  onde  $\rho(P) = \{A \rightarrow \rho(y) / A \rightarrow y \in P\}$  é uma gramática livre do contexto tal que  $L(\rho(G)) = \rho(L(G))$  e por tanto as linguagens livres do contexto são fechadas sobre o operador  $\neg$ .

7. Mostre que existe um algoritmo que determina quando a linguagem gerada por alguma gramática livre do contexto contém qualquer palavra de comprimento menor que algum número  $n$ .

**Resposta:** Seja  $G$  uma gramática livre do contexto. Transforme  $G$  à forma normal de Chomsky. Observe que neste caso, a cada derivação de uma cadeia aumentamos em um o tamanho da forma sentencial (pois trocamos uma variável por duas) ou deixamos do mesmo tamanho, mas diminuímos o número de variáveis em um. Por tanto seria suficiente usar o algoritmo de pesquisa exaustiva e ir gerando primeiro todas as formas sentenciais de tamanho 1, depois as de tamanho 2, e assim por diante. É claro que caso não for gerada uma cadeia só de símbolos terminais antes de gerar todas as formas sentenciais de tamanho  $n$ , então  $G$  não gerará nunca uma palavra de tamanho menor que  $n$ .

## A.8 Soluções de Exercícios do Capítulo 8

4. Construir máquinas de Turing que reconheçam as seguintes linguagens sobre  $\{a, b\}$ .

(4a)  $\mathcal{L} = L(aba^*b)$

**Resposta:** Seja  $M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_3\} \rangle$ , com

	a	b	$\square$
$q_0$	$(q_1, a, D)$		
$q_1$		$(q_2, b, D)$	
$q_2$	$(q_2, a, D)$	$(q_3, b, D)$	
$q_3$			

Claramente  $L(M) = L(aba^*b)$ .

(4c)  $\mathcal{L} = \{w / w = \tilde{w}^R\}$ , onde  $\tilde{\lambda} = \lambda$ ,  $\tilde{w}a = \tilde{w}b$  e  $\tilde{w}b = \tilde{w}a$ .

**Resposta:** Primeiro é importante entendermos que cadeias fazem parte e quais não fazem parte desta linguagem. Então alguns exemplos de cadeias que estão em  $\mathcal{L}$ :  $\lambda$ ,  $aababb$ ,  $aaaabbbb$  e  $abaabbaabbab$ . Observe que  $abaabbaabbab \in \mathcal{L}$ , pois  $(abaabbaabbab) = babbaabbaabba$  e portanto  $(abaabbaabbab)^R = (babbaabbaabba)^R = abaabbaabbab$ . Assim, claramente  $w \in \mathcal{L}$  se e somente se  $w = \lambda$ , ou  $w = aw'b$  ou  $w = bw'a$  para algum  $w' \in \mathcal{L}$ .

Seja  $M = \langle \{q_0, \dots, q_5\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_5\} \rangle$ , com

	a	b	$\square$
$q_0$	$(q_1, \square, D)$	$(q, \square, D)$	$(q, \square, D)$
$q_1$	$(q_1, a, D)$	$(q_1, b, D)$	$(q_2, \square, E)$
$q_2$		$(q_3, \square, E)$	
$q_3$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_0, \square, D)$
$q_4$	$(q_3, \square, E)$		
$q_5$			

Claramente  $L(M) = \mathcal{L}$ .

(4e)  $\mathcal{L} = \{a^n b^m / n \geq 1, n \leq m \leq 3n\}$

**Resposta:** Seja  $M = \langle \{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_3\} \rangle$ , com

	a	b	$\square$
$q_0$	$(q_1, a, D)$		
$q_1$		$(q_2, b, D)$	
$q_2$	$(q_2, a, D)$	$(q_3, b, D)$	
$q_3$			

Claramente  $L(M) = \mathcal{L}$ .

(4g)  $\mathcal{L} = \{w / \mathcal{N}_a(w) = \mathcal{N}_b(w)\}$

## A.8. Soluções de Exercícios do Capítulo 8

---

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_4\}, \{a, b\}, \{a, b, X, \square\}, \delta, \square, \{q_4\} \rangle$ , com

	$a$	$b$	$X$	$\square$
$q_0$	$(q_1, \square, D)$	$(q_3, \square, D)$	$(q_0, \square, D)$	$(q_4, \square, D)$
$q_1$	$(q_1, a, D)$	$(q_2, X, E)$	$(q_1, X, D)$	
$q_2$	$(q_2, a, E)$	$(q_2, b, E)$	$(q_2, X, E)$	$(q_0, \square, D)$
$q_3$	$(q_2, X, E)$	$(q_3, b, D)$	$(q_3, X, D)$	
$q_4$				

Testando as cadeias  $aababbabba \in \mathcal{L}$  e  $aabbabba \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

$$(4i) \quad \mathcal{L} = \{w / \mathcal{N}_a(w) \neq 2\mathcal{N}_b(w)\}$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_6\}, \{a, b\}, \{a, b, X, \square\}, \delta, \square, \{q_6\} \rangle$ , com

	$a$	$b$	$X$	$\square$
$q_0$	$(q_1, X, D)$	$(q_0, b, D)$	$(q_0, X, D)$	$(q_5, \square, E)$
$q_1$	$(q_2, X, E)$	$(q_1, b, D)$	$(q_1, X, D)$	$(q_6, \square, E)$
$q_2$		$(q_2, b, E)$	$(q_2, X, E)$	$(q_3, \square, D)$
$q_3$	$(q_3, a, D)$	$(q_4, \square, E)$	$(q_3, X, D)$	$(q_6, \square, E)$
$q_4$	$(q_4, a, E)$		$(q_4, X, E)$	$(q_0, \square, D)$
$q_5$		$(q_6, b, D)$	$(q_5, X, E)$	
$q_6$				

Testando as cadeias  $aababbaabaa \in \mathcal{L}$  e  $aaabbabaaa \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

$$(4k) \quad \mathcal{L} = \{a^n b^m a^{n+m} / n \geq 0, m \geq 1\}$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_8\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_8\} \rangle$ , com

	$a$	$b$	$\square$
$q_0$	$(q_1, \square, D)$	$(q_4, \square, D)$	
$q_1$	$(q_1, a, D)$	$(q_1, b, D)$	$(q_2, \square, E)$
$q_2$	$(q_3, \square, E)$		
$q_3$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_0, \square, D)$
$q_4$	$(q_4, a, D)$	$(q_4, b, D)$	$(q_5, \square, E)$
$q_5$	$(q_6, \square, E)$		
$q_6$	$(q_6, a, E)$	$(q_6, b, E)$	$(q_7, \square, D)$
$q_7$		$(q_4, \square, D)$	$(q_8, \square, D)$
$q_8$			

Testando as cadeias  $aabbbaaaaa \in \mathcal{L}$ ,  $aabbabaaaa \notin \mathcal{L}$  e  $aabbaaa \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

$$(4m) \quad \mathcal{L} = \{a^k b^m a^n / 0 < k < n \text{ e } m = n + 2\}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_9\}, \{a, b\}, \{a, b, A, \square\}, \delta, \square, \{q_9\} \rangle$ , com

	$a$	$b$	$A$	$\square$
$q_0$	$(q_1, \square, D)$	$(q_4, \square, D)$		
$q_1$	$(q_1, a, D)$	$(q_1, b, D)$	$(q_2, A, E)$	$(q_2, \square, E)$
$q_2$	$(q_3, A, E)$			
$q_3$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_0, \square, D)$	
$q_4$	$(q_4, a, D)$	$(q_4, b, D)$	$(q_4, a, D)$	$(q_5, \square, E)$
$q_5$	$(q_6, \square, E)$	$(q_8, \square, E)$		
$q_6$	$(q_6, a, E)$	$(q_6, b, E)$		$(q_7, \square, D)$
$q_7$		$(q_4, \square, D)$		
$q_8$				$(q_9, \square, D)$
$q_9$				

Testando as cadeias  $aabbbaaaa \in \mathcal{L}$ ,  $abbabbaa \notin \mathcal{L}$  e  $aabbbaaa \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

$$(4o) \quad \mathcal{L} = \{a^n b^{2n} a^n / n \text{ é ímpar}\}$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_{10}\}, \{a, b\}, \{a, b, X, \square\}, \delta, \square, \{q_{10}\} \rangle$ , com

	$a$	$b$	$X$	$\square$
$q_0$	$(q_1, a, D)$			
$q_1$	$(q_0, a, D)$	$(q_2, b, E)$		
$q_2$	$(q_2, a, E)$			$(q_3, \square, D)$
$q_3$	$(q_4, \square, D)$		$(q_9, \square, D)$	
$q_4$	$(q_4, a, D)$	$(q_5, X, D)$	$(q_4, X, D)$	
$q_5$		$(q_6, X, D)$		
$q_6$	$(q_6, a, D)$	$(q_6, b, D)$		$(q_7, \square, E)$
$q_7$	$(q_8, \square, E)$			
$q_8$	$(q_8, a, E)$	$(q_8, b, E)$	$(q_8, X, E)$	$(q_3, \square, D)$
$q_9$			$(q_9, \square, D)$	$(q_{10}, \square, D)$
$q_{10}$				

Testando as cadeias  $aaabbbaaaa \in \mathcal{L}$ ,  $aabbbaa \notin \mathcal{L}$  e  $aaabbbaaa \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

$$(4q) \quad \mathcal{L} = \{a^n b^{n+1} a^{2n} / n \geq 1\}$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_7\}, \{a, b\}, \{a, b, X, \square\}, \delta, \square, \{q_7\} \rangle$ , com

## A.8. Soluções de Exercícios do Capítulo 8

---

	$a$	$b$	$X$	$\square$
$q_0$	$(q_1, \square, D)$		$(q_5, \square, D)$	
$q_1$	$(q_1, a, D)$	$(q_2, X, D)$		
$q_2$	$(q_3, X, D)$	$(q_2, b, D)$		
$q_3$	$(q_4, X, E)$			
$q_4$	$(q_4, a, E)$	$(q_4, X, E)$	$(q_4, X, E)$	$(q_0, \square, D)$
$q_5$		$(q_6, \square, D)$	$(q_5, \square, D)$	
$q_6$			$(q_6, \square, D)$	$(q_7, \square, E)$
$q_7$				

Testando as cadeias  $aaabbbbaaaaaa \in \mathcal{L}$ ,  $aabbbbbaaaa \notin \mathcal{L}$  e  $aaabbbbaaaaa \notin \mathcal{L}$  é possível se convencer que  $L(M) = \mathcal{L}$ .

6. Seja  $\Sigma = \{a, b\}$ . Construir uma máquina de Turing que compute as seguintes funções  
 $f : \Sigma^* \longrightarrow \Sigma^*$

$$(6a) f(w) = w^R$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_9\}, \{a, b\}, \{a, b, A, B, \square\}, \delta, \square, \{q_9\} \rangle$ , com

	$a$	$b$	$A$	$B$	$\square$
$q_0$	$(q_1, A, D)$	$(q_5, B, D)$	$(q_7, A, D)$	$(q_7, A, D)$	
$q_1$	$(q_1, a, D)$	$(q_1, b, D)$	$(q_2, A, E)$	$(q_2, B, E)$	$(q_2, \square, E)$
$q_2$	$(q_4, A, E)$	$(q_3, A, E)$	$(q_7, A, D)$		
$q_3$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_0, B, D)$	$(q_0, A, D)$	
$q_4$	$(q_4, a, E)$	$(q_4, b, E)$	$(q_0, A, D)$	$(q_0, B, D)$	
$q_5$	$(q_5, a, D)$	$(q_5, b, D)$	$(q_6, A, E)$	$(q_6, B, E)$	$(q_6, \square, E)$
$q_6$	$(q_3, B, E)$	$(q_4, B, E)$		$(q_7, A, D)$	
$q_7$			$(q_7, A, D)$	$(q_7, A, D)$	$(q_8, \square, E)$
$q_8$			$(q_8, a, E)$	$(q_8, b, E)$	$(q_9, \square, D)$
$q_9$					

Testando as cadeias  $aaba$  e  $baaba$  é possível se convencer que  $M$  computa  $f$ .

$$(6c) f(w) = w\bar{w} \text{ onde } \bar{\lambda} = \lambda, \bar{w}a = \bar{w}b \text{ e } \bar{w}b = \bar{w}a$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_6\}, \{a, b\}, \{a, b, A, B, \square\}, \delta, \square, \{q_6\} \rangle$ , com

	$a$	$b$	$A$	$B$	$\square$
$q_0$	$(q_1, A, D)$	$(q_4, B, D)$			
$q_1$	$(q_1, a, D)$	$(q_1, b, D)$	$(q_1, A, D)$	$(q_1, B, D)$	$(q_2, B, E)$
$q_2$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_2, A, E)$	$(q_2, B, E)$	$(q_5, \square, D)$
$q_3$	$(q_3, a, E)$	$(q_3, b, E)$	$(q_0, A, D)$	$(q_0, B, D)$	
$q_4$	$(q_4, a, D)$	$(q_4, b, D)$	$(q_4, A, D)$	$(q_4, B, D)$	$(q_2, A, E)$
$q_5$			$(q_5, a, D)$	$(q_5, b, D)$	$(q_6, \square, E)$
$q_6$					

Testando as cadeias  $aaba$  e  $baaba$  é possível se convencer que  $M$  computa  $f$ .

$$(6e) f(w) = a^{|\mathcal{N}_a(w)|} b^{|\mathcal{N}_b(w)|}$$

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_4\}, \{a, b\}, \{a, b, \square\}, \delta, \square, \{q_4\} \rangle$ , com

	$a$	$b$	$\square$
$q_0$	$(q_0, a, D)$	$(q_1, b, D)$	
$q_1$	$(q_2, b, E)$	$(q_1, b, D)$	$(q_4, \square, E)$
$q_2$	$(q_3, a, D)$	$(q_2, b, E)$	
$q_3$		$(q_1, a, D)$	
$q_4$			

Testando as cadeias  $aababb$  e  $baabaa$  é possível se convencer que  $M$  computa  $f$ .

8. Construir máquinas de Turing para computar as seguintes funções para  $x$  e  $y$  inteiros positivos representados em unário. Caso a função tenha dois argumentos eles viram separados por um zero.

$$(8a) f(x) = 2x$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_4\}, \{1\}, \{1, X, \square\}, \delta, \square, \{q_4\} \rangle$ , com

	$1$	$X$	$\square$
$q_0$	$(q_1, X, D)$	$(q_0, X, D)$	$(q_3, \square, E)$
$q_1$	$(q_1, 1, D)$	$(q_1, X, D)$	$(q_2, X, E)$
$q_2$	$(q_2, 1, E)$	$(q_2, X, E)$	$(q_0, \square, D)$
$q_3$		$(q_3, 1, E)$	$(q_4, \square, D)$
$q_4$			

Testando a cadeia  $1111$  é possível se convencer que  $M$  computa  $f$ .

$$(8c) f(x) = \lfloor \frac{x}{2} \rfloor$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_5\}, \{1\}, \{1, X, \square\}, \delta, \square, \{q_5\} \rangle$ , com

	$1$	$X$	$\square$
$q_0$	$(q_1, \square, D)$	$(q_4, 1, D)$	
$q_1$	$(q_2, \square, D)$	$(q_4, 1, D)$	
$q_2$	$(q_2, 1, D)$	$(q_2, X, D)$	$(q_3, X, E)$
$q_3$	$(q_3, 1, E)$	$(q_3, X, E)$	$(q_0, \square, D)$
$q_4$		$(q_4, 1, D)$	$(q_5, \square, E)$
$q_5$			

Testando as cadeias  $11111$  e  $1111$  é possível se convencer que  $M$  computa  $f$ .

$$(8e) f(x) = x \bmod 5$$

**Resposta:** Seja  $M = \langle \{q_0, \dots, q_9\}, \{1\}, \{1, \square\}, \delta, \square, \{q_9\} \rangle$ , com

## A.8. Soluções de Exercícios do Capítulo 8

---

	1	$\square$
$q_0$	$(q_1, \square, D)$	
$q_1$	$(q_2, \square, D)$	$(q_9, 1, D)$
$q_2$	$(q_3, \square, D)$	$(q_8, 1, D)$
$q_3$	$(q_4, \square, D)$	$(q_7, 1, D)$
$q_4$	$(q_5, \square, D)$	$(q_6, 1, D)$
$q_5$	$(q_1, \square, D)$	$(q_9, \square, D)$
$q_6$		$(q_7, 1, D)$
$q_7$		$(q_8, 1, D)$
$q_8$		$(q_9, 1, D)$
$q_9$		

Testando as cadeias 11111111 e 1111111111 é possível se convencer que  $M$  computa  $f$ .

(8g)

$$f(x) = \begin{cases} 0 & , \text{ se } x \text{ for par} \\ \frac{x-1}{2} & , \text{ caso contrário} \end{cases}$$

**Resposta:** Antes observe que esta função coincide com a do exercício (8c) para o caso do valor ser ímpar. Porém aqui faremos um algoritmo diferente do feito para essa função. Seja  $M = \langle \{q_0, \dots, q_6\}, \{1\}, \{1, X, \square\}, \delta, \square, \{q_6\} \rangle$ , com

	1	X	$\square$
$q_0$	$(q_1, X, D)$		$(q_4, \square, E)$
$q_1$	$(q_1, 1, D)$		$(q_2, \square, E)$
$q_2$	$(q_3, \square, E)$	$(q_5, \square, E)$	
$q_3$	$(q_3, 1, E)$	$(q_0, X, D)$	
$q_4$		$(q_4, \square, E)$	$(q_6, \square, D)$
$q_5$		$(q_5, 1, E)$	$(q_6, \square, D)$
$q_6$			

Testando as cadeias 11111 e 111111 é possível se convencer que  $M$  computa  $f$ .

(8i)

$$f(x, y) = \begin{cases} y - x & , \text{ se } y \geq x \\ 0 & , \text{ se } x > y \end{cases}$$

**Resposta:** O zero de saída neste caso será representado pela cadeia vazia. Seja  $M = \langle \{q_0, \dots, q_5\}, \{0, 1\}, \{0, 1, \square\}, \delta, \square, \{q_5\} \rangle$ , com

	1	0	$\square$
$q_0$	$(q_1, \square, D)$	$(q_5, \square, E)$	
$q_1$	$(q_1, 1, D)$	$(q_1, 0, D)$	$(q_2, \square, E)$
$q_2$	$(q_3, \square, E)$	$(q_4, \square, E)$	
$q_3$	$(q_3, 1, E)$	$(q_3, 0, E)$	$(q_0, \square, D)$
$q_4$	$(q_4, \square, E)$		$(q_5, \square, D)$
$q_5$			

## Capítulo A. Soluções de Alguns dos Exercícios Propostos

---

Testando as cadeias 111011, 11011 e 11011111 é possível se convencer que  $M$  computa  $f$ .

(8k)

$$f(x, y) = \begin{cases} x \bmod y & , \text{ se } x > y \\ x & , \text{ se } x \leq y \end{cases}$$

**Resposta:** TEM QUE AJEITAR Seja  $M = \langle \{q_0, \dots, q_5\}, \{0, 1\}, \{0, 1, X, \square\}, \delta, \square, \{q_5\} \rangle$ , com

	1	0	X	$\square$
$q_0$	$(q_1, X, D)$	$(q_4, \square, D)$		
$q_1$	$(q_1, 1, D)$	$(q_1, 0, D)$	$(q_2, X, E)$	$(q_2, \square, E)$
$q_2$	$(q_3, X, E)$	$(q_5, 0, D)$		
$q_3$	$(q_3, 1, E)$	$(q_3, 0, E)$	$(q_0, X, D)$	
$q_4$	$(q_4, \square, D)$		$(q_4, \square, D)$	$(q_9, \square, E)$
$q_5$	$(q_6, 1, E)$		$(q_5, X, D)$	$(q_6, \square, E)$
$q_6$			$(q_7, 1, E)$	
$q_7$		$(q_8, 0, E)$	$(q_7, X, E)$	
$q_8$	$(q_8, 1, E)$		$(q_9, X, D)$	

Testando as cadeias 111011, 11011 e 11011111 é possível se convencer que  $M$  computa  $f$ .

## A.8. Soluções de Exercícios do Capítulo 8

---

### Capítulo 8

22.b)  $S \longrightarrow aSB \mid aB \mid abc$

$B \longrightarrow bC$

$Cb \longrightarrow bC$

$CC \longrightarrow cc$

$cC \longrightarrow cc$

24) A solução-CP é dada pelas sequências 4-2-5-5-3-1. Isto é, da sequência  $A$  obtemos

$baab \ baba \ ab \ ab \ abaa \ aba$

e da sequência  $B$  obtemos

$baa \ bbab \ a \ a \ baba \ baaaba$

A cadeia resultante é  $baabbabaabababaaaaba.$

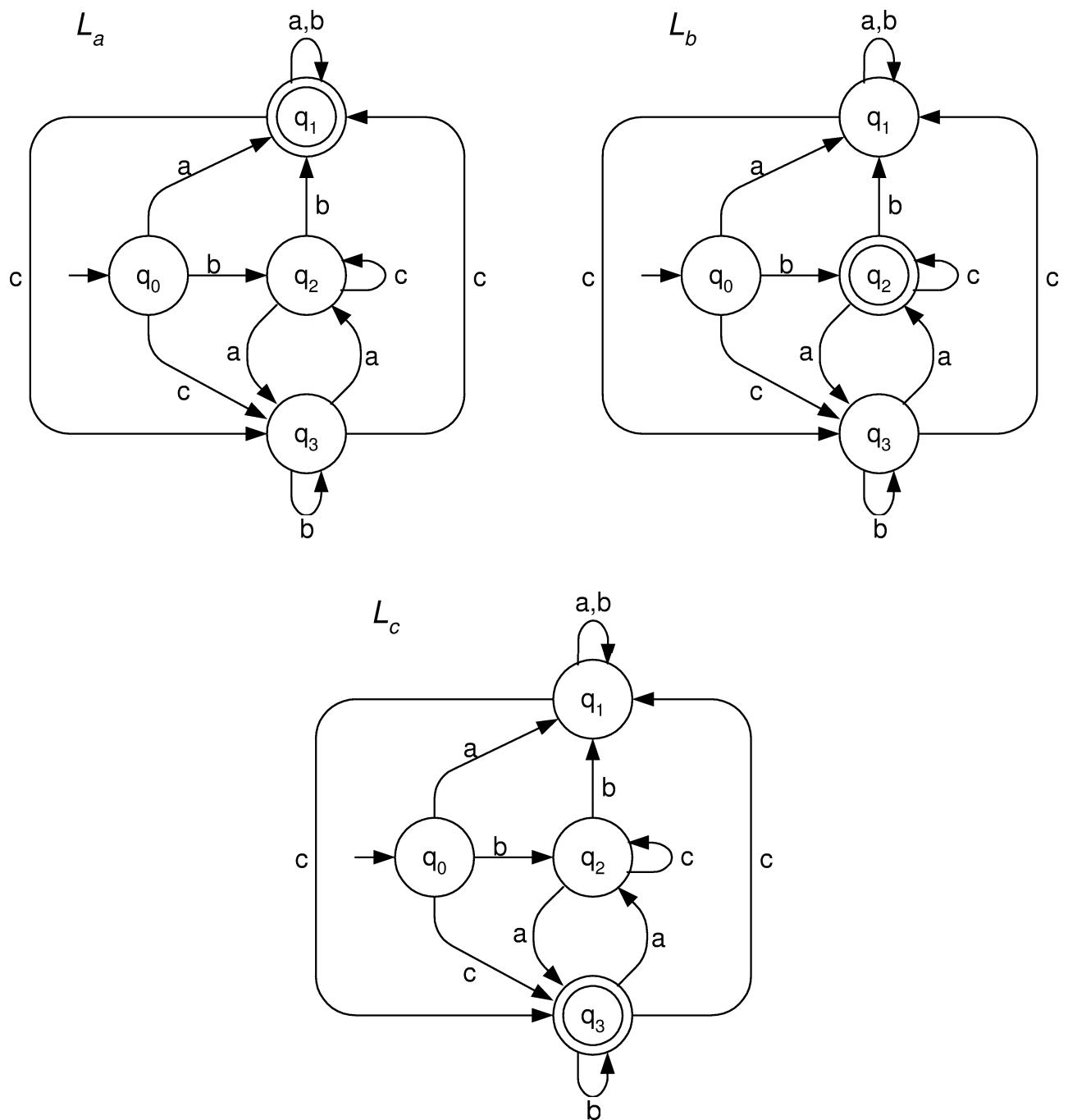


Figura A.39: AFD que reconhecem as linguagens  $\mathcal{L}_a$ ,  $\mathcal{L}_b$  e  $\mathcal{L}_c$

## A.8. Soluções de Exercícios do Capítulo 8

---

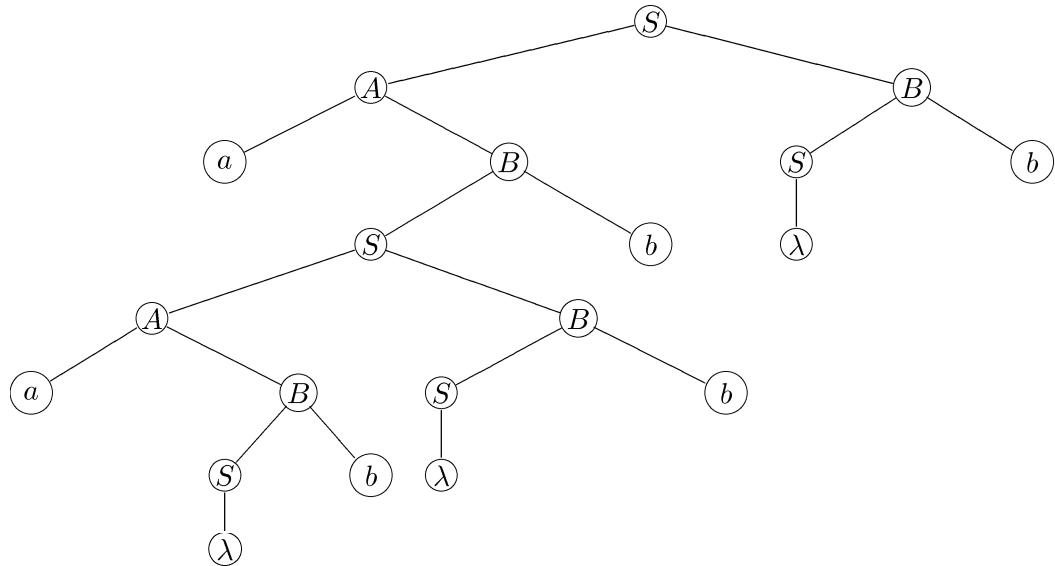


Figura A.40: Uma árvore de derivação para a cadeia  $aabbba$ .

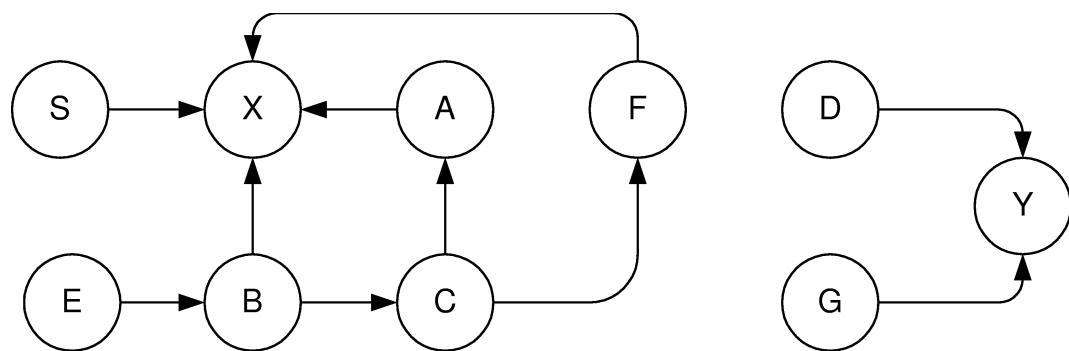


Figura A.41: Grafo de dependência entre variáveis de uma gramática livre do contexto.

# Bibliografia

- [ARBS04] Frederiko S. Araújo, Karla D.N. Ramos, Benjamín C. Bedregal and Ivan S. Silva. Papílio Cryptography Algorithm. *LNCS*, 3314:928–933, 2005.
- [ABB97] Jean-Michel Autebert, Jean Berstel and Luc Boasson. Context-Free Languages and Pushdown Automata. In *Handbook of formal languages, vol. 1: word, language, grammar*, pages 111–174, Springer-Verlag, New York, 1997.
- [AFP02] Mauricio Ayala-Rincón, Alexsandro F. da Fonseca, Haydée W. Poubel and José Siqueira. A framework to visualize equivalences between computational models of regular languages. *Information Processing Letters*, 84:5–16, 2002.
- [BA99] Benjamín C. Bedregal and Benedito M. Acioly. Effectively Given Continuous Domains: A Computable Interval Analysis. *Electronic Journal on Mathematics of Computation*, 1(0):1 – 22, 1999.
- [Bed07] Benjamín C. Bedregal. Pushdown automata free of explicit nondeterminism and an infinite hierarchy of context-free languages. *Fundamenta Informaticae*, 81(4):367–377, 2007.
- [Bed08] Benjamín C. Bedregal. Nondeterministic Linear Automata. *Journal of Computer and System Sciences*, 2008. (submetido).
- [BF08] Benjamín C. Bedregal and Santiago Figueira. On the computing power of fuzzy Turing machines. *Fuzzy Sets and Systems*, 159(9):1072–1083, 2008.
- [BS04] Yaakov Benenson and Ehud Shapiro. Molecular Computing Machines. *Dekker Encyclopedia of Nanoscience and Nanotechnology*, pag. 2043–2055, 2004.
- [Ben82] Charles H. Bennett. The thermodynamics of computation: A review. *Int. J. Theor. Phys.*, 21(12): 905–940, 1982.
- [Bir96] Rodolfo E. Biraben. *Tese de Church: Algumas questões histórico-conceituais*. Centro de Lógica, Epistemologia e História da Ciência - UNICAMP, coleção CLE Vol. 20, Campinas-SP, 1996.
- [BSS89] Lenore Blum, Michael Shub and Stephen Smale. On a theory of computation and complexity over real number: NP-completeness, recursive functions and universal machines. *Bull. of the Amer. Math. Soc.*, 21:1–46, 1989.

## BIBLIOGRAFIA

---

- [BDLS96] Dan Boneh, Christopher Dunworth, Richard J. Lipton and Jirí Sgall. On the computational power of DNA. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 71:79–94, 1996.
- [BL74] Walter S. Brainerd and Lawrence H. Landweber. *Theory of Computation*. John Wiley & Sons, New York, 1974.
- [Bra96] Vasco Brattka. Recursive characterization of computable real-valued functions and relations. *Theoretical Computer Science*, 162(1):45–77, 1996.
- [Car03] Marco Carpentieri. On the simulation of quantum Turing machines. *Theoretical Computer Science*, 304:103–128, 2003
- [CO98] Roberto L. de Carvalho e Cláudia M. G. M. de Oliveira. Modelos de computação e sistemas formais. 11<sup>a</sup> Escola de Computação, Rio de Janeiro: DCC/IM,COPPE/Sistemas, NCE/UFRJ, julho de 1998.
- [CS70] John Cocke and Jacob T. Schwartz. *Programming languages and their compilers: Preliminary notes*. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.
- [CD05] Antônio C.R. Costa and Graçaliz P. Dimuro. Interactive Computation: Stepping Stone in the Pathway From Classical to Developmental Computation. *Electronic Notes in Theoretical Computer Science*, 141(5):5–31, 2005.
- [Cut80] Nigel J. Cutland. *Computability: An introduction to recursive function theory*. Cambridge University Press, Cambridge-UK, 1980.
- [DW83] Martin D. Davis and Elaine J. Weyuker. *Computability Complexity And Languages: Fundamentals Of Theoretical Computer Science*. New York, NY: Academic Press, 1983.
- [Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London*, 400:97–117, 1985.
- [Ear70] Jay Early. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.
- [EMM06] Zoltán Ésik, Carlos Martín-Vide and Victor Mitrana, *Recent Advances in Formal Languages and Applications*. Springer, Berlim, 2006.
- [Fle01] Arthur Fleck. *Formal Models of Computation: The ultimate limits of computing*. AMAST Series in Computing Vol. 7, World Scientific Publishing, Singapore, 2001.
- [GSAS04] Dina Q. Goldin, Scott A. Smolka, Paul C. Attie and Elaine L. Sonderegger. Turing machines, transition systems, and interaction. *Information and Computation* 194(2):101–128, 2004.
- [GLW05] Jonathan Goldstine, Hing Leung, Detlef Wotschke. Measuring nondeterminism in pushdown automata, *Journal of Computer and System Sciences* 71(4):440–466, 2005.
- [Gop06] Ganesh Gopalakrishnan. *Computation Engineering: Applied Automata Theory and Logic*. Springer, Berlim, 2006.

## BIBLIOGRAFIA

---

- [Har78] Michael A. Harrison. *Introduction to Formal Languages Theory*. Reading, Mass.: Addison-Wesley, 1978.
- [HE04] Hendrik Jan Hoogeboom and Joost Engelfriet. Pushdown Automata. In: *Formal Languages and Applications*, Martin-Vide, C., Mitrana, V. and Paun G. (eds). Springer, Berlim, 2004.
- [Her97] Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity, *Theoretical Computer Science* 181(1–2): 141–157, 1997.
- [HU79] John E. Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation*. Reading, Mass.: Addison-Wesley, 1979.
- [KGY00] Lila Kari, Greg Gloor and Sheng Yu. Using DNA to solve the Bounded Post Correspondence Problem. *Theoretical Computer Science*, 231(2):193–203, 2000.
- [Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Milner and J. Thatcher eds., pp. 85-104, Plenum Press, New York, 1972.
- [Kas65] Tadao Kasami. *An efficient recognition and syntax-analysis algorithm for context-free languages*. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA. 1965.
- [Koz97] Dexter C. Kozen. *Automata and Computability*. Springer, Heidelberg, 1997.
- [LV95] Ming Li and Paul M.B. Vitanyi. A new approach to formal language theory by Kolmogorov complexity, *SIAM J. Comput.*, 24(2):398–410, 1995.
- [Lin90] Peter Linz. *An Introduction to Formal Languages and Automata*. D.C. Heath and Company, Lexington-USA, 1990.
- [LSS79] Cláudio L. Lucchesi, Imre Simon, Istvan Simon, Janos Simon e Tomasz Kowaltowski. *Aspectos Teóricos da Computação*. IMPA, Rio de Janeiro-RJ, (Projeto Euclides 8) 1979
- [Mak03] Erkki Mäkinen. Inferring Finite Transducer. *Journal Brazilian Computational Society* 9(1):5–8, 2003.
- [Mar91] John C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, Inc., New York, 1991.
- [Lof70] Per Martin-Löf. *Notes on Constructive Mathematics*. Almqvist & Wiksell, Stockholm, 1970.
- [Men05] Paulo B. Menezes. *Linguagens Formais e Autômatos* (5a edição). Sagra-Luzzatto, Porto Alegre, 2005.
- [Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, USA, 1994 (reprinted with correction, 1995).
- [PJ06] Simon Perdrix and Philippe Jorrand. Classically-controlled Quantum Computation. *Electronic Notes in Theoretical Computer Science*, 135:119–128, 2006.

## BIBLIOGRAFIA

---

- [Phi92] Iain C.C. Phillips. Recursion Theory. In *Handbook of Logic in Computer Science*, Vol. 1. S. Abramsky, Dov M. Gabbay and T.S.E. Maibaum Eds., Oxford University Press, New York, 1992.
- [Pri97] Carlos A. di Prisco. *Una Introducción a la Teoría de Conjuntos y los Fundamentos de las Matemáticas*. Centro de Lógica, Epistemologia e História da Ciência - UNICAMP, coleção CLE Vol. 20, Campinas-SP, 1997.
- [Ros67] Arnold L. Rosenberg. A Machine Realization of the Linear Context-Free Languages. *Information and Control* 10: 175–188, 1967.
- [Rot96] Paul W.K. Rothemund. A DNA and Restriction Enzyme Implementation of Turing Machine. In *DNA Based Computers: Proceedings of the DIMACS Workshop*, April 4, 1995, Princeton University. R.J. Lipton, and E.B. Baum (Eds.); American Mathematical Society: Providence, RI, 75–119, 1996.
- [Sal73] Arto Salomaa. *Formal Language*. Academic Press, Inc., Orlando-Florida, USA, 1973.
- [SY94] Kai Salomaa and Sheng Yu. Measures of nondeterminism for pushdown automata, *Journal of Computer and System Sciences* 49(2): 362–374, 1994.
- [SB04] Regivan H.N. Santiago e Benjamín C. Bedregal. *Computabilidade: Os limites da computação*. Sociedade Brasileira de Matemática Aplicada e Computacional, São Carlos-SP, (Notas em Matemática Aplicada, Vol. 11) 2004.
- [Sav73] Walter J. Savitch. A note on multihead automata and context-sensitive languages. *Acta Informatica* 2(3):249–252, 1973.
- [Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Co. 1997.
- [Smi94] Carl H. Smith. *A recursive Introduction to the Theory of Computation*. Springer-Verlag, New York, 1994.
- [Smi95] Warren D. Smith. DNA Computers in vivo and in vitro. In *DNA Based Computers: Proceedings of the DIMACS Workshop*, April 4, 1995, Princeton University. R.J. Lipton and E.B. Baum (Eds.), American Mathematical Society: Providence, RI, 121–185, 1996.
- [Sob87] Jacob Z. Sobrinho. Aspectos da Tese de Church-Turing. *Matemática Universitária*, 6: 1–23, 1987.
- [Sup60] Patrick Suppes. *Axiomatic Set Theory*. D. Van Nostrand Company, Inc., USA, 1960.
- [Tho73] William J.D. Thomas. About some standard arguments for Church’s Thesis. In: Radu Bogden and Ilkka Inluoto (eds), *Logic, Language and Probability*. Dordrecht: D. Reidel, pages 55–65, 1973
- [VS81] Dirk Vermeir and Walter J. Savitch. On the amount of nondeterminism in pushdown automata, *Fundamenta Informaticae* 4:401–418, 1981.
- [Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM* 40:80–91, 1997.

## BIBLIOGRAFIA

---

- [Wei00] Klaus Weihrauch. *Computable Analysis: an introduction*. Springer Verlag, Berlim, 2000.
- [Wie04] Jirí Wiedermann. Characterizing the super-Turing computing power and efficiency of classical fuzzy Turing machines. *Theoret. Comput. Sci.* 317:61–69, 2004.
- [You67] Daniel H. Younger. Recognition and parsing of context-free languages in time  $n^3$ . *Information and Control* 10(2): 189–208, 1967.
- [Yu89] Sheng Yu. A pumping lemma for deterministic context-free languages. *Information Processing Letters*, 31:47–51, 1989.
- [Zad68] Lotfi A. Zadeh. Fuzzy algorithms. *Information and Control*, 2:94–102, 1968.

# Índice

- $DTempo(T(n))$ , 244  
 $L_{Ack}$ , 234  
 $NTempo(T(n))$ , 244  
 $\lambda$ -produções, 123  
 $\lambda$ -aresta, 29  
 $\lambda$ -fecho, 31  
 $\lambda$ -produção, 123  
 $\lambda$ -transição, 29, 144  
árvore, 8  
    altura, 8  
    de derivação, 104  
    parcial, 105  
    ordenada, 104  
AFD, 21  
    isomorfismo, 47  
    mínimo, 47  
AFN, 28  
ALL  
    determinísticos, 241  
APD, 142, 163  
APN, 144  
    livre de não-determinismo explícito, 161  
BNF, 115  
 $DEspaço(S(n))$ , 244  
 $NEspaço(S(n))$ , 244  
alfabeto, 10  
    da fita, 178  
    da pilha, 142, 144  
    de entrada, 21  
    fecho estrela, 11  
    fecho positivo, 11  
algoritmo, 194, 197  
    CYK, 113  
    de pertinência, 110  
    Early, 113  
análise, 110  
    de pesquisa exaustiva, 111  
pesquisa exaustiva, 111  
aresta  
    chegada, 7  
    saída, 7  
arestas, 7  
armazenamento temporário, 17  
autômato, 17  
    com pilha  
        determinístico, 142  
    com pilha não-determinístico, 144  
        linguagem aceita, 147  
finito, 21  
    determinístico, 21  
    não-determinístico, 28  
limitado linearmente, 230  
linear não-determinístico, 70  
reconhecedor, 21  
autômato com pilha determinístico, 163  
backtracking, 35  
cadeia, 10  
    comprimento, 10  
    concatenação, 10  
    prefixo, 10  
    reversa, 10  
    sufixo, 10  
    vazia, 10  
caminho, 7  
    comprimento, 7  
ciclo, 7  
    simples, 8  
Classe de complexidade  
    NP, 248  
    NP-espaço, 249  
    P, 248  
    P-Espaço, 249  
classe de equivalência, 46

- codificação de Gödel, 202  
complexidade, 242  
    computacional, 242  
    de espaço, 243  
    de tempo, 243  
    intrínseca, 242  
computável, 186  
computação  
    máquina de Turing, 182  
computação analógica, 205  
computação com DNA, 205  
computação mecânica, 196  
computação molecular, 205  
computação quântica, 205  
computação realística, 249  
computabilidade, 253  
configuração, 17  
    de morte, 35  
    de parada, 182  
conjunto, 3  
    contável, 203  
    das partes, 5  
    finito, 4  
    infinito, 4  
    potência, 5  
    universo, 3  
    vazio, 3  
conjuntos disjuntos, 4  
contradomínio, 5  
decibilidade, 253, 254  
derivação, 15  
    mais à direita, 104  
    mais à esquerda, 104  
descrição instantânea, 146, 181, 182  
determinístico, 23  
diagonalização, 216  
diagrama de bloco, 190  
domínio, 5  
  
estado  
    acessível, 44  
    equivalência, 44  
    inicial, 21  
    mesmo tipo, 44  
estados  
    de aceitação, 21  
  
de morte, 24  
finais, 21  
internos, 17, 21  
expressão regular, 57  
    equivalência, 61  
linguagem, 58  
primitiva, 57  
semântica, 58  
  
família de conjuntos, 4  
fecho, 83  
fecho de conjuntos, 7  
fecho estrela, 11, 12  
fecho positivo, 11, 13  
fita de entrada, 17  
folhas, 8  
forma de Backus-Naur, 115  
forma normal, 128  
    APN, 156  
    de Chomsky, 128  
    de Greibach, 132  
    pre-Chomsky, 129  
    pseudo-Greibach, 151  
    SC, 225  
forma normal linear, 70  
formas sentenciais, 15  
função, 5  
    computável, 242  
    de Ackermann, 233  
    de transição, 17, 21  
        estendida, 22, 30  
    gráfico, 6  
    parcial, 5  
        não total, 5  
    recursiva, 234  
        primitiva, 234  
        total, 5  
função de transição, 142, 144  
  
grafo, 7  
    de dependência, 120, 127  
    de transição, 22  
    dirigido, 7  
Gramática  
    simples, 132  
gramática, 14  
    ambigua, 107

## ÍNDICE

---

- esquerda-direita direita recursiva, 115
- esquerda-direita esquerda recursiva, 115
- esquerda-recursiva, 119
- irrestrita, 218
- linear, 69
  - à direita, 70
  - à esquerda, 70
- livre do contexto, 101
- LL, 115
- LR, 115
- regular, 70
  - sensível ao contexto, 223
  - simples, 114
- grau de não-determinismo, 161
- hierarquia de Chomsky, 240
- hipercomputabilidade, 205
- homomorfismo, 85
  - estendido, 86
- imagem, 5
  - homomorfa, 86
- indecibilidade, 254
- lógica difusa, 205
- laço infinito, 180
- lema do bombeamento, 91, 169
- linguagem, 11
  - aceita por um AFD, 23
  - aceita por um AFN, 34
  - complemento, 12
  - concatenação, 12
  - de Ackermann, 234
  - fecho estrela, 12
  - fecho positivo, 13
  - formal, 9
  - gerada, 15
  - inerentemente ambígua, 109
  - linear, 70, 240
    - autômato, 70
    - determinística, 241
    - forma normal, 70
  - livre de  $\lambda$ , 123
  - livre do contexto, 101
  - livre do contexto determinístico, 240
  - não ambígua, 109
  - natural, 9
- prefixos, 13
- reconhecida por um AFD, 23
- reconhecida por um AFN, 34
- recursiva, 214
- recursivamente enumerável, 213
- regular, 27
- reversa, 13
- sensível ao contexto, 223
  - determinística, 241
- sufixos, 13
- máquina de Turing, 177
  - com opção de permanência, 198
  - padrão, 177
  - universal, 202
  - difusa, 205
  - fita semi-infinita, 198
  - múltiplas fitas, 199
  - molecular, 206
  - multidimensional, 200
  - não-determinística, 201
  - off-line, 199
  - persistente, 205
  - quântica, 205
- macroinstrução, 191
- minimização de estados, 44
- movimento, 17
- números de Gödel, 202
- números randômicos, 205
- não-determinismo, 35
  - implícito, 164
- não-determinismo explícito, 161
- ordem de magnitude, 243
- ordem própria, 248
- palíndromo, 72, 240
- par ordenado, 5, 6
- pilha
  - alfabeto, 142
  - símbolo de início, 142
- problema
  - co-semi-decidível, 271
  - da correspondência de Post, 263
    - modificado, 264
  - da divergência, 271

## ÍNDICE

---

- da parada, 254
- duplo, 271
- solução, 255
- de decisão, 254
- decidível, 254
- intratável, 249
- polinomial, 248
- semi-decidível, 271
- procedimento de enumeração, 203
- procedimento efetivo, 194
- produção, 14
  - esquerda-recursiva, 118, 119
  - inútil, 120
  - unitária, 125
- produto cartesiano, 6
- pseudo-código, 191
- rótulo, 8
- raiz, 8
- recursão primitiva, 210
- redução de um problema, 258
- reflexividade, 6
- regra geral de substituição, 116
- relação, 5, 6
  - de equivalência, 6
- relação de equivalência, 45
- representação padrão, 88
- S-condição, 114
- S-gramáticas, 114, 132
- símbolo terminal, 14
- sentença, 11, 15
- simetria, 6
- Simulador de autômatos, 89
- solução-CP, 263
- solução-CPM, 264
- sub-expressão regular, 58
- subconjunto, 3
  - próprio, 3
- subprograma, 192
- sufixo, 97, 309
- teorema
  - de Rice, 263
  - de Savitch, 249
- teorema de Savitch, 246
- teoria da complexidade, 242
- tese
  - de Church, 196
  - de Cook-Karp, 249
  - de Turing, 195
- tradutor, 186
- tradutores finitos, 70
- transição determinística, 163
- transitividade, 6
- trilha, 7
- Turing-computável, 186
- união
  - disjunta, 97
- unidade de controle, 17
- vértice, 7
  - filho, 8
  - pai, 8
- variável
  - útil, 120
  - inútil, 120
- variável anulável, 123
- variável de início, 14